

A Markdown Interpreter for T_EX

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

Version 3.7.1-0-g8e726800
2024-09-30

Contents

1	Introduction	1	3	Implementation	147
1.1	Requirements	2	3.1	Lua Implementation	147
1.2	Feedback	6	3.2	Plain T _E X Implementation	359
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation	384
2	Interfaces	7	3.4	ConT _E Xt Implementation	416
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	53			
2.3	L ^A T _E X Interface	135			
2.4	ConT _E Xt Interface	143			
				References	424
				Index	425

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	Various formats of mathematical formulae	141
5	The banner of the Markdown package	142

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. Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

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")
```

Selene Unicode A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and note tags to the lower case. Selene Unicode is included in all releases of Lua_{TeX} (T_EX Live \geq 2008).

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

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

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

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

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

```
17 ;(function()
```

If Kpathsea has not been loaded before or if LuaT_EX has not yet been initialized, configure Kpathsea on top of loading it. Since ConT_EXt 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.

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

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

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

```
25 hard lua-uni-algos
26 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 `jekyllData` option is enabled. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

```
27 # hard lua-tinyyaml # TODO: Uncomment after TeX Live 2022 deprecation.
```

1.1.2 Plain T_EX Requirements

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

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

```
28 hard l3kernel
29 \unprotect
```

```

30 \ifx\ExplSyntaxOn\undefined
31   \input expl3-generic
32 \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.

```
33 hard lt3luabridge
```

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

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

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

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

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

1.1.3 L^ATeX Requirements

The L^ATeX part of the package requires that the L^ATeX 2 ϵ format is loaded,

```

34 \NeedsTeXFormat{LaTeX2e}
35 \RequirePackage{expl3}

```

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

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or L^ATeX themes (see Section 2.3.3) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
36 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.

```
37 soft graphics
```

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

```
38 soft paralist
```

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L^AT_EX theme (see Section 2.3.3).

```
39 soft latex
```

```
40 soft epstopdf-pkg # required by `latex`
```

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

```
41 soft fancyvrb
```

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

```
42 soft csvsimple
```

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

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

gobble A package that provides the `\@gobblethree` T_EX command that is used in the default renderer prototype for citations. The package is included in T_EXLive \geq 2016.

```
45 soft gobble
```

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

```
46 soft amsmath
```

```
47 soft amssymb
```

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` L^AT_EX theme, see Section 2.3.3.

```
48 soft catchfile
```

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

49 `soft grffile`

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.8, and also in the default renderer prototype for identifier attributes.

50 `soft etoolbox`

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

51 `soft soul`

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.

52 `soft ltxcmds`

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

53 `soft verse`

1.1.4 ConT \E Xt Prerequisites

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

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

1.2 Feedback

Please use the Markdown project page on GitHub⁴ to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T\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 [2] is gratefully acknowledged.

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

The $\text{T}_{\text{E}}\text{X}$ 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.

```
54 local M = {metadata = metadata}
```

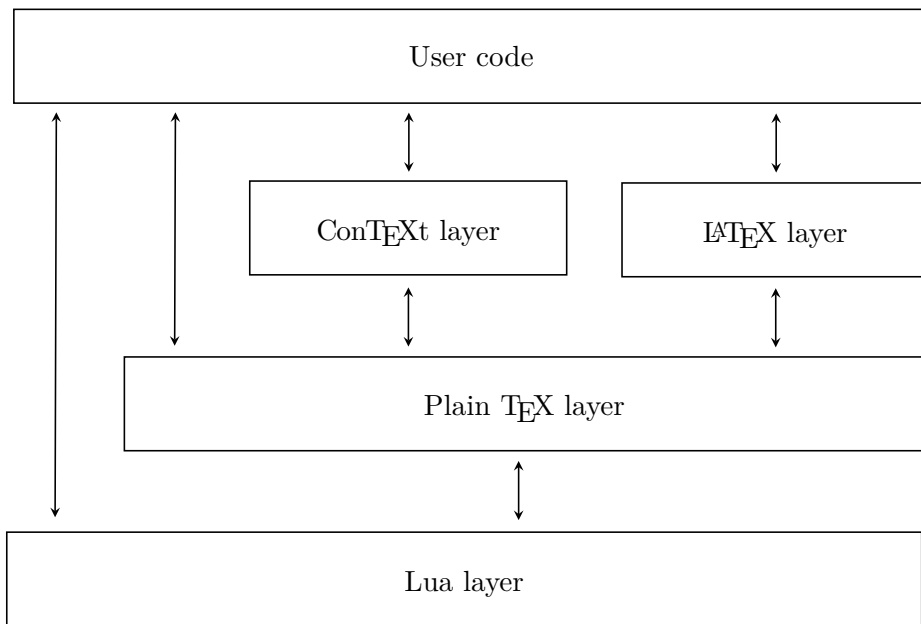


Figure 1: A block diagram of the Markdown package

</lua, lua-loader, lua-unicode-data> <*lua>

2.1.1 Conversion from Markdown to Plain T_EX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain T_EX according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

The following example Lua code converts the markdown string `Hello *world*!` to a T_EX output using the default options and prints the T_EX output:

```
local md = require("markdown")
local convert = md.new()
print(convert("Hello *world*!"))
```

2.1.2 User-Defined Syntax Extensions

For the purpose of user-defined syntax extensions, the Lua interface also exposes the `reader` object, which performs the lexical and syntactic analysis of markdown text and which exposes the `reader->insert_pattern` and `reader->add_special_character` methods for extending the PEG grammar of markdown.

The read-only `walkable_syntax` hash table stores those rules of the PEG grammar of markdown that can be represented as an ordered choice of terminal symbols. These rules can be modified by user-defined syntax extensions.

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

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.

```
87 local defaultOptions = {}
```

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

```
88 \ExplSyntaxOn
89 \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.

```
90 \prop_new:N \g_@@_lua_option_types_prop
91 \prop_new:N \g_@@_default_lua_options_prop
92 \seq_new:N \g_@@_option_layers_seq
93 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
94 \seq_gput_right:NV
95   \g_@@_option_layers_seq
96   \c_@@_option_layer_lua_tl
97 \cs_new:Nn
98   \@@_add_lua_option:nnn
99   {
100     \@@_add_option:Vnnn
101       \c_@@_option_layer_lua_tl
102       { #1 }
103       { #2 }
104       { #3 }
105   }
106 \cs_new:Nn
107   \@@_add_option:nnnn
108   {
109     \seq_gput_right:cn
110       { g_@@_ #1 _options_seq }
111       { #2 }
112     \prop_gput:cnn
113       { g_@@_ #1 _option_types_prop }
114       { #2 }
115       { #3 }
```

```

116     \prop_gput:cnn
117     { g_@@_default_ #1 _options_prop }
118     { #2 }
119     { #4 }
120     \@@_typecheck_option:n
121     { #2 }
122 }
123 \cs_generate_variant:Nn
124 \@@_add_option:nnnn
125 { Vnnn }
126 \tl_const:Nn \c_@@_option_value_true_tl { true }
127 \tl_const:Nn \c_@@_option_value_false_tl { false }
128 \cs_new:Nn \@@_typecheck_option:n
129 {
130     \@@_get_option_type:nN
131     { #1 }
132     \l_tmpa_tl
133     \str_case_e:Vn
134     \l_tmpa_tl
135     {
136         { \c_@@_option_type_boolean_tl }
137         {
138             \@@_get_option_value:nN
139             { #1 }
140             \l_tmpa_tl
141             \bool_if:nF
142             {
143                 \str_if_eq_p:VV
144                 \l_tmpa_tl
145                 \c_@@_option_value_true_tl ||
146                 \str_if_eq_p:VV
147                 \l_tmpa_tl
148                 \c_@@_option_value_false_tl
149             }
150             {
151                 \msg_error:nnnV
152                 { markdown }
153                 { failed-typecheck-for-boolean-option }
154                 { #1 }
155                 \l_tmpa_tl
156             }
157         }
158     }
159 }
160 \msg_new:nnn
161 { markdown }
162 { failed-typecheck-for-boolean-option }

```

```

163 {
164   Option~#1~has~value~#2,~
165   but~a~boolean~(true~or~false)~was~expected.
166 }
167 \cs_generate_variant:Nn
168   \str_case_e:nn
169   { Vn }
170 \cs_generate_variant:Nn
171   \msg_error:nnnn
172   { nnnV }
173 \seq_new:N
174   \g_@@_option_types_seq
175 \tl_const:Nn
176   \c_@@_option_type_clist_tl
177   { clist }
178 \seq_gput_right:NV
179   \g_@@_option_types_seq
180   \c_@@_option_type_clist_tl
181 \tl_const:Nn
182   \c_@@_option_type_counter_tl
183   { counter }
184 \seq_gput_right:NV
185   \g_@@_option_types_seq
186   \c_@@_option_type_counter_tl
187 \tl_const:Nn
188   \c_@@_option_type_boolean_tl
189   { boolean }
190 \seq_gput_right:NV
191   \g_@@_option_types_seq
192   \c_@@_option_type_boolean_tl
193 \tl_const:Nn
194   \c_@@_option_type_number_tl
195   { number }
196 \seq_gput_right:NV
197   \g_@@_option_types_seq
198   \c_@@_option_type_number_tl
199 \tl_const:Nn
200   \c_@@_option_type_path_tl
201   { path }
202 \seq_gput_right:NV
203   \g_@@_option_types_seq
204   \c_@@_option_type_path_tl
205 \tl_const:Nn
206   \c_@@_option_type_slice_tl
207   { slice }
208 \seq_gput_right:NV
209   \g_@@_option_types_seq

```

```

210 \c_@@_option_type_slice_tl
211 \tl_const:Nn
212 \c_@@_option_type_string_tl
213 { string }
214 \seq_gput_right:NV
215 \g_@@_option_types_seq
216 \c_@@_option_type_string_tl
217 \cs_new:Nn
218 \@@_get_option_type:nN
219 {
220 \bool_set_false:N
221 \l_tmpa_bool
222 \seq_map_inline:Nn
223 \g_@@_option_layers_seq
224 {
225 \prop_get:cnNT
226 { g_@@_ ##1 _option_types_prop }
227 { #1 }
228 \l_tmpa_tl
229 {
230 \bool_set_true:N
231 \l_tmpa_bool
232 \seq_map_break:
233 }
234 }
235 \bool_if:nF
236 \l_tmpa_bool
237 {
238 \msg_error:nnn
239 { markdown }
240 { undefined-option }
241 { #1 }
242 }
243 \seq_if_in:NVF
244 \g_@@_option_types_seq
245 \l_tmpa_tl
246 {
247 \msg_error:nnnV
248 { markdown }
249 { unknown-option-type }
250 { #1 }
251 \l_tmpa_tl
252 }
253 \tl_set_eq:NN
254 #2
255 \l_tmpa_tl
256 }

```

```

257 \msg_new:nnn
258   { markdown }
259   { unknown-option-type }
260   {
261     Option~#1~has~unknown~type~#2.
262   }
263 \msg_new:nnn
264   { markdown }
265   { undefined-option }
266   {
267     Option~#1~is~undefined.
268   }
269 \cs_new:Nn
270   \@@_get_default_option_value:nN
271   {
272     \bool_set_false:N
273       \l_tmpa_bool
274     \seq_map_inline:Nn
275       \g_@@_option_layers_seq
276       {
277         \prop_get:cnNT
278           { g_@@_default_ ##1 _options_prop }
279           { #1 }
280           #2
281           {
282             \bool_set_true:N
283               \l_tmpa_bool
284             \seq_map_break:
285           }
286         }
287     \bool_if:nF
288       \l_tmpa_bool
289       {
290         \msg_error:nnn
291           { markdown }
292           { undefined-option }
293           { #1 }
294       }
295   }
296 \cs_new:Nn
297   \@@_get_option_value:nN
298   {
299     \@@_option_tl_to_csname:nN
300       { #1 }
301     \l_tmpa_tl
302     \cs_if_free:cTF
303       { \l_tmpa_tl }

```

```

304     {
305     \@@_get_default_option_value:nN
306     { #1 }
307     #2
308     }
309     {
310     \@@_get_option_type:nN
311     { #1 }
312     \l_tmpa_tl
313     \str_if_eq:NNTF
314     \c_@@_option_type_counter_tl
315     \l_tmpa_tl
316     {
317     \@@_option_tl_to_csname:nN
318     { #1 }
319     \l_tmpa_tl
320     \tl_set:Nx
321     #2
322     { \the \cs:w \l_tmpa_tl \cs_end: }
323     }
324     {
325     \@@_option_tl_to_csname:nN
326     { #1 }
327     \l_tmpa_tl
328     \tl_set:Nv
329     #2
330     { \l_tmpa_tl }
331     }
332     }
333   }
334 \cs_new:Nn \@@_option_tl_to_csname:nN
335   {
336   \tl_set:Nn
337   \l_tmpa_tl
338   { \str_uppercase:n { #1 } }
339   \tl_set:Nx
340   #2
341   {
342   markdownOption
343   \tl_head:f { \l_tmpa_tl }
344   \tl_tail:n { #1 }
345   }
346   }

```

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.

```

347 \cs_new:Nn \@@_with_various_cases:nn
348 {
349   \seq_clear:N
350   \l_tmpa_seq
351   \seq_map_inline:Nn
352   \g_@@_cases_seq
353   {
354     \tl_set:Nn
355     \l_tmpa_tl
356     { #1 }
357     \use:c { ##1 }
358     \l_tmpa_tl
359     \seq_put_right:NV
360     \l_tmpa_seq
361     \l_tmpa_tl
362   }
363   \seq_map_inline:Nn
364   \l_tmpa_seq
365   { #2 }
366 }

```

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

```

367 \cs_new:Nn \@@_with_various_cases_break:
368 {
369   \seq_map_break:
370 }

```

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

```

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

```



```

389     \tl_set:Nx
390         #1
391     { #1 }
392 }
393 \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.

```

394 \@@_add_lua_option:nnn
395   { eagerCache }
396   { boolean }
397   { true }
398 defaultOptions.eagerCache = true

```

`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.

```
399 \@@_add_lua_option:nnn
400   { singletonCache }
401   { boolean }
402   { true }

403 defaultOptions.singletonCache = true

404 local singletonCache = {
405   convert = nil,
406   options = nil,
407 }
```

`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.

```
408 \@@_add_lua_option:nnn
409   { unicodeNormalization }
410   { boolean }
411   { true }

412 defaultOptions.unicodeNormalization = true
```

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

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

`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.

```
413 \@@_add_lua_option:nnn
414   { unicodeNormalizationForm }
415   { string }
416   { nfc }
417 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.

```
418 \@@_add_lua_option:nnn
419   { cacheDir }
420   { path }
421   { \markdownOptionOutputDir / _markdown_\jobname }
422 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.

```
423 \@@_add_lua_option:nnn
424   { contentBlocksLanguageMap }
425   { path }
426   { markdown-languages.json }
427 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.

```
428 \@@_add_lua_option:nnn
429   { debugExtensionsFileName }
430   { path }
431   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
432 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

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

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

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

```
433 \@@_add_lua_option:nnn
434   { frozenCacheFileName }
435   { path }
436   { \markdownOptionCacheDir / frozenCache.tex }
437 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`.

```
438 \@@_add_lua_option:nnn
439   { autoIdentifiers }
440   { boolean }
441   { false }
442 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.

```
443 \@@_add_lua_option:nnn
444   { blankBeforeBlockquote }
445   { boolean }
446   { false }
447 defaultOptions.blankBeforeBlockquote = false
```

`blankBeforeCodeFence=true, false` default: false

true Require a blank line between a paragraph and the following fenced code block.

false Do not require a blank line between a paragraph and the following fenced code block.

```
448 \@@_add_lua_option:nnn
449   { blankBeforeCodeFence }
450   { boolean }
451   { false }
452 defaultOptions.blankBeforeCodeFence = false
```

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

`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.

```
453 \@@_add_lua_option:nnn
454   { blankBeforeDivFence }
455   { boolean }
456   { false }

457 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.

```
458 \@@_add_lua_option:nnn
459   { blankBeforeHeading }
460   { boolean }
461   { false }

462 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.

```
463 \@@_add_lua_option:nnn
464   { blankBeforeList }
465   { boolean }
466   { false }

467 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.

```
468 \@@_add_lua_option:nnn
469   { bracketedSpans }
470   { boolean }
471   { false }

472 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.

```
473 \@@_add_lua_option:nnn
474   { breakableBlockquotes }
475   { boolean }
476   { true }

477 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.

```
478 \@@_add_lua_option:nnn
479   { citationNbsps }
480   { boolean }
481   { true }

482 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.

```
483 \@@_add_lua_option:nnn
484 { citations }
485 { boolean }
486 { false }
487 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.``
```

```
488 \@@_add_lua_option:nnn
489 { codeSpans }
490 { boolean }
491 { true }
492 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 [3]:

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

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

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

```
498 \@@_add_lua_option:nnn
499   { contentLevel }
500   { string }
501   { block }
502 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.

```
503 \@@_add_lua_option:nnn
504   { debugExtensions }
505   { boolean }
506   { false }

507 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.

```
508 \@@_add_lua_option:nnn
509   { definitionLists }
510   { boolean }
511   { false }

512 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.

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

```
518 \@@_add_lua_option:nnn
519 { expectJekyllData }
520 { boolean }
521 { false }

522 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:

```

523 metadata.user_extension_api_version = 2
524 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.

```

525 \cs_generate_variant:Nn
526 \@@_add_lua_option:nnn
527 { nnV }
528 \@@_add_lua_option:nnV
529 { extensions }
530 { clist }
531 \c_empty_clist
532 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

true Enable the Pandoc fancy list syntax extension¹¹:

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

false Disable the Pandoc fancy list syntax extension.

```
533 \@@_add_lua_option:nnn
534 { fancyLists }
535 { boolean }
536 { false }
537 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.

```
538 \@@_add_lua_option:nnn
539 { fencedCode }
540 { boolean }
541 { true }
542 defaultOptions.fencedCode = true
```

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

`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.

```
543 \@@_add_lua_option:nnn
544 { fencedCodeAttributes }
545 { boolean }
546 { false }

547 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.

```
548 \@@_add_lua_option:nnn
549 { fencedDivs }
550 { boolean }
551 { false }

552 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 \TeX document that contains markdown documents without invoking Lua using the `frozenCache` plain \TeX option. As a result, the plain \TeX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```
553 \@@_add_lua_option:nnn
554   { finalizeCache }
555   { boolean }
556   { false }

557 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>`

default: `0`

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

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

```
558 \@@_add_lua_option:nnn
559   { frozenCacheCounter }
560   { counter }
561   { 0 }

562 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](#).

```
563 \@@_add_lua_option:nnn
564   { gfmAutoIdentifiers }
565   { boolean }
566   { false }

567 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.

```
568 \@@_add_lua_option:nnn
569   { hashEnumerators }
570   { boolean }
571   { false }

572 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.

```
573 \@@_add_lua_option:nnn
574   { headerAttributes }
575   { boolean }
576   { false }

577 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.

```
578 \@@_add_lua_option:nnn
579   { html }
580   { boolean }
581   { true }

582 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:

```
\[distance[i] =
 \begin{dcases}
 a & b \\
 c & d
 \end{dcases}
\]
```

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

`inlineCodeAttributes=true, false`

default: false

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

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

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

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

```
588 \@@_add_lua_option:nnn
589 { inlineCodeAttributes }
590 { boolean }
591 { false }

592 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

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

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

`false` Disable the Pandoc inline note syntax extension.

```
593 \@@_add_lua_option:nnn
594 { inlineNotes }
595 { boolean }
596 { false }

597 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

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

```

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

 It consists of two paragraphs.

```

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

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

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

```
598 \@@_add_lua_option:nnn
599 { jekyllData }
600 { boolean }
601 { false }
602 defaultOptions.jekyllData = false
```

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

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

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

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

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

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

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

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

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

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

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

---

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

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

`mark=true, false` default: false

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

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

`false` Disable the Pandoc mark syntax extension.

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

`notes=true, false` default: false

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

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

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

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

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

```
 { some.code }
```

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

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

`false` Disable the Pandoc note syntax extension.

```
618 \@@_add_lua_option:nnn
619 { notes }
620 { boolean }
621 { false }
622 defaultOptions.notes = false
```

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

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

`pipeTables=true, false`

default: false

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

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

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

```
623 \@@_add_lua_option:nnn
624 { pipeTables }
625 { boolean }
626 { false }

627 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.

```
628 \@@_add_lua_option:nnn
629 { preserveTabs }
630 { boolean }
631 { true }

632 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

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

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

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

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

²²See 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.

```

633 \@@_add_lua_option:nnn
634   { rawAttribute }
635   { boolean }
636   { false }

637 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false`

default: `false`

`true` Enable relative references²³ 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.

```

638 \@@_add_lua_option:nnn
639   { relativeReferences }
640   { boolean }
641   { false }

642 defaultOptions.relativeReferences = false

```

²³See <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.

```
643 \@@_add_lua_option:nnn
644   { shiftHeadings }
645   { number }
646   { 0 }

647 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.

```
648 \@@_add_lua_option:nnn
649   { slice }
650   { slice }
651   { ^-$ }

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

`smartEllipses=true, false` default: false

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

`false` Preserve all ellipses in the input.

```
653 \@@_add_lua_option:nnn
654 { smartEllipses }
655 { boolean }
656 { false }

657 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

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

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

```
658 \@@_add_lua_option:nnn
659 { startNumber }
660 { boolean }
661 { true }

662 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension²⁴:

This ~~is deleted text.~~

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

```
663 \@@_add_lua_option:nnn
664 { strikeThrough }
665 { boolean }
666 { false }

667 defaultOptions.strikeThrough = false
```

²⁴See <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.

```
668 \@@_add_lua_option:nmn
669   { stripIndent }
670   { boolean }
671   { false }
672 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

true Enable the Pandoc subscript syntax extension²⁵:

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

false Disable the Pandoc subscript syntax extension.

```
673 \@@_add_lua_option:nmn
674   { subscripts }
675   { boolean }
676   { false }
677 defaultOptions.subscripts = false
```

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

`superscripts=true, false`

default: false

true Enable the Pandoc superscript syntax extension²⁶:

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

false Disable the Pandoc superscript syntax extension.

```
678 \@@_add_lua_option:nnn
679 { superscripts }
680 { boolean }
681 { false }

682 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.

```
683 \@@_add_lua_option:nnn
684 { tableAttributes }
685 { boolean }
686 { false }

687 defaultOptions.tableAttributes = false
```

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

`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension²⁷ 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.

```
688 \@@_add_lua_option:nnn
689 { tableCaptions }
690 { boolean }
691 { false }

692 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

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

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

`false` Disable the Pandoc task list syntax extension.

```
693 \@@_add_lua_option:nnn
694 { taskLists }
695 { boolean }
696 { false }

697 defaultOptions.taskLists = false
```

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

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

`texComments=true, false`

default: false

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

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

Always enabled when `hybrid` is enabled.

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

```
698 \@@_add_lua_option:nnn
699 { texComments }
700 { boolean }
701 { false }
702 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

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

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

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

```
703 \@@_add_lua_option:nnn
704 { texMathDollars }
705 { boolean }
706 { false }
707 defaultOptions.texMathDollars = false
```

---

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

`texMathDoubleBackslash=true, false` default: false

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

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

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

```
708 \\@@_add_lua_option:nnn
709 { texMathDoubleBackslash }
710 { boolean }
711 { false }

712 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

```
713 \\@@_add_lua_option:nnn
714 { texMathSingleBackslash }
715 { boolean }
716 { false }

717 defaultOptions.texMathSingleBackslash = false
```

`tightLists=true, false` default: true

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

---

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

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

```

- This is
- a tight
- unordered list.

- This is

 not a tight

- unordered list.

```

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

```

718 \@@_add_lua_option:nmn
719 { tightLists }
720 { boolean }
721 { true }

722 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.

```

723 \@@_add_lua_option:nmn
724 { underscores }
725 { boolean }
726 { true }
727 \ExplSyntaxOff

728 defaultOptions.underscores = true

```

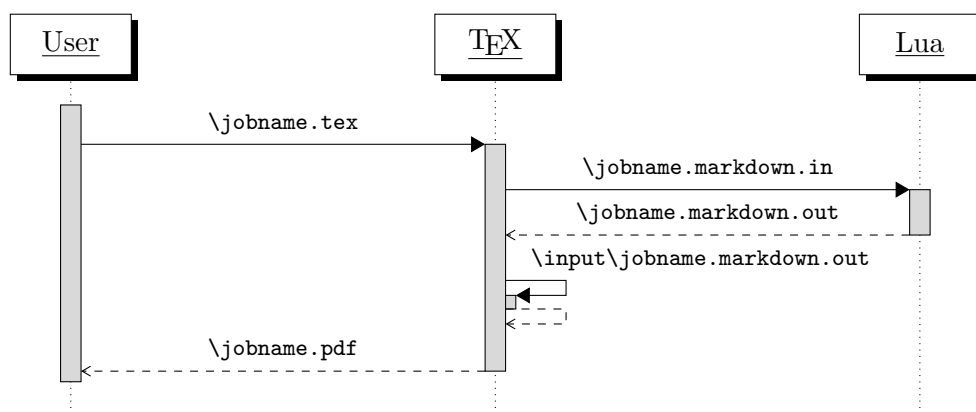


### 2.1.7 Command-Line Interface

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

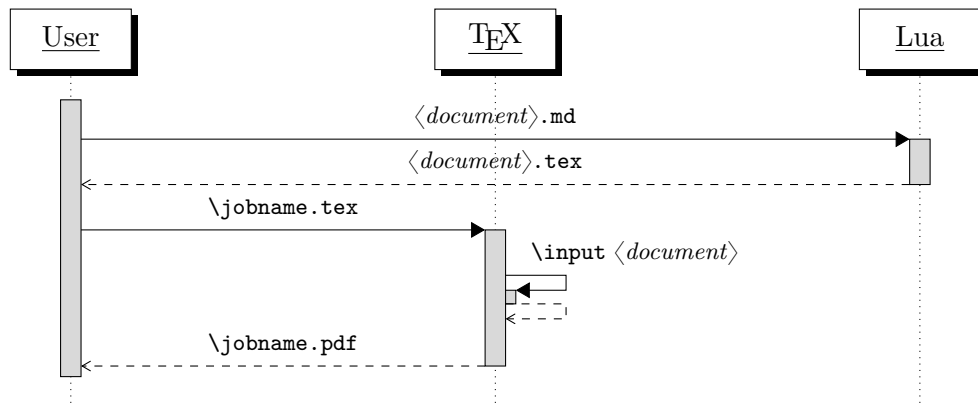
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{T}_{\text{E}}\text{X}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{T}_{\text{E}}\text{X}$  engine includes a Lua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

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



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

```
729
730 local HELP_STRING = [[
731 Usage: texlua]] .. arg[0] .. [[[OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
732 where OPTIONS are documented in the Lua interface section of the
733 technical Markdown package documentation.
734
735 When OUTPUT_FILE is unspecified, the result of the conversion will be
736 written to the standard output. When INPUT_FILE is also unspecified, the
737 result of the conversion will be read from the standard input.
738
739 Report bugs to: witiko@mail.muni.cz
740 Markdown package home page: <https://github.com/witiko/markdown>]]
741
```



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

```

742 local VERSION_STRING = [[
743 markdown-cli.lua (Markdown)]] .. metadata.version .. [[
744
745 Copyright (C)]] .. table.concat(metadata.copyright,
746 "\nCopyright (C) ") .. [[
747
748 License:]] .. metadata.license
749
750 local function warn(s)
751 io.stderr:write("Warning: " .. s .. "\n")
752 end
753
754 local function error(s)
755 io.stderr:write("Error: " .. s .. "\n")
756 os.exit(1)
757 end

```

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

```

758 local function camel_case(option_name)
759 local cased_option_name = option_name:gsub("_(%l)", function(match)
760 return match:sub(2, 2):upper()
761 end)
762 return cased_option_name
763 end
764
765 local function snake_case(option_name)
766 local cased_option_name = option_name:gsub("%l%u", function(match)

```

```

767 return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
768 end)
769 return cased_option_name
770 end
771
772 local cases = {camel_case, snake_case}
773 local various_case_options = {}
774 for option_name, _ in pairs(defaultOptions) do
775 for _, case in ipairs(cases) do
776 various_case_options[case(option_name)] = option_name
777 end
778 end
779
780 local process_options = true
781 local options = {}
782 local input_filename
783 local output_filename
784 for i = 1, #arg do
785 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.

```

786 if arg[i] == "--" then
787 process_options = false
788 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.

```

789 elseif arg[i]:match("=") then
790 local key, value = arg[i]:match("(.)=(.*)")
791 if defaultOptions[key] == nil and
792 various_case_options[key] ~= nil then
793 key = various_case_options[key]
794 end

```

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

```

795 local default_type = type(defaultOptions[key])
796 if default_type == "boolean" then
797 options[key] = (value == "true")
798 elseif default_type == "number" then
799 options[key] = tonumber(value)
800 elseif default_type == "table" then
801 options[key] = {}
802 for item in value:gmatch("[^,]+") do

```

```

803 table.insert(options[key], item)
804 end
805 else
806 if default_type ~= "string" then
807 if default_type == "nil" then
808 warn('Option "' .. key .. '" not recognized.')
809 else
810 warn('Option "' .. key .. '" type not recognized, ' ..
811 'please file a report to the package maintainer.')
812 end
813 warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
814 key .. '" as a string.')
815 end
816 options[key] = value
817 end
818 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.

```

819 elseif arg[i] == "--help" or arg[i] == "-h" then
820 print(HELP_STRING)
821 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.

```

822 elseif arg[i] == "--version" or arg[i] == "-v" then
823 print(VERSION_STRING)
824 os.exit()
825 end
826 end

```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a  $\text{T}_{\text{E}}\text{X}$  document.

```

827 if input_filename == nil then
828 input_filename = arg[i]

```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the  $\text{T}_{\text{E}}\text{X}$  document that will result from the conversion.

```

829 elseif output_filename == nil then
830 output_filename = arg[i]
831 else
832 error('Unexpected argument: "' .. arg[i] .. "'.')
833 end
834 ::continue::

```

```
835 end
```

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

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

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

```
\input hello
```

## 2.2 Plain TeX Interface

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

```
836 \def\markdownLastModified{((LASTMODIFIED))}%
```

```
837 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown

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

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

```
838 \let\markdownBegin\relax
```

```
839 \let\markdownEnd\relax
```

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

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

it will not be recognized as the end of the markdown string. As a corollary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

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

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

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

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

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

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

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

```
840 \let\markinline\relax
```

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

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

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

```
\input markdown
\markdownSetup{contentLevel=inline}
```

```
\markdownBegin
Hello **world** ...
\markdownEnd
\bye
```

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

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

```
841 \let\markdownInput\relax
```

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

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

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

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

```
842 \let\markdownEscape\relax
```

## 2.2.2 Options

The plain T<sub>E</sub>X options are represented by T<sub>E</sub>X 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 T<sub>E</sub>X interface.

To determine whether plain T<sub>E</sub>X is the top layer or if there are other layers above plain 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 plain T<sub>E</sub>X is the top layer.

```
843 \ExplSyntaxOn
844 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
845 \cs_generate_variant:Nn
846 \tl_const:Nn
847 { NV }
848 \tl_if_exist:NF
```

```

849 \c_@@_top_layer_tl
850 {
851 \tl_const:NV
852 \c_@@_top_layer_tl
853 \c_@@_option_layer_plain_tex_tl
854 }

```

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

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

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

```

856 \prop_new:N \g_@@_plain_tex_option_types_prop
857 \prop_new:N \g_@@_default_plain_tex_options_prop
858 \seq_gput_right:NV
859 \g_@@_option_layers_seq
860 \c_@@_option_layer_plain_tex_tl
861 \cs_new:Nn
862 \@@_add_plain_tex_option:nmn
863 {
864 \@@_add_option:Vmn
865 \c_@@_option_layer_plain_tex_tl
866 { #1 }
867 { #2 }
868 { #3 }
869 }

```

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

```

870 \cs_new:Nn
871 \@@_setup:n
872 {
873 \keys_set:nn
874 { markdown/options }
875 { #1 }
876 }
877 \cs_gset_eq:NN
878 \markdownSetup
879 \@@_setup:n

```

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.



```

880 \prg_new_conditional:Nnn
881 \@@_if_option:n
882 { TF, T, F }
883 {
884 \@@_get_option_type:nN
885 { #1 }
886 \l_tmpa_tl
887 \str_if_eq:NNF
888 \l_tmpa_tl
889 \c_@@_option_type_boolean_tl
890 {
891 \msg_error:nxxx
892 { markdown }
893 { expected-boolean-option }
894 { #1 }
895 { \l_tmpa_tl }
896 }
897 \@@_get_option_value:nN
898 { #1 }
899 \l_tmpa_tl
900 \str_if_eq:NNTF
901 \l_tmpa_tl
902 \c_@@_option_value_true_tl
903 { \prg_return_true: }
904 { \prg_return_false: }
905 }
906 \msg_new:nnn
907 { markdown }
908 { expected-boolean-option }
909 {
910 Option~#1~has~type~#2,~
911 but~a~boolean~was~expected.
912 }
913 \let\markdownIfOption=\@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `frozenCacheFileName` (frozen cache) that contains a mapping between an enumeration of the markdown documents in the plain  $\TeX$  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`.

```

914 \@@_add_plain_tex_option:nnn
915 { frozenCache }
916 { boolean }
917 { 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.

```

918 \@@_add_plain_tex_option:nnn
919 { inputTempFileName }
920 { path }
921 { \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.

```

922 \@@_add_plain_tex_option:nnn
923 { outputDir }
924 { path }
925 { . }

```

### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra

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

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

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

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

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

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

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

```
926 \@_add_plain_tex_option:nnn
927 { plain }
928 { boolean }
929 { false }
```

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

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

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

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

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

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

```

930 \@@_add_plain_tex_option:nnn
931 { noDefaults }
932 { boolean }
933 { false }

```

#### 2.2.2.4 Miscellaneous Options

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

```

934 \seq_gput_right:Nn
935 \g_@@_plain_tex_options_seq
936 { stripPercentSigns }
937 \prop_gput:Nnn
938 \g_@@_plain_tex_option_types_prop
939 { stripPercentSigns }
940 { boolean }
941 \prop_gput:Nnx
942 \g_@@_default_plain_tex_options_prop
943 { stripPercentSigns }
944 { false }

```

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

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

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

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

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

```

945 \cs_new:Nn
946 \@@_define_option_commands_and_keyvals:
947 {
948 \seq_map_inline:Nn

```

```

949 \g_@@_option_layers_seq
950 {
951 \seq_map_inline:cn
952 { g_@@_ ##1 _options_seq }
953 {
954 \@@_define_option_command:n
955 { ####1 }

```

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

```

956 \@@_with_various_cases:n
957 { ####1 }
958 {
959 \@@_define_option_keyval:nnn
960 { ##1 }
961 { ####1 }
962 { #####1 }
963 }
964 }
965 }
966 }
967 \cs_new:Nn
968 \@@_define_option_command:n
969 {

```

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.

```

970 \str_if_eq:nnTF
971 { #1 }
972 { outputDir }
973 { \@@_define_option_command_output_dir: }
974 {

```

Do not override options defined before loading the package.

```

975 \@@_option_tl_to_csname:nN
976 { #1 }
977 \l_tmpa_tl
978 \cs_if_exist:cF
979 { \l_tmpa_tl }
980 {
981 \@@_get_default_option_value:nN
982 { #1 }
983 \l_tmpa_tl
984 \@@_set_option_value:nV
985 { #1 }

```

```

986 \l_tmpa_tl
987 }
988 }
989 }
990 \ExplSyntaxOff
991 \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.

```

992 \ExplSyntaxOn
993 \cs_new:Nn
994 \@@_define_option_command_output_dir:
995 {
996 \cs_if_free:NT
997 \markdownOptionOutputDir
998 {
999 \bool_if:nTF
1000 {
1001 \cs_if_exist_p:N
1002 \luabridge_tl_set:Nn &&
1003 (
1004 \int_compare_p:nNn
1005 { \g_luabridge_method_int }
1006 =
1007 { \c_luabridge_method_directlua_int } ||
1008 \sys_if_shell_unrestricted_p:
1009)
1010 }
1011 {

```

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.

```

1012 \group_begin:
1013 \cctab_select:N
1014 \c_str_cctab
1015 \luabridge_tl_set:Nn
1016 \l_tmpa_tl
1017 { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
1018 \tl_gset:NV
1019 \markdownOptionOutputDir
1020 \l_tmpa_tl
1021 \group_end:
1022 }
1023 {
1024 \tl_gset:Nn

```

```

1025 \markdownOptionOutputDir
1026 { . }
1027 }
1028 }
1029 }
1030 \cs_new:Nn
1031 \@@_set_option_value:nn
1032 {
1033 \@@_define_option:n
1034 { #1 }
1035 \@@_get_option_type:nN
1036 { #1 }
1037 \l_tmpa_tl
1038 \str_if_eq:NNTF
1039 \c_@@_option_type_counter_tl
1040 \l_tmpa_tl
1041 {
1042 \@@_option_tl_to_csname:nN
1043 { #1 }
1044 \l_tmpa_tl
1045 \int_gset:cn
1046 { \l_tmpa_tl }
1047 { #2 }
1048 }
1049 {
1050 \@@_option_tl_to_csname:nN
1051 { #1 }
1052 \l_tmpa_tl
1053 \cs_set:cpn
1054 { \l_tmpa_tl }
1055 { #2 }
1056 }
1057 }
1058 \cs_generate_variant:Nn
1059 \@@_set_option_value:nn
1060 { nV }
1061 \cs_new:Nn
1062 \@@_define_option:n
1063 {
1064 \@@_option_tl_to_csname:nN
1065 { #1 }
1066 \l_tmpa_tl
1067 \cs_if_free:cT
1068 { \l_tmpa_tl }
1069 {
1070 \@@_get_option_type:nN
1071 { #1 }

```

```

1072 \l_tmpb_tl
1073 \str_if_eq:NNT
1074 \c_@@_option_type_counter_tl
1075 \l_tmpb_tl
1076 {
1077 \@@_option_tl_to_csname:nN
1078 { #1 }
1079 \l_tmpa_tl
1080 \int_new:c
1081 { \l_tmpa_tl }
1082 }
1083 }
1084 }
1085 \cs_new:Nn
1086 \@@_define_option_keyval:nnn
1087 {
1088 \prop_get:cnN
1089 { g_@@_ #1 _option_types_prop }
1090 { #2 }
1091 \l_tmpa_tl
1092 \str_if_eq:VVTF
1093 \l_tmpa_tl
1094 \c_@@_option_type_boolean_tl
1095 {
1096 \keys_define:nn
1097 { markdown/options }
1098 {

```

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

```

1099 #3 .code:n = {
1100 \tl_set:Nx
1101 \l_tmpa_tl
1102 {
1103 \str_case:nnF
1104 { ##1 }
1105 {
1106 { yes } { true }
1107 { no } { false }
1108 }
1109 { ##1 }
1110 }
1111 \@@_set_option_value:nV
1112 { #2 }
1113 \l_tmpa_tl
1114 },
1115 #3 .default:n = { true },

```



```

1116 }
1117 }
1118 {
1119 \keys_define:nn
1120 { markdown/options }
1121 {
1122 #3 .code:n = {
1123 \@@_set_option_value:nn
1124 { #2 }
1125 { ##1 }
1126 },
1127 }
1128 }

```

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.

```

1129 \str_if_eq:VVT
1130 \l_tmpa_tl
1131 \c_@@_option_type_clist_tl
1132 {
1133 \tl_set:Nn
1134 \l_tmpa_tl
1135 { #3 }
1136 \tl_reverse:N
1137 \l_tmpa_tl
1138 \str_if_eq:enF
1139 {
1140 \tl_head:V
1141 \l_tmpa_tl
1142 }
1143 { s }
1144 {
1145 \msg_error:nnn
1146 { markdown }
1147 { malformed-name-for-clist-option }
1148 { #3 }
1149 }
1150 \tl_set:Nx
1151 \l_tmpa_tl
1152 {
1153 \tl_tail:V
1154 \l_tmpa_tl
1155 }
1156 \tl_reverse:N

```

```

1157 \l_tmpa_tl
1158 \tl_put_right:Nn
1159 \l_tmpa_tl
1160 {
1161 .code:n = {
1162 \@@_get_option_value:nN
1163 { #2 }
1164 \l_tmpa_tl
1165 \clist_set:NV
1166 \l_tmpa_clist
1167 { \l_tmpa_tl, { ##1 } }
1168 \@@_set_option_value:nV
1169 { #2 }
1170 \l_tmpa_clist
1171 }
1172 }
1173 \keys_define:nV
1174 { markdown/options }
1175 \l_tmpa_tl
1176 }
1177 }
1178 \cs_generate_variant:Nn
1179 \clist_set:Nn
1180 { NV }
1181 \cs_generate_variant:Nn
1182 \keys_define:nn
1183 { nV }
1184 \cs_generate_variant:Nn
1185 \@@_set_option_value:nn
1186 { nV }
1187 \prg_generate_conditional_variant:Nnn
1188 \str_if_eq:nn
1189 { en }
1190 { F }
1191 \msg_new:nnn
1192 { markdown }
1193 { malformed-name-for-clist-option }
1194 {
1195 Clist~option~name~#1~does~not~end~with~-s.
1196 }

```

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

```

1197 \str_if_eq:VVT
1198 \c_@@_top_layer_tl
1199 \c_@@_option_layer_plain_tex_tl

```

```

1200 {
1201 \@@_define_option_commands_and_keyvals:
1202 }
1203 \ExplSyntaxOff

```

### 2.2.3 Themes

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

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

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

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the T<sub>E</sub>X directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T<sub>E</sub>X document package named `markdownthemewitiko_beamer_MU.tex`.

```

1204 \ExplSyntaxOn
1205 \keys_define:nn
1206 { markdown/options }
1207 {
1208 theme .code:n = {
1209 \@@_set_theme:n
1210 { #1 }
1211 },
1212 import .code:n = {
1213 \tl_set:Nn
1214 \l_tmpa_tl
1215 { #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.

```

1216 \tl_replace_all:NnV
1217 \l_tmpa_tl
1218 { / }
1219 \c_backslash_str
1220 \keys_set:nV
1221 { markdown/options/import }
1222 \l_tmpa_tl
1223 },
1224 }

```

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

```

1225 \seq_new:N
1226 \g_@@_themes_seq
1227 \tl_new:N
1228 \g_@@_current_theme_tl
1229 \tl_gset:Nn
1230 \g_@@_current_theme_tl
1231 { }
1232 \seq_gput_right:NV
1233 \g_@@_themes_seq
1234 \g_@@_current_theme_tl
1235 \cs_new:Nn
1236 \@@_set_theme:n
1237 {

```

First, we validate the theme name.

```

1238 \str_if_in:nnF
1239 { #1 }
1240 { / }
1241 {
1242 \msg_error:nnn
1243 { markdown }
1244 { unqualified-theme-name }
1245 { #1 }
1246 }
1247 \str_if_in:nnT
1248 { #1 }
1249 { _ }
1250 {
1251 \msg_error:nnn
1252 { markdown }
1253 { underscores-in-theme-name }
1254 { #1 }
1255 }

```

Next, we munge the theme name.

```

1256 \str_set:Nn
1257 \l_tmpa_str
1258 { #1 }
1259 \str_replace_all:Nnn
1260 \l_tmpa_str
1261 { / }
1262 { _ }

```

Finally, we load the theme.

```

1263 \tl_gset:Nn
1264 \g_@@_current_theme_tl
1265 { #1 / }
1266 \seq_gput_right:NV
1267 \g_@@_themes_seq
1268 \g_@@_current_theme_tl
1269 \@@_load_theme:nV
1270 { #1 }
1271 \l_tmpa_str
1272 \seq_gpop_right:NN
1273 \g_@@_themes_seq
1274 \l_tmpa_tl
1275 \seq_get_right:NN
1276 \g_@@_themes_seq
1277 \l_tmpa_tl
1278 \tl_gset:NV
1279 \g_@@_current_theme_tl
1280 \l_tmpa_tl
1281 }
1282 \msg_new:nnnn
1283 { markdown }
1284 { unqualified-theme-name }
1285 { Won't~load~theme~with~unqualified~name~#1 }
1286 { Theme~names~must~contain~at~least~one~forward~slash }
1287 \msg_new:nnnn
1288 { markdown }
1289 { underscores-in-theme-name }
1290 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1291 { Theme~names~must~not~contain~underscores~in~their~names }
1292 \cs_generate_variant:Nn
1293 \tl_replace_all:Nnn
1294 { NnV }
1295 \ExplSyntaxOff

```

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

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

```



```

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.

```

1296 \ExplSyntaxOn
1297 \prop_new:N
1298 \g_@@_snippets_prop
1299 \cs_new:Nn
1300 \@@_setup_snippet:nn
1301 {
1302 \tl_if_empty:nT
1303 { #1 }
1304 {
1305 \msg_error:nnn
1306 { markdown }
1307 { empty-snippet-name }
1308 { #1 }
1309 }
1310 \tl_set:NV
1311 \l_tmpa_tl
1312 \g_@@_current_theme_tl
1313 \tl_put_right:Nn
1314 \l_tmpa_tl
1315 { #1 }
1316 \@@_if_snippet_exists:nT
1317 { #1 }

```

```

1318 {
1319 \msg_warning:nnV
1320 { markdown }
1321 { redefined-snippet }
1322 \l_tmpa_tl
1323 }
1324 \keys_precompile:nnN
1325 { markdown/options }
1326 { #2 }
1327 \l_tmpb_tl
1328 \prop_gput:NVV
1329 \g_@@_snippets_prop
1330 \l_tmpa_tl
1331 \l_tmpb_tl
1332 }
1333 \cs_gset_eq:NN
1334 \markdownSetupSnippet
1335 \@@_setup_snippet:nn
1336 \msg_new:nnnn
1337 { markdown }
1338 { empty-snippet-name }
1339 { Empty-snippet-name~#1 }
1340 { Pick-a-non-empty-name-for-your-snippet }
1341 \msg_new:nnn
1342 { markdown }
1343 { redefined-snippet }
1344 { Redefined~snippet~#1 }

```

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

```

1345 \prg_new_conditional:Nnn
1346 \@@_if_snippet_exists:n
1347 { TF, T, F }
1348 {
1349 \tl_set:NV
1350 \l_tmpa_tl
1351 \g_@@_current_theme_tl
1352 \tl_put_right:Nn
1353 \l_tmpa_tl
1354 { #1 }
1355 \prop_get:NVNTF
1356 \g_@@_snippets_prop
1357 \l_tmpa_tl
1358 \l_tmpb_tl
1359 { \prg_return_true: }
1360 { \prg_return_false: }
1361 }
1362 \cs_gset_eq:NN

```

```

1363 \markdownIfSnippetExists
1364 \@@_if_snippet_exists:nTF
The option with key snippet invokes a snippet named $\langle value \rangle$.
1365 \keys_define:nn
1366 { markdown/options }
1367 {
1368 snippet .code:n = {
1369 \tl_set:NV
1370 \l_tmpa_tl
1371 \g_@@_current_theme_tl
1372 \tl_put_right:Nn
1373 \l_tmpa_tl
1374 { #1 }
1375 \@@_if_snippet_exists:nTF
1376 { #1 }
1377 {
1378 \prop_get:NVN
1379 \g_@@_snippets_prop
1380 \l_tmpa_tl
1381 \l_tmpb_tl
1382 \tl_use:N
1383 \l_tmpb_tl
1384 }
1385 {
1386 \msg_error:nnV
1387 { markdown }
1388 { undefined-snippet }
1389 \l_tmpa_tl
1390 }
1391 }
1392 }
1393 \msg_new:nnn
1394 { markdown }
1395 { undefined-snippet }
1396 { Can't~invoke~undefined~snippet~#1 }
1397 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:



1. wahid
2. aithnayn

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

```
\markdownSetup{import=jdooe/lists}
\begin{markdown}[snippet=jdooe/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 = {
 jdooe/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` L<sup>A</sup>T<sub>E</sub>X option:

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

```

1398 \ExplSyntaxOn
1399 \tl_new:N
1400 \l_@@_import_current_theme_tl
1401 \keys_define:nn
1402 { markdown/options/import }
1403 {

```

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

```

1404 unknown .default:n = {},
1405 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.

```

1406 \tl_set_eq:NN
1407 \l_@@_import_current_theme_tl
1408 \l_keys_key_str
1409 \tl_replace_all:NVN
1410 \l_@@_import_current_theme_tl
1411 \c_backslash_str
1412 { / }

```

Here, we import the snippets.

```

1413 \clist_map_inline:nn
1414 { #1 }
1415 {
1416 \regex_extract_once:nnNTF
1417 { ^(.*)\s+as\s+(.*)$ }
1418 { ##1 }
1419 \l_tmpa_seq
1420 {
1421 \seq_pop:NN
1422 \l_tmpa_seq
1423 \l_tmpa_tl
1424 \seq_pop:NN
1425 \l_tmpa_seq
1426 \l_tmpa_tl
1427 \seq_pop:NN
1428 \l_tmpa_seq
1429 \l_tmpb_tl
1430 }
1431 }
1432 \tl_set:Nn
1433 \l_tmpa_tl

```

```

1434 { ##1 }
1435 \tl_set:Nn
1436 \l_tmpb_tl
1437 { ##1 }
1438 }
1439 \tl_put_left:Nn
1440 \l_tmpa_tl
1441 { / }
1442 \tl_put_left:NV
1443 \l_tmpa_tl
1444 \l_@@_import_current_theme_tl
1445 \@@_setup_snippet:Vx
1446 \l_tmpb_tl
1447 { snippet = { \l_tmpa_tl } }
1448 }

```

Here, we load the theme.

```

1449 \@@_set_theme:V
1450 \l_@@_import_current_theme_tl
1451 },
1452 }
1453 \cs_generate_variant:Nn
1454 \tl_replace_all:Nnn
1455 { NVn }
1456 \cs_generate_variant:Nn
1457 \@@_set_theme:n
1458 { V }
1459 \cs_generate_variant:Nn
1460 \@@_setup_snippet:nn
1461 { 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.

```

1462 \ExplSyntaxOn
1463 \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.

```

1464 \prop_new:N \g_@@_renderer_arities_prop
1465 \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.

```
1466 \def\markdownRendererAttributeIdentifier{%
1467 \markdownRendererAttributeIdentifierPrototype}%
1468 \ExplSyntaxOn
1469 \seq_gput_right:Nn
1470 \g_@@_renderers_seq
1471 { attributeIdentifier }
1472 \prop_gput:Nnn
1473 \g_@@_renderer_arities_prop
1474 { attributeIdentifier }
1475 { 1 }
1476 \ExplSyntaxOff
1477 \def\markdownRendererAttributeClassName{%
1478 \markdownRendererAttributeClassNamePrototype}%
1479 \ExplSyntaxOn
1480 \seq_gput_right:Nn
1481 \g_@@_renderers_seq
1482 { attributeClassName }
1483 \prop_gput:Nnn
1484 \g_@@_renderer_arities_prop
1485 { attributeClassName }
1486 { 1 }
1487 \ExplSyntaxOff
1488 \def\markdownRendererAttributeKeyValue{%
1489 \markdownRendererAttributeKeyValuePrototype}%
```

```

1490 \ExplSyntaxOn
1491 \seq_gput_right:Nn
1492 \g_@@_renderers_seq
1493 { attributeKeyValue }
1494 \prop_gput:Nnn
1495 \g_@@_renderer_arities_prop
1496 { attributeKeyValue }
1497 { 2 }
1498 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1499 \def\markdownRendererBlockQuoteBegin{%
1500 \markdownRendererBlockQuoteBeginPrototype}%
1501 \ExplSyntaxOn
1502 \seq_gput_right:Nn
1503 \g_@@_renderers_seq
1504 { blockQuoteBegin }
1505 \prop_gput:Nnn
1506 \g_@@_renderer_arities_prop
1507 { blockQuoteBegin }
1508 { 0 }
1509 \ExplSyntaxOff

```

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

```

1510 \def\markdownRendererBlockQuoteEnd{%
1511 \markdownRendererBlockQuoteEndPrototype}%
1512 \ExplSyntaxOn
1513 \seq_gput_right:Nn
1514 \g_@@_renderers_seq
1515 { blockQuoteEnd }
1516 \prop_gput:Nnn
1517 \g_@@_renderer_arities_prop
1518 { blockQuoteEnd }
1519 { 0 }
1520 \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.

```

1521 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1522 \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1523 \ExplSyntaxOn
1524 \seq_gput_right:Nn
1525 \g_@@_renderers_seq
1526 { bracketedSpanAttributeContextBegin }
1527 \prop_gput:Nnn
1528 \g_@@_renderer_arities_prop
1529 { bracketedSpanAttributeContextBegin }
1530 { 0 }
1531 \ExplSyntaxOff
1532 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1533 \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1534 \ExplSyntaxOn
1535 \seq_gput_right:Nn
1536 \g_@@_renderers_seq
1537 { bracketedSpanAttributeContextEnd }
1538 \prop_gput:Nnn
1539 \g_@@_renderer_arities_prop
1540 { bracketedSpanAttributeContextEnd }
1541 { 0 }
1542 \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.

```

1543 \def\markdownRendererUlBegin{%
1544 \markdownRendererUlBeginPrototype}%
1545 \ExplSyntaxOn
1546 \seq_gput_right:Nn
1547 \g_@@_renderers_seq
1548 { ulBegin }
1549 \prop_gput:Nnn
1550 \g_@@_renderer_arities_prop
1551 { ulBegin }
1552 { 0 }
1553 \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.

```

1554 \def\markdownRendererUlBeginTight{%
1555 \markdownRendererUlBeginTightPrototype}%

```

```

1556 \ExplSyntaxOn
1557 \seq_gput_right:Nn
1558 \g_@@_renderers_seq
1559 { ulBeginTight }
1560 \prop_gput:Nnn
1561 \g_@@_renderer_arities_prop
1562 { ulBeginTight }
1563 { 0 }
1564 \ExplSyntaxOff

```

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

```

1565 \def\markdownRendererUListItem{%
1566 \markdownRendererUListItemPrototype}%
1567 \ExplSyntaxOn
1568 \seq_gput_right:Nn
1569 \g_@@_renderers_seq
1570 { ulItem }
1571 \prop_gput:Nnn
1572 \g_@@_renderer_arities_prop
1573 { ulItem }
1574 { 0 }
1575 \ExplSyntaxOff

```

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

```

1576 \def\markdownRendererUListItemEnd{%
1577 \markdownRendererUListItemEndPrototype}%
1578 \ExplSyntaxOn
1579 \seq_gput_right:Nn
1580 \g_@@_renderers_seq
1581 { ulItemEnd }
1582 \prop_gput:Nnn
1583 \g_@@_renderer_arities_prop
1584 { ulItemEnd }
1585 { 0 }
1586 \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.

```

1587 \def\markdownRendererUListEnd{%
1588 \markdownRendererUListEndPrototype}%
1589 \ExplSyntaxOn
1590 \seq_gput_right:Nn
1591 \g_@@_renderers_seq

```



```

1592 { ulEnd }
1593 \prop_gput:Nnn
1594 \g_@@_renderer_arities_prop
1595 { ulEnd }
1596 { 0 }
1597 \ExplSyntaxOff

```

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

```

1598 \def\markdownRendererUEndTight{%
1599 \markdownRendererUEndTightPrototype}%
1600 \ExplSyntaxOn
1601 \seq_gput_right:Nn
1602 \g_@@_renderers_seq
1603 { ulEndTight }
1604 \prop_gput:Nnn
1605 \g_@@_renderer_arities_prop
1606 { ulEndTight }
1607 { 0 }
1608 \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.

```

1609 \def\markdownRendererCite{%
1610 \markdownRendererCitePrototype}%
1611 \ExplSyntaxOn
1612 \seq_gput_right:Nn
1613 \g_@@_renderers_seq
1614 { cite }
1615 \prop_gput:Nnn
1616 \g_@@_renderer_arities_prop
1617 { cite }
1618 { 1 }
1619 \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.

```
1620 \def\markdownRendererTextCite{%
1621 \markdownRendererTextCitePrototype}%
1622 \ExplSyntaxOn
1623 \seq_gput_right:Nn
1624 \g_@@_renderers_seq
1625 { textCite }
1626 \prop_gput:Nnn
1627 \g_@@_renderer_arities_prop
1628 { textCite }
1629 { 1 }
1630 \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.

```
1631 \def\markdownRendererInputVerbatim{%
1632 \markdownRendererInputVerbatimPrototype}%
1633 \ExplSyntaxOn
1634 \seq_gput_right:Nn
1635 \g_@@_renderers_seq
1636 { inputVerbatim }
1637 \prop_gput:Nnn
1638 \g_@@_renderer_arities_prop
1639 { inputVerbatim }
1640 { 1 }
1641 \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.

```
1642 \def\markdownRendererInputFencedCode{%
1643 \markdownRendererInputFencedCodePrototype}%
1644 \ExplSyntaxOn
1645 \seq_gput_right:Nn
1646 \g_@@_renderers_seq
1647 { inputFencedCode }
1648 \prop_gput:Nnn
1649 \g_@@_renderer_arities_prop
1650 { inputFencedCode }
1651 { 3 }
```

```
1652 \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.

```
1653 \def\markdownRendererCodeSpan{%
1654 \markdownRendererCodeSpanPrototype}%
1655 \ExplSyntaxOn
1656 \seq_gput_right:Nn
1657 \g_@@_renderers_seq
1658 { codeSpan }
1659 \prop_gput:Nnn
1660 \g_@@_renderer_arities_prop
1661 { codeSpan }
1662 { 1 }
1663 \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.

```
1664 \def\markdownRendererCodeSpanAttributeContextBegin{%
1665 \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1666 \ExplSyntaxOn
1667 \seq_gput_right:Nn
1668 \g_@@_renderers_seq
1669 { codeSpanAttributeContextBegin }
1670 \prop_gput:Nnn
1671 \g_@@_renderer_arities_prop
1672 { codeSpanAttributeContextBegin }
1673 { 0 }
1674 \ExplSyntaxOff
1675 \def\markdownRendererCodeSpanAttributeContextEnd{%
1676 \markdownRendererCodeSpanAttributeContextEndPrototype}%
1677 \ExplSyntaxOn
1678 \seq_gput_right:Nn
1679 \g_@@_renderers_seq
1680 { codeSpanAttributeContextEnd }
1681 \prop_gput:Nnn
1682 \g_@@_renderer_arities_prop
1683 { codeSpanAttributeContextEnd }
1684 { 0 }
1685 \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.

```
1686 \def\markdownRendererContentBlock{%
1687 \markdownRendererContentBlockPrototype}%
1688 \ExplSyntaxOn
1689 \seq_gput_right:Nn
1690 \g_@@_renderers_seq
1691 { contentBlock }
1692 \prop_gput:Nnn
1693 \g_@@_renderer_arities_prop
1694 { contentBlock }
1695 { 4 }
1696 \ExplSyntaxOff
```

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

```
1697 \def\markdownRendererContentBlockOnlineImage{%
1698 \markdownRendererContentBlockOnlineImagePrototype}%
1699 \ExplSyntaxOn
1700 \seq_gput_right:Nn
1701 \g_@@_renderers_seq
1702 { contentBlockOnlineImage }
1703 \prop_gput:Nnn
1704 \g_@@_renderer_arities_prop
1705 { contentBlockOnlineImage }
1706 { 4 }
1707 \ExplSyntaxOff
```

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

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local T<sub>E</sub>X directory structure. In this file,

---

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

you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```

1708 \def\markdownRendererContentBlockCode{%
1709 \markdownRendererContentBlockCodePrototype}%
1710 \ExplSyntaxOn
1711 \seq_gput_right:Nn
1712 \g_@@_renderers_seq
1713 { contentBlockCode }
1714 \prop_gput:Nnn
1715 \g_@@_renderer_arities_prop
1716 { contentBlockCode }
1717 { 5 }
1718 \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.

```

1719 \def\markdownRendererDlBegin{%
1720 \markdownRendererDlBeginPrototype}%
1721 \ExplSyntaxOn
1722 \seq_gput_right:Nn
1723 \g_@@_renderers_seq
1724 { dlBegin }
1725 \prop_gput:Nnn
1726 \g_@@_renderer_arities_prop
1727 { dlBegin }
1728 { 0 }
1729 \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.

```

1730 \def\markdownRendererDlBeginTight{%
1731 \markdownRendererDlBeginTightPrototype}%
1732 \ExplSyntaxOn
1733 \seq_gput_right:Nn
1734 \g_@@_renderers_seq
1735 { dlBeginTight }

```

```

1736 \prop_gput:Nnn
1737 \g_@@_renderer_arities_prop
1738 { dlBeginTight }
1739 { 0 }
1740 \ExplSyntaxOff

```

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

```

1741 \def\markdownRendererDlItem{%
1742 \markdownRendererDlItemPrototype}%
1743 \ExplSyntaxOn
1744 \seq_gput_right:Nn
1745 \g_@@_renderers_seq
1746 { dlItem }
1747 \prop_gput:Nnn
1748 \g_@@_renderer_arities_prop
1749 { dlItem }
1750 { 1 }
1751 \ExplSyntaxOff

```

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

```

1752 \def\markdownRendererDlItemEnd{%
1753 \markdownRendererDlItemEndPrototype}%
1754 \ExplSyntaxOn
1755 \seq_gput_right:Nn
1756 \g_@@_renderers_seq
1757 { dlItemEnd }
1758 \prop_gput:Nnn
1759 \g_@@_renderer_arities_prop
1760 { dlItemEnd }
1761 { 0 }
1762 \ExplSyntaxOff

```

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

```

1763 \def\markdownRendererDlDefinitionBegin{%
1764 \markdownRendererDlDefinitionBeginPrototype}%
1765 \ExplSyntaxOn
1766 \seq_gput_right:Nn
1767 \g_@@_renderers_seq
1768 { dlDefinitionBegin }
1769 \prop_gput:Nnn
1770 \g_@@_renderer_arities_prop
1771 { dlDefinitionBegin }
1772 { 0 }
1773 \ExplSyntaxOff

```

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

```
1774 \def\markdownRendererDlDefinitionEnd{%
1775 \markdownRendererDlDefinitionEndPrototype}%
1776 \ExplSyntaxOn
1777 \seq_gput_right:Nn
1778 \g_@@_renderers_seq
1779 { dlDefinitionEnd }
1780 \prop_gput:Nnn
1781 \g_@@_renderer_arities_prop
1782 { dlDefinitionEnd }
1783 { 0 }
1784 \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.

```
1785 \def\markdownRendererDlEnd{%
1786 \markdownRendererDlEndPrototype}%
1787 \ExplSyntaxOn
1788 \seq_gput_right:Nn
1789 \g_@@_renderers_seq
1790 { dlEnd }
1791 \prop_gput:Nnn
1792 \g_@@_renderer_arities_prop
1793 { dlEnd }
1794 { 0 }
1795 \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.

```
1796 \def\markdownRendererDlEndTight{%
1797 \markdownRendererDlEndTightPrototype}%
1798 \ExplSyntaxOn
1799 \seq_gput_right:Nn
1800 \g_@@_renderers_seq
1801 { dlEndTight }
1802 \prop_gput:Nnn
1803 \g_@@_renderer_arities_prop
1804 { dlEndTight }
1805 { 0 }
1806 \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.

```
1807 \def\markdownRendererEllipsis{%
1808 \markdownRendererEllipsisPrototype}%
1809 \ExplSyntaxOn
1810 \seq_gput_right:Nn
1811 \g_@@_renderers_seq
1812 { ellipsis }
1813 \prop_gput:Nnn
1814 \g_@@_renderer_arities_prop
1815 { ellipsis }
1816 { 0 }
1817 \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.

```
1818 \def\markdownRendererEmphasis{%
1819 \markdownRendererEmphasisPrototype}%
1820 \ExplSyntaxOn
1821 \seq_gput_right:Nn
1822 \g_@@_renderers_seq
1823 { emphasis }
1824 \prop_gput:Nnn
1825 \g_@@_renderer_arities_prop
1826 { emphasis }
1827 { 1 }
1828 \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.

```
1829 \def\markdownRendererStrongEmphasis{%
1830 \markdownRendererStrongEmphasisPrototype}%
1831 \ExplSyntaxOn
1832 \seq_gput_right:Nn
1833 \g_@@_renderers_seq
1834 { strongEmphasis }
1835 \prop_gput:Nnn
1836 \g_@@_renderer_arities_prop
1837 { strongEmphasis }
1838 { 1 }
```



1839 \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.

```
1840 \def\markdownRendererFencedCodeAttributeContextBegin{%
1841 \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1842 \ExplSyntaxOn
1843 \seq_gput_right:Nn
1844 \g_@@_renderers_seq
1845 { fencedCodeAttributeContextBegin }
1846 \prop_gput:Nnn
1847 \g_@@_renderer_arities_prop
1848 { fencedCodeAttributeContextBegin }
1849 { 0 }
1850 \ExplSyntaxOff
1851 \def\markdownRendererFencedCodeAttributeContextEnd{%
1852 \markdownRendererFencedCodeAttributeContextEndPrototype}%
1853 \ExplSyntaxOn
1854 \seq_gput_right:Nn
1855 \g_@@_renderers_seq
1856 { fencedCodeAttributeContextEnd }
1857 \prop_gput:Nnn
1858 \g_@@_renderer_arities_prop
1859 { fencedCodeAttributeContextEnd }
1860 { 0 }
1861 \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.

```
1862 \def\markdownRendererFencedDivAttributeContextBegin{%
1863 \markdownRendererFencedDivAttributeContextBeginPrototype}%
1864 \ExplSyntaxOn
1865 \seq_gput_right:Nn
1866 \g_@@_renderers_seq
1867 { fencedDivAttributeContextBegin }
1868 \prop_gput:Nnn
1869 \g_@@_renderer_arities_prop
```

```

1870 { fencedDivAttributeContextBegin }
1871 { 0 }
1872 \ExplSyntaxOff
1873 \def\markdownRendererFencedDivAttributeContextEnd{%
1874 \markdownRendererFencedDivAttributeContextEndPrototype}%
1875 \ExplSyntaxOn
1876 \seq_gput_right:Nn
1877 \g_@@_renderers_seq
1878 { fencedDivAttributeContextEnd }
1879 \prop_gput:Nnn
1880 \g_@@_renderer_arities_prop
1881 { fencedDivAttributeContextEnd }
1882 { 0 }
1883 \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.

```

1884 \def\markdownRendererHeaderAttributeContextBegin{%
1885 \markdownRendererHeaderAttributeContextBeginPrototype}%
1886 \ExplSyntaxOn
1887 \seq_gput_right:Nn
1888 \g_@@_renderers_seq
1889 { headerAttributeContextBegin }
1890 \prop_gput:Nnn
1891 \g_@@_renderer_arities_prop
1892 { headerAttributeContextBegin }
1893 { 0 }
1894 \ExplSyntaxOff
1895 \def\markdownRendererHeaderAttributeContextEnd{%
1896 \markdownRendererHeaderAttributeContextEndPrototype}%
1897 \ExplSyntaxOn
1898 \seq_gput_right:Nn
1899 \g_@@_renderers_seq
1900 { headerAttributeContextEnd }
1901 \prop_gput:Nnn
1902 \g_@@_renderer_arities_prop
1903 { headerAttributeContextEnd }
1904 { 0 }
1905 \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.

```
1906 \def\markdownRendererHeadingOne{%
1907 \markdownRendererHeadingOnePrototype}%
1908 \ExplSyntaxOn
1909 \seq_gput_right:Nn
1910 \g_@@_renderers_seq
1911 { headingOne }
1912 \prop_gput:Nnn
1913 \g_@@_renderer_arities_prop
1914 { headingOne }
1915 { 1 }
1916 \ExplSyntaxOff
```

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

```
1917 \def\markdownRendererHeadingTwo{%
1918 \markdownRendererHeadingTwoPrototype}%
1919 \ExplSyntaxOn
1920 \seq_gput_right:Nn
1921 \g_@@_renderers_seq
1922 { headingTwo }
1923 \prop_gput:Nnn
1924 \g_@@_renderer_arities_prop
1925 { headingTwo }
1926 { 1 }
1927 \ExplSyntaxOff
```

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

```
1928 \def\markdownRendererHeadingThree{%
1929 \markdownRendererHeadingThreePrototype}%
1930 \ExplSyntaxOn
1931 \seq_gput_right:Nn
1932 \g_@@_renderers_seq
1933 { headingThree }
1934 \prop_gput:Nnn
1935 \g_@@_renderer_arities_prop
1936 { headingThree }
1937 { 1 }
1938 \ExplSyntaxOff
```

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

```
1939 \def\markdownRendererHeadingFour{%
1940 \markdownRendererHeadingFourPrototype}%
```

```

1941 \ExplSyntaxOn
1942 \seq_gput_right:Nn
1943 \g_@@_renderers_seq
1944 { headingFour }
1945 \prop_gput:Nnn
1946 \g_@@_renderer_arities_prop
1947 { headingFour }
1948 { 1 }
1949 \ExplSyntaxOff

```

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

```

1950 \def\markdownRendererHeadingFive{%
1951 \markdownRendererHeadingFivePrototype}%
1952 \ExplSyntaxOn
1953 \seq_gput_right:Nn
1954 \g_@@_renderers_seq
1955 { headingFive }
1956 \prop_gput:Nnn
1957 \g_@@_renderer_arities_prop
1958 { headingFive }
1959 { 1 }
1960 \ExplSyntaxOff

```

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

```

1961 \def\markdownRendererHeadingSix{%
1962 \markdownRendererHeadingSixPrototype}%
1963 \ExplSyntaxOn
1964 \seq_gput_right:Nn
1965 \g_@@_renderers_seq
1966 { headingSix }
1967 \prop_gput:Nnn
1968 \g_@@_renderer_arities_prop
1969 { headingSix }
1970 { 1 }
1971 \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.

```

1972 \def\markdownRendererInlineHtmlComment{%
1973 \markdownRendererInlineHtmlCommentPrototype}%

```

```

1974 \ExplSyntaxOn
1975 \seq_gput_right:Nn
1976 \g_@@_renderers_seq
1977 { inlineHtmlComment }
1978 \prop_gput:Nnn
1979 \g_@@_renderer_arities_prop
1980 { inlineHtmlComment }
1981 { 1 }
1982 \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.

```

1983 \def\markdownRendererInlineHtmlTag{%
1984 \markdownRendererInlineHtmlTagPrototype}%
1985 \ExplSyntaxOn
1986 \seq_gput_right:Nn
1987 \g_@@_renderers_seq
1988 { inlineHtmlTag }
1989 \prop_gput:Nnn
1990 \g_@@_renderer_arities_prop
1991 { inlineHtmlTag }
1992 { 1 }
1993 \ExplSyntaxOff
1994 \def\markdownRendererInputBlockHtmlElement{%
1995 \markdownRendererInputBlockHtmlElementPrototype}%
1996 \ExplSyntaxOn
1997 \seq_gput_right:Nn
1998 \g_@@_renderers_seq
1999 { inputBlockHtmlElement }
2000 \prop_gput:Nnn
2001 \g_@@_renderer_arities_prop
2002 { inputBlockHtmlElement }
2003 { 1 }
2004 \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.

```

2005 \def\markdownRendererImage{%
2006 \markdownRendererImagePrototype}%
2007 \ExplSyntaxOn
2008 \seq_gput_right:Nn
2009 \g_@@_renderers_seq
2010 { image }
2011 \prop_gput:Nnn
2012 \g_@@_renderer_arities_prop
2013 { image }
2014 { 4 }
2015 \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.

```

2016 \def\markdownRendererImageAttributeContextBegin{%
2017 \markdownRendererImageAttributeContextBeginPrototype}%
2018 \ExplSyntaxOn
2019 \seq_gput_right:Nn
2020 \g_@@_renderers_seq
2021 { imageAttributeContextBegin }
2022 \prop_gput:Nnn
2023 \g_@@_renderer_arities_prop
2024 { imageAttributeContextBegin }
2025 { 0 }
2026 \ExplSyntaxOff
2027 \def\markdownRendererImageAttributeContextEnd{%
2028 \markdownRendererImageAttributeContextEndPrototype}%
2029 \ExplSyntaxOn
2030 \seq_gput_right:Nn
2031 \g_@@_renderers_seq
2032 { imageAttributeContextEnd }
2033 \prop_gput:Nnn
2034 \g_@@_renderer_arities_prop
2035 { imageAttributeContextEnd }
2036 { 0 }
2037 \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.

```
2038 \def\markdownRendererInterblockSeparator{%
2039 \markdownRendererInterblockSeparatorPrototype}%
2040 \ExplSyntaxOn
2041 \seq_gput_right:Nn
2042 \g_@@_renderers_seq
2043 { interblockSeparator }
2044 \prop_gput:Nnn
2045 \g_@@_renderer_arities_prop
2046 { interblockSeparator }
2047 { 0 }
2048 \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.

```
2049 \def\markdownRendererParagraphSeparator{%
2050 \markdownRendererParagraphSeparatorPrototype}%
2051 \ExplSyntaxOn
2052 \seq_gput_right:Nn
2053 \g_@@_renderers_seq
2054 { paragraphSeparator }
2055 \prop_gput:Nnn
2056 \g_@@_renderer_arities_prop
2057 { paragraphSeparator }
2058 { 0 }
2059 \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.

```
2060 \def\markdownRendererLineBlockBegin{%
2061 \markdownRendererLineBlockBeginPrototype}%
2062 \ExplSyntaxOn
2063 \seq_gput_right:Nn
2064 \g_@@_renderers_seq
2065 { lineBlockBegin }
2066 \prop_gput:Nnn
```

```

2067 \g_@@_renderer_arities_prop
2068 { lineBlockBegin }
2069 { 0 }
2070 \ExplSyntaxOff
2071 \def\markdownRendererLineBlockEnd{%
2072 \markdownRendererLineBlockEndPrototype}%
2073 \ExplSyntaxOn
2074 \seq_gput_right:Nn
2075 \g_@@_renderers_seq
2076 { lineBlockEnd }
2077 \prop_gput:Nnn
2078 \g_@@_renderer_arities_prop
2079 { lineBlockEnd }
2080 { 0 }
2081 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2082 \def\markdownRendererSoftLineBreak{%
2083 \markdownRendererSoftLineBreakPrototype}%
2084 \ExplSyntaxOn
2085 \seq_gput_right:Nn
2086 \g_@@_renderers_seq
2087 { softLineBreak }
2088 \prop_gput:Nnn
2089 \g_@@_renderer_arities_prop
2090 { softLineBreak }
2091 { 0 }
2092 \ExplSyntaxOff

```

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

```

2093 \def\markdownRendererHardLineBreak{%
2094 \markdownRendererHardLineBreakPrototype}%
2095 \ExplSyntaxOn
2096 \seq_gput_right:Nn
2097 \g_@@_renderers_seq
2098 { hardLineBreak }
2099 \prop_gput:Nnn
2100 \g_@@_renderer_arities_prop
2101 { hardLineBreak }
2102 { 0 }
2103 \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.

```

2104 \def\markdownRendererLink{%
2105 \markdownRendererLinkPrototype}%
2106 \ExplSyntaxOn
2107 \seq_gput_right:Nn
2108 \g_@@_renderers_seq
2109 { link }
2110 \prop_gput:Nnn
2111 \g_@@_renderer_arities_prop
2112 { link }
2113 { 4 }
2114 \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.

```

2115 \def\markdownRendererLinkAttributeContextBegin{%
2116 \markdownRendererLinkAttributeContextBeginPrototype}%
2117 \ExplSyntaxOn
2118 \seq_gput_right:Nn
2119 \g_@@_renderers_seq
2120 { linkAttributeContextBegin }
2121 \prop_gput:Nnn
2122 \g_@@_renderer_arities_prop
2123 { linkAttributeContextBegin }
2124 { 0 }
2125 \ExplSyntaxOff
2126 \def\markdownRendererLinkAttributeContextEnd{%
2127 \markdownRendererLinkAttributeContextEndPrototype}%
2128 \ExplSyntaxOn
2129 \seq_gput_right:Nn
2130 \g_@@_renderers_seq
2131 { linkAttributeContextEnd }
2132 \prop_gput:Nnn
2133 \g_@@_renderer_arities_prop
2134 { linkAttributeContextEnd }
2135 { 0 }
2136 \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.

```
2137 \def\markdownRendererMark{%
2138 \markdownRendererMarkPrototype}%
2139 \ExplSyntaxOn
2140 \seq_gput_right:Nn
2141 \g_@@_renderers_seq
2142 { mark }
2143 \prop_gput:Nnn
2144 \g_@@_renderer_arities_prop
2145 { mark }
2146 { 1 }
2147 \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  $\TeX$  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.

```
2148 \def\markdownRendererDocumentBegin{%
2149 \markdownRendererDocumentBeginPrototype}%
2150 \ExplSyntaxOn
2151 \seq_gput_right:Nn
2152 \g_@@_renderers_seq
2153 { documentBegin }
2154 \prop_gput:Nnn
2155 \g_@@_renderer_arities_prop
2156 { documentBegin }
2157 { 0 }
2158 \ExplSyntaxOff
2159 \def\markdownRendererDocumentEnd{%
2160 \markdownRendererDocumentEndPrototype}%
2161 \ExplSyntaxOn
2162 \seq_gput_right:Nn
2163 \g_@@_renderers_seq
2164 { documentEnd }
2165 \prop_gput:Nnn
2166 \g_@@_renderer_arities_prop
2167 { documentEnd }
2168 { 0 }
2169 \ExplSyntaxOff
```

### 2.2.5.28 Non-Breaking Space Renderer

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

```
2170 \def\markdownRendererNbsp{%
2171 \markdownRendererNbspPrototype}%
2172 \ExplSyntaxOn
2173 \seq_gput_right:Nn
2174 \g_@@_renderers_seq
2175 { nbsp }
2176 \prop_gput:Nnn
2177 \g_@@_renderer_arities_prop
2178 { nbsp }
2179 { 0 }
2180 \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.

```
2181 \def\markdownRendererNote{%
2182 \markdownRendererNotePrototype}%
2183 \ExplSyntaxOn
2184 \seq_gput_right:Nn
2185 \g_@@_renderers_seq
2186 { note }
2187 \prop_gput:Nnn
2188 \g_@@_renderer_arities_prop
2189 { note }
2190 { 1 }
2191 \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.

```
2192 \def\markdownRendererOlBegin{%
2193 \markdownRendererOlBeginPrototype}%
2194 \ExplSyntaxOn
2195 \seq_gput_right:Nn
2196 \g_@@_renderers_seq
2197 { olBegin }
2198 \prop_gput:Nnn
2199 \g_@@_renderer_arities_prop
2200 { olBegin }
```

```

2201 { 0 }
2202 \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.

```

2203 \def\markdownRendererOlBeginTight{%
2204 \markdownRendererOlBeginTightPrototype}%
2205 \ExplSyntaxOn
2206 \seq_gput_right:Nn
2207 \g_@@_renderers_seq
2208 { olBeginTight }
2209 \prop_gput:Nnn
2210 \g_@@_renderer_arities_prop
2211 { olBeginTight }
2212 { 0 }
2213 \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`).

```

2214 \def\markdownRendererFancyOlBegin{%
2215 \markdownRendererFancyOlBeginPrototype}%
2216 \ExplSyntaxOn
2217 \seq_gput_right:Nn
2218 \g_@@_renderers_seq
2219 { fancyOlBegin }
2220 \prop_gput:Nnn
2221 \g_@@_renderer_arities_prop
2222 { fancyOlBegin }
2223 { 2 }
2224 \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.

```

2225 \def\markdownRendererFancyOlBeginTight{%
2226 \markdownRendererFancyOlBeginTightPrototype}%

```

```

2227 \ExplSyntaxOn
2228 \seq_gput_right:Nn
2229 \g_@@_renderers_seq
2230 { fancyOlBeginTight }
2231 \prop_gput:Nnn
2232 \g_@@_renderer_arities_prop
2233 { fancyOlBeginTight }
2234 { 2 }
2235 \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.

```

2236 \def\markdownRendererOlItem{%
2237 \markdownRendererOlItemPrototype}%
2238 \ExplSyntaxOn
2239 \seq_gput_right:Nn
2240 \g_@@_renderers_seq
2241 { olItem }
2242 \prop_gput:Nnn
2243 \g_@@_renderer_arities_prop
2244 { olItem }
2245 { 0 }
2246 \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.

```

2247 \def\markdownRendererOlItemEnd{%
2248 \markdownRendererOlItemEndPrototype}%
2249 \ExplSyntaxOn
2250 \seq_gput_right:Nn
2251 \g_@@_renderers_seq
2252 { olItemEnd }
2253 \prop_gput:Nnn
2254 \g_@@_renderer_arities_prop
2255 { olItemEnd }
2256 { 0 }
2257 \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.

```

2258 \def\markdownRendererOlItemWithNumber{%
2259 \markdownRendererOlItemWithNumberPrototype}%

```

```

2260 \ExplSyntaxOn
2261 \seq_gput_right:Nn
2262 \g_@@_renderers_seq
2263 { olItemWithNumber }
2264 \prop_gput:Nnn
2265 \g_@@_renderer_arities_prop
2266 { olItemWithNumber }
2267 { 1 }
2268 \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.

```

2269 \def\markdownRendererFancyOlItem{%
2270 \markdownRendererFancyOlItemPrototype}%
2271 \ExplSyntaxOn
2272 \seq_gput_right:Nn
2273 \g_@@_renderers_seq
2274 { fancyOlItem }
2275 \prop_gput:Nnn
2276 \g_@@_renderer_arities_prop
2277 { fancyOlItem }
2278 { 0 }
2279 \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.

```

2280 \def\markdownRendererFancyOlItemEnd{%
2281 \markdownRendererFancyOlItemEndPrototype}%
2282 \ExplSyntaxOn
2283 \seq_gput_right:Nn
2284 \g_@@_renderers_seq
2285 { fancyOlItemEnd }
2286 \prop_gput:Nnn
2287 \g_@@_renderer_arities_prop
2288 { fancyOlItemEnd }
2289 { 0 }
2290 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItemWithNumber` 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.

```

2291 \def\markdownRendererFancyOlItemWithNumber{%
2292 \markdownRendererFancyOlItemWithNumberPrototype}%

```

```

2293 \ExplSyntaxOn
2294 \seq_gput_right:Nn
2295 \g_@@_renderers_seq
2296 { fancyOListItemWithNumber }
2297 \prop_gput:Nnn
2298 \g_@@_renderer_arities_prop
2299 { fancyOListItemWithNumber }
2300 { 1 }
2301 \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.

```

2302 \def\markdownRendererO1End{%
2303 \markdownRendererO1EndPrototype}%
2304 \ExplSyntaxOn
2305 \seq_gput_right:Nn
2306 \g_@@_renderers_seq
2307 { olEnd }
2308 \prop_gput:Nnn
2309 \g_@@_renderer_arities_prop
2310 { olEnd }
2311 { 0 }
2312 \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.

```

2313 \def\markdownRendererO1EndTight{%
2314 \markdownRendererO1EndTightPrototype}%
2315 \ExplSyntaxOn
2316 \seq_gput_right:Nn
2317 \g_@@_renderers_seq
2318 { olEndTight }
2319 \prop_gput:Nnn
2320 \g_@@_renderer_arities_prop
2321 { olEndTight }
2322 { 0 }
2323 \ExplSyntaxOff

```

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

```

2324 \def\markdownRendererFancyO1End{%
2325 \markdownRendererFancyO1EndPrototype}%
2326 \ExplSyntaxOn
2327 \seq_gput_right:Nn
2328 \g_@@_renderers_seq
2329 { fancyO1End }
2330 \prop_gput:Nnn
2331 \g_@@_renderer_arities_prop
2332 { fancyO1End }
2333 { 0 }
2334 \ExplSyntaxOff

```

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

```

2335 \def\markdownRendererFancyO1EndTight{%
2336 \markdownRendererFancyO1EndTightPrototype}%
2337 \ExplSyntaxOn
2338 \seq_gput_right:Nn
2339 \g_@@_renderers_seq
2340 { fancyO1EndTight }
2341 \prop_gput:Nnn
2342 \g_@@_renderer_arities_prop
2343 { fancyO1EndTight }
2344 { 0 }
2345 \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.

```

2346 \def\markdownRendererInputRawInline{%
2347 \markdownRendererInputRawInlinePrototype}%
2348 \ExplSyntaxOn
2349 \seq_gput_right:Nn
2350 \g_@@_renderers_seq
2351 { inputRawInline }
2352 \prop_gput:Nnn
2353 \g_@@_renderer_arities_prop
2354 { inputRawInline }
2355 { 2 }
2356 \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.

```

2357 \def\markdownRendererInputRawBlock{%
2358 \markdownRendererInputRawBlockPrototype}%
2359 \ExplSyntaxOn
2360 \seq_gput_right:Nn
2361 \g_@@_renderers_seq
2362 { inputRawBlock }
2363 \prop_gput:Nnn
2364 \g_@@_renderer_arities_prop
2365 { inputRawBlock }
2366 { 2 }
2367 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2368 \def\markdownRendererSectionBegin{%
2369 \markdownRendererSectionBeginPrototype}%
2370 \ExplSyntaxOn
2371 \seq_gput_right:Nn
2372 \g_@@_renderers_seq
2373 { sectionBegin }
2374 \prop_gput:Nnn
2375 \g_@@_renderer_arities_prop
2376 { sectionBegin }
2377 { 0 }
2378 \ExplSyntaxOff
2379 \def\markdownRendererSectionEnd{%
2380 \markdownRendererSectionEndPrototype}%
2381 \ExplSyntaxOn
2382 \seq_gput_right:Nn
2383 \g_@@_renderers_seq
2384 { sectionEnd }
2385 \prop_gput:Nnn
2386 \g_@@_renderer_arities_prop
2387 { sectionEnd }
2388 { 0 }
2389 \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.

```

2390 \def\markdownRendererReplacementCharacter{%
2391 \markdownRendererReplacementCharacterPrototype}%
2392 \ExplSyntaxOn
2393 \seq_gput_right:Nn
2394 \g_@@_renderers_seq
2395 { replacementCharacter }
2396 \prop_gput:Nnn
2397 \g_@@_renderer_arities_prop
2398 { replacementCharacter }
2399 { 0 }
2400 \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`.

```

2401 \def\markdownRendererLeftBrace{%
2402 \markdownRendererLeftBracePrototype}%
2403 \ExplSyntaxOn
2404 \seq_gput_right:Nn
2405 \g_@@_renderers_seq
2406 { leftBrace }
2407 \prop_gput:Nnn
2408 \g_@@_renderer_arities_prop
2409 { leftBrace }
2410 { 0 }
2411 \ExplSyntaxOff
2412 \def\markdownRendererRightBrace{%
2413 \markdownRendererRightBracePrototype}%
2414 \ExplSyntaxOn
2415 \seq_gput_right:Nn
2416 \g_@@_renderers_seq
2417 { rightBrace }
2418 \prop_gput:Nnn
2419 \g_@@_renderer_arities_prop
2420 { rightBrace }
2421 { 0 }
2422 \ExplSyntaxOff
2423 \def\markdownRendererDollarSign{%
2424 \markdownRendererDollarSignPrototype}%
2425 \ExplSyntaxOn
2426 \seq_gput_right:Nn
2427 \g_@@_renderers_seq

```

```

2428 { dollarSign }
2429 \prop_gput:Nnn
2430 \g_@@_renderer_arities_prop
2431 { dollarSign }
2432 { 0 }
2433 \ExplSyntaxOff
2434 \def\markdownRendererPercentSign{%
2435 \markdownRendererPercentSignPrototype}%
2436 \ExplSyntaxOn
2437 \seq_gput_right:Nn
2438 \g_@@_renderers_seq
2439 { percentSign }
2440 \prop_gput:Nnn
2441 \g_@@_renderer_arities_prop
2442 { percentSign }
2443 { 0 }
2444 \ExplSyntaxOff
2445 \def\markdownRendererAmpersand{%
2446 \markdownRendererAmpersandPrototype}%
2447 \ExplSyntaxOn
2448 \seq_gput_right:Nn
2449 \g_@@_renderers_seq
2450 { ampersand }
2451 \prop_gput:Nnn
2452 \g_@@_renderer_arities_prop
2453 { ampersand }
2454 { 0 }
2455 \ExplSyntaxOff
2456 \def\markdownRendererUnderscore{%
2457 \markdownRendererUnderscorePrototype}%
2458 \ExplSyntaxOn
2459 \seq_gput_right:Nn
2460 \g_@@_renderers_seq
2461 { underscore }
2462 \prop_gput:Nnn
2463 \g_@@_renderer_arities_prop
2464 { underscore }
2465 { 0 }
2466 \ExplSyntaxOff
2467 \def\markdownRendererHash{%
2468 \markdownRendererHashPrototype}%
2469 \ExplSyntaxOn
2470 \seq_gput_right:Nn
2471 \g_@@_renderers_seq
2472 { hash }
2473 \prop_gput:Nnn
2474 \g_@@_renderer_arities_prop

```

```

2475 { hash }
2476 { 0 }
2477 \ExplSyntaxOff
2478 \def\markdownRendererCircumflex{%
2479 \markdownRendererCircumflexPrototype}%
2480 \ExplSyntaxOn
2481 \seq_gput_right:Nn
2482 \g_@@_renderers_seq
2483 { circumflex }
2484 \prop_gput:Nnn
2485 \g_@@_renderer_arities_prop
2486 { circumflex }
2487 { 0 }
2488 \ExplSyntaxOff
2489 \def\markdownRendererBackslash{%
2490 \markdownRendererBackslashPrototype}%
2491 \ExplSyntaxOn
2492 \seq_gput_right:Nn
2493 \g_@@_renderers_seq
2494 { backslash }
2495 \prop_gput:Nnn
2496 \g_@@_renderer_arities_prop
2497 { backslash }
2498 { 0 }
2499 \ExplSyntaxOff
2500 \def\markdownRendererTilde{%
2501 \markdownRendererTildePrototype}%
2502 \ExplSyntaxOn
2503 \seq_gput_right:Nn
2504 \g_@@_renderers_seq
2505 { tilde }
2506 \prop_gput:Nnn
2507 \g_@@_renderer_arities_prop
2508 { tilde }
2509 { 0 }
2510 \ExplSyntaxOff
2511 \def\markdownRendererPipe{%
2512 \markdownRendererPipePrototype}%
2513 \ExplSyntaxOn
2514 \seq_gput_right:Nn
2515 \g_@@_renderers_seq
2516 { pipe }
2517 \prop_gput:Nnn
2518 \g_@@_renderer_arities_prop
2519 { pipe }
2520 { 0 }
2521 \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.

```
2522 \def\markdownRendererStrikeThrough{%
2523 \markdownRendererStrikeThroughPrototype}%
2524 \ExplSyntaxOn
2525 \seq_gput_right:Nn
2526 \g_@@_renderers_seq
2527 { strikeThrough }
2528 \prop_gput:Nnn
2529 \g_@@_renderer_arities_prop
2530 { strikeThrough }
2531 { 1 }
2532 \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.

```
2533 \def\markdownRendererSubscript{%
2534 \markdownRendererSubscriptPrototype}%
2535 \ExplSyntaxOn
2536 \seq_gput_right:Nn
2537 \g_@@_renderers_seq
2538 { subscript }
2539 \prop_gput:Nnn
2540 \g_@@_renderer_arities_prop
2541 { subscript }
2542 { 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.

```
2543 \def\markdownRendererSuperscript{%
2544 \markdownRendererSuperscriptPrototype}%
2545 \ExplSyntaxOn
2546 \seq_gput_right:Nn
2547 \g_@@_renderers_seq
2548 { superscript }
2549 \prop_gput:Nnn
2550 \g_@@_renderer_arities_prop
```

```

2551 { superscript }
2552 { 1 }
2553 \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.

```

2554 \def\markdownRendererTableAttributeContextBegin{%
2555 \markdownRendererTableAttributeContextBeginPrototype}%
2556 \ExplSyntaxOn
2557 \seq_gput_right:Nn
2558 \g_@@_renderers_seq
2559 { tableAttributeContextBegin }
2560 \prop_gput:Nnn
2561 \g_@@_renderer_arities_prop
2562 { tableAttributeContextBegin }
2563 { 0 }
2564 \ExplSyntaxOff
2565 \def\markdownRendererTableAttributeContextEnd{%
2566 \markdownRendererTableAttributeContextEndPrototype}%
2567 \ExplSyntaxOn
2568 \seq_gput_right:Nn
2569 \g_@@_renderers_seq
2570 { tableAttributeContextEnd }
2571 \prop_gput:Nnn
2572 \g_@@_renderer_arities_prop
2573 { tableAttributeContextEnd }
2574 { 0 }
2575 \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.

```

2576 \def\markdownRendererTable{%
2577 \markdownRendererTablePrototype}%
2578 \ExplSyntaxOn
2579 \seq_gput_right:Nn
2580 \g_@@_renderers_seq
2581 { table }
2582 \prop_gput:Nnn
2583 \g_@@_renderer_arities_prop
2584 { table }
2585 { 3 }
2586 \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.

```

2587 \def\markdownRendererInlineMath{%
2588 \markdownRendererInlineMathPrototype}%
2589 \ExplSyntaxOn
2590 \seq_gput_right:Nn
2591 \g_@@_renderers_seq
2592 { inlineMath }
2593 \prop_gput:Nnn
2594 \g_@@_renderer_arities_prop
2595 { inlineMath }
2596 { 1 }
2597 \ExplSyntaxOff
2598 \def\markdownRendererDisplayMath{%
2599 \markdownRendererDisplayMathPrototype}%
2600 \ExplSyntaxOn
2601 \seq_gput_right:Nn
2602 \g_@@_renderers_seq
2603 { displayMath }
2604 \prop_gput:Nnn
2605 \g_@@_renderer_arities_prop
2606 { displayMath }
2607 { 1 }
2608 \ExplSyntaxOff

```

#### 2.2.5.41 Thematic Break Renderer

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

```

2609 \def\markdownRendererThematicBreak{%
2610 \markdownRendererThematicBreakPrototype}%
2611 \ExplSyntaxOn
2612 \seq_gput_right:Nn
2613 \g_@@_renderers_seq
2614 { thematicBreak }
2615 \prop_gput:Nnn
2616 \g_@@_renderer_arities_prop
2617 { thematicBreak }
2618 { 0 }
2619 \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.

```

2620 \def\markdownRendererTickedBox{%
2621 \markdownRendererTickedBoxPrototype}%
2622 \ExplSyntaxOn
2623 \seq_gput_right:Nn
2624 \g_@@_renderers_seq
2625 { tickedBox }
2626 \prop_gput:Nnn
2627 \g_@@_renderer_arities_prop
2628 { tickedBox }
2629 { 0 }
2630 \ExplSyntaxOff
2631 \def\markdownRendererHalfTickedBox{%
2632 \markdownRendererHalfTickedBoxPrototype}%
2633 \ExplSyntaxOn
2634 \seq_gput_right:Nn
2635 \g_@@_renderers_seq
2636 { halfTickedBox }
2637 \prop_gput:Nnn
2638 \g_@@_renderer_arities_prop
2639 { halfTickedBox }
2640 { 0 }
2641 \ExplSyntaxOff
2642 \def\markdownRendererUntickedBox{%
2643 \markdownRendererUntickedBoxPrototype}%
2644 \ExplSyntaxOn

```



```

2645 \seq_gput_right:Nn
2646 \g_@@_renderers_seq
2647 { untickedBox }
2648 \prop_gput:Nnn
2649 \g_@@_renderer_arities_prop
2650 { untickedBox }
2651 { 0 }
2652 \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.

```

2653 \def\markdownRendererWarning{%
2654 \markdownRendererWarningPrototype}%
2655 \def\markdownRendererError{%
2656 \markdownRendererErrorPrototype}%
2657 \ExplSyntaxOn
2658 \seq_gput_right:Nn
2659 \g_@@_renderers_seq
2660 { warning }
2661 \prop_gput:Nnn
2662 \g_@@_renderer_arities_prop
2663 { warning }
2664 { 4 }
2665 \seq_gput_right:Nn
2666 \g_@@_renderers_seq
2667 { error }
2668 \prop_gput:Nnn
2669 \g_@@_renderer_arities_prop
2670 { error }
2671 { 4 }
2672 \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.

```
2673 \def\markdownRendererJekyllDataBegin{%
2674 \markdownRendererJekyllDataBeginPrototype}%
2675 \ExplSyntaxOn
2676 \seq_gput_right:Nn
2677 \g_@@_renderers_seq
2678 { jekyllDataBegin }
2679 \prop_gput:Nnn
2680 \g_@@_renderer_arities_prop
2681 { jekyllDataBegin }
2682 { 0 }
2683 \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.

```
2684 \def\markdownRendererJekyllDataEnd{%
2685 \markdownRendererJekyllDataEndPrototype}%
2686 \ExplSyntaxOn
2687 \seq_gput_right:Nn
2688 \g_@@_renderers_seq
2689 { jekyllDataEnd }
2690 \prop_gput:Nnn
2691 \g_@@_renderer_arities_prop
2692 { jekyllDataEnd }
2693 { 0 }
2694 \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.

```
2695 \def\markdownRendererJekyllDataMappingBegin{%
2696 \markdownRendererJekyllDataMappingBeginPrototype}%
2697 \ExplSyntaxOn
2698 \seq_gput_right:Nn
2699 \g_@@_renderers_seq
2700 { jekyllDataMappingBegin }
2701 \prop_gput:Nnn
2702 \g_@@_renderer_arities_prop
2703 { jekyllDataMappingBegin }
2704 { 2 }
2705 \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.

```
2706 \def\markdownRendererJekyllDataMappingEnd{%
2707 \markdownRendererJekyllDataMappingEndPrototype}%
2708 \ExplSyntaxOn
2709 \seq_gput_right:Nn
2710 \g_@@_renderers_seq
2711 { jekyllDataMappingEnd }
2712 \prop_gput:Nnn
2713 \g_@@_renderer_arities_prop
2714 { jekyllDataMappingEnd }
2715 { 0 }
2716 \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.

```
2717 \def\markdownRendererJekyllDataSequenceBegin{%
2718 \markdownRendererJekyllDataSequenceBeginPrototype}%
2719 \ExplSyntaxOn
2720 \seq_gput_right:Nn
2721 \g_@@_renderers_seq
2722 { jekyllDataSequenceBegin }
2723 \prop_gput:Nnn
2724 \g_@@_renderer_arities_prop
2725 { jekyllDataSequenceBegin }
2726 { 2 }
2727 \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.

```
2728 \def\markdownRendererJekyllDataSequenceEnd{%
2729 \markdownRendererJekyllDataSequenceEndPrototype}%
2730 \ExplSyntaxOn
2731 \seq_gput_right:Nn
2732 \g_@@_renderers_seq
2733 { jekyllDataSequenceEnd }
2734 \prop_gput:Nnn
2735 \g_@@_renderer_arities_prop
2736 { jekyllDataSequenceEnd }
2737 { 0 }
2738 \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.

```

2739 \def\markdownRendererJekyllDataBoolean{%
2740 \markdownRendererJekyllDataBooleanPrototype}%
2741 \ExplSyntaxOn
2742 \seq_gput_right:Nn
2743 \g_@@_renderers_seq
2744 { jekyllDataBoolean }
2745 \prop_gput:Nnn
2746 \g_@@_renderer_arities_prop
2747 { jekyllDataBoolean }
2748 { 2 }
2749 \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.

```

2750 \def\markdownRendererJekyllDataNumber{%
2751 \markdownRendererJekyllDataNumberPrototype}%
2752 \ExplSyntaxOn
2753 \seq_gput_right:Nn
2754 \g_@@_renderers_seq
2755 { jekyllDataNumber }
2756 \prop_gput:Nnn
2757 \g_@@_renderer_arities_prop
2758 { jekyllDataNumber }
2759 { 2 }
2760 \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 T<sub>E</sub>X.

```

2761 \def\markdownRendererJekyllDataTypographicString{%
2762 \markdownRendererJekyllDataTypographicStringPrototype}%
2763 \def\markdownRendererJekyllDataProgrammaticString{%
2764 \markdownRendererJekyllDataProgrammaticStringPrototype}%
2765 \ExplSyntaxOn
2766 \seq_gput_right:Nn
2767 \g_@@_renderers_seq
2768 { jekyllDataTypographicString }
2769 \prop_gput:Nnn
2770 \g_@@_renderer_arities_prop
2771 { jekyllDataTypographicString }
2772 { 2 }
2773 \seq_gput_right:Nn
2774 \g_@@_renderers_seq
2775 { jekyllDataProgrammaticString }
2776 \prop_gput:Nnn
2777 \g_@@_renderer_arities_prop
2778 { jekyllDataProgrammaticString }
2779 { 2 }
2780 \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.

```

2781 \ExplSyntaxOn
2782 \cs_gset:Npn
2783 \markdownRendererJekyllDataTypographicString
2784 {
2785 \cs_if_exist:NTF
2786 \markdownRendererJekyllDataString
2787 {
2788 \markdownWarning
2789 {
2790 The~jekyllDataString~renderer~has~been~deprecated,~
2791 to~be~removed~in~Markdown~4.0.0
2792 }
2793 \markdownRendererJekyllDataString
2794 }
2795 {
2796 \cs_if_exist:NTF
2797 \markdownRendererJekyllDataStringPrototype
2798 {
2799 \markdownWarning

```

```

2800 {
2801 The~jekyllDataString~renderer~prototype~
2802 has~been~deprecated,~
2803 to~be~removed~in~Markdown~4.0.0
2804 }
2805 \markdownRendererJekyllDataStringPrototype
2806 }
2807 {
2808 \markdownRendererJekyllDataTypographicStringPrototype
2809 }
2810 }
2811 }
2812 \seq_gput_right:Nn
2813 \g_@@_renderers_seq
2814 { jekyllDataString }
2815 \prop_gput:Nnn
2816 \g_@@_renderer_arities_prop
2817 { jekyllDataString }
2818 { 2 }
2819 \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.

```

2820 \def\markdownRendererJekyllDataEmpty{%
2821 \markdownRendererJekyllDataEmptyPrototype}%
2822 \ExplSyntaxOn
2823 \seq_gput_right:Nn
2824 \g_@@_renderers_seq
2825 { jekyllDataEmpty }
2826 \prop_gput:Nnn
2827 \g_@@_renderer_arities_prop
2828 { jekyllDataEmpty }
2829 { 1 }
2830 \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` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer macros and the values are new definitions of these token renderers.

```

2831 \ExplSyntaxOn
2832 \cs_new:Nn \@@_define_renderers:
2833 {
2834 \seq_map_inline:Nn
2835 \g_@@_renderers_seq
2836 {
2837 \@@_define_renderer:n
2838 { ##1 }
2839 }
2840 }
2841 \cs_new:Nn \@@_define_renderer:n
2842 {
2843 \@@_renderer_tl_to_csname:nN
2844 { #1 }
2845 \l_tmpa_tl
2846 \prop_get:NnN
2847 \g_@@_renderer_arities_prop
2848 { #1 }
2849 \l_tmpb_tl
2850 \@@_define_renderer:ncV
2851 { #1 }
2852 { \l_tmpa_tl }
2853 \l_tmpb_tl
2854 }
2855 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2856 {
2857 \tl_set:Nn
2858 \l_tmpa_tl
2859 { \str_uppercase:n { #1 } }
2860 \tl_set:Nx
2861 #2
2862 {
2863 markdownRenderer
2864 \tl_head:f { \l_tmpa_tl }
2865 \tl_tail:n { #1 }
2866 }
2867 }
2868 \tl_new:N
2869 \l_@@_renderer_definition_tl
2870 \bool_new:N
2871 \g_@@_appending_renderer_bool
2872 \cs_new:Nn \@@_define_renderer:nNn
2873 {
2874 \keys_define:nn
2875 { markdown/options/renderers }
2876 {
2877 #1 .code:n = {

```

```

2878 \tl_set:Nn
2879 \l_@@_renderer_definition_tl
2880 { ##1 }
2881 \regex_replace_all:nnN
2882 { \cP\#0 }
2883 { #1 }
2884 \l_@@_renderer_definition_tl
2885 \bool_if:NT
2886 \g_@@_appending_renderer_bool
2887 {
2888 \@@_tl_set_from_cs:NNn
2889 \l_tmpa_tl
2890 #2
2891 { #3 }
2892 \tl_put_left:NV
2893 \l_@@_renderer_definition_tl
2894 \l_tmpa_tl
2895 }
2896 \cs_generate_from_arg_count:NNnV
2897 #2
2898 \cs_set:Npn
2899 { #3 }
2900 \l_@@_renderer_definition_tl
2901 },
2902 }

```

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.

```

2903 \str_if_eq:nnT
2904 { #1 }
2905 { jekyllDataString }
2906 {
2907 \cs_undefine:N
2908 #2
2909 }
2910 }

```

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

```

2911 \cs_new_protected:Nn
2912 \@@_tl_set_from_cs:NNn
2913 {
2914 \tl_set:Nn
2915 \l_tmpa_tl

```



```

2916 { #2 }
2917 \int_step_inline:nn
2918 { #3 }
2919 {
2920 \exp_args:NnC
2921 \tl_put_right:Nn
2922 \l_tmpa_tl
2923 { @@_tl_set_from_cs_parameter_ ##1 }
2924 }
2925 \exp_args:NNV
2926 \tl_set:No
2927 \l_tmpb_tl
2928 \l_tmpa_tl
2929 \regex_replace_all:nnN
2930 { \cP. }
2931 { \0\0 }
2932 \l_tmpb_tl
2933 \int_step_inline:nn
2934 { #3 }
2935 {
2936 \regex_replace_all:nnN
2937 { \c { @@_tl_set_from_cs_parameter_ ##1 } }
2938 { \cP\# ##1 }
2939 \l_tmpb_tl
2940 }
2941 \tl_set:NV
2942 #1
2943 \l_tmpb_tl
2944 }
2945 \cs_generate_variant:Nn
2946 \@@_define_renderer:nNn
2947 { ncV }
2948 \cs_generate_variant:Nn
2949 \cs_generate_from_arg_count:NNnn
2950 { NNnV }
2951 \cs_generate_variant:Nn
2952 \tl_put_left:Nn
2953 { Nv }
2954 \keys_define:nn
2955 { markdown/options }
2956 {
2957 renderers .code:n = {
2958 \keys_set:nn
2959 { markdown/options/renderers }
2960 { #1 }
2961 },
2962 }

```

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.
 }
}
```

```
2963 \tl_new:N
2964 \l_@@_renderer_glob_definition_tl
2965 \seq_new:N
2966 \l_@@_renderer_glob_results_seq
2967 \regex_const:Nn
2968 \c_@@_appending_key_regex
2969 { \s*+$ }
2970 \keys_define:nn
2971 { markdown/options/renderers }
2972 {
2973 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 += {...},
 },
},
},
}

```

```

2974 \regex_match:NVTF
2975 \c_@@_appending_key_regex
2976 \l_keys_key_str
2977 {
2978 \bool_gset_true:N
2979 \g_@@_appending_renderer_bool
2980 \tl_set:NV
2981 \l_tmpa_tl
2982 \l_keys_key_str
2983 \regex_replace_once:NnN
2984 \c_@@_appending_key_regex
2985 { }
2986 \l_tmpa_tl
2987 \tl_set:Nx
2988 \l_tmpb_tl
2989 { { \l_tmpa_tl } = }
2990 \tl_put_right:Nn
2991 \l_tmpb_tl
2992 { { #1 } }
2993 \keys_set:nV
2994 { markdown/options/renderers }
2995 \l_tmpb_tl
2996 \bool_gset_false:N
2997 \g_@@_appending_renderer_bool
2998 }

```

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

```

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

```

Wildcards and enumerations can be combined:

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

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

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

```
2999 {
3000 \@@_glob_seq:VnN
3001 \l_keys_key_str
3002 { g_@@_renderers_seq }
3003 \l_@@_renderer_glob_results_seq
3004 \seq_if_empty:NTF
3005 \l_@@_renderer_glob_results_seq
3006 {
3007 \msg_error:nnV
3008 { markdown }
3009 { undefined-renderer }
3010 \l_keys_key_str
3011 }
3012 {
3013 \tl_set:Nn
3014 \l_@@_renderer_glob_definition_tl
3015 { \exp_not:n { #1 } }
3016 \seq_map_inline:Nn
3017 \l_@@_renderer_glob_results_seq
3018 {
3019 \tl_set:Nn
3020 \l_tmpa_tl
3021 { { ##1 } = }
3022 \tl_put_right:Nx
3023 \l_tmpa_tl
3024 { { \l_@@_renderer_glob_definition_tl } }
3025 \keys_set:nV
3026 { markdown/options/renderers }
3027 \l_tmpa_tl
```

```

3028 }
3029 }
3030 }
3031 },
3032 }
3033 \msg_new:nnn
3034 { markdown }
3035 { undefined-renderer }
3036 {
3037 Renderer~#1~is~undefined.
3038 }
3039 \cs_generate_variant:Nn
3040 \@@_glob_seq:nnN
3041 { VnN }
3042 \cs_generate_variant:Nn
3043 \cs_generate_from_arg_count:NNnn
3044 { cNvV }
3045 \cs_generate_variant:Nn
3046 \msg_error:nnn
3047 { nnV }
3048 \prg_generate_conditional_variant:Nnn
3049 \regex_match:Nn
3050 { NV }
3051 { TF }
3052 \prop_new:N
3053 \g_@@_glob_cache_prop
3054 \tl_new:N
3055 \l_@@_current_glob_tl
3056 \cs_new:Nn
3057 \@@_glob_seq:nnN
3058 {
3059 \tl_set:Nn
3060 \l_@@_current_glob_tl
3061 { ^ #1 $ }
3062 \prop_get:NeNTF
3063 \g_@@_glob_cache_prop
3064 { #2 / \l_@@_current_glob_tl }
3065 \l_tmpa_clist
3066 {
3067 \seq_set_from_clist:NN
3068 #3
3069 \l_tmpa_clist
3070 }
3071 {
3072 \seq_clear:N
3073 #3
3074 \regex_replace_all:nnN

```

```

3075 { * }
3076 { .* }
3077 \l_@@_current_glob_tl
3078 \regex_set:NV
3079 \l_tmpa_regex
3080 \l_@@_current_glob_tl
3081 \seq_map_inline:cn
3082 { #2 }
3083 {
3084 \regex_match:NnT
3085 \l_tmpa_regex
3086 { ##1 }
3087 {
3088 \seq_put_right:Nn
3089 #3
3090 { ##1 }
3091 }
3092 }
3093 \clist_set_from_seq:NN
3094 \l_tmpa_clist
3095 #3
3096 \prop_gput:NeV
3097 \g_@@_glob_cache_prop
3098 { #2 / \l_@@_current_glob_tl }
3099 \l_tmpa_clist
3100 }
3101 }
3102 % TODO: Remove in TeX Live 2023.
3103 \prg_generate_conditional_variant:Nnn
3104 \prop_get:NnN
3105 { NeN }
3106 { TF }
3107 \cs_generate_variant:Nn
3108 \regex_set:Nn
3109 { NV }
3110 \cs_generate_variant:Nn
3111 \prop_gput:Nnn
3112 { NeV }

```

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

```

3113 \str_if_eq:VVT
3114 \c_@@_top_layer_tl
3115 \c_@@_option_layer_plain_tex_tl
3116 {
3117 \@@_define_renderers:

```

```

3118 }
3119 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the L<sup>A</sup>T<sub>E</sub>X3 kernel.

```

3120 \ExplSyntaxOn
3121 \keys_define:nn
3122 { markdown/jekyllData }
3123 { }
3124 \ExplSyntaxOff

```

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

```

3125 \ExplSyntaxOn
3126 \@@_with_various_cases:nn
3127 { jekyllDataRenderers }
3128 {
3129 \keys_define:nn
3130 { markdown/options }
3131 {
3132 #1 .code:n = {
3133 \tl_set:Nn
3134 \l_tmpa_tl
3135 { ##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.

```

3136 \tl_replace_all:NnV
3137 \l_tmpa_tl
3138 { / }
3139 \c_backslash_str
3140 \keys_set:nV
3141 { markdown/options/jekyll-data-renderers }
3142 \l_tmpa_tl
3143 },
3144 }
3145 }
3146 \keys_define:nn
3147 { markdown/options/jekyll-data-renderers }

```

```

3148 {
3149 unknown .code:n = {
3150 \tl_set_eq:NN
3151 \l_tmpa_tl
3152 \l_keys_key_str
3153 \tl_replace_all:NVn
3154 \l_tmpa_tl
3155 \c_backslash_str
3156 { / }
3157 \tl_put_right:Nn
3158 \l_tmpa_tl
3159 {
3160 .code:n = { #1 }
3161 }
3162 \keys_define:nV
3163 { markdown/jekyllData }
3164 \l_tmpa_tl
3165 }
3166 }
3167 \cs_generate_variant:Nn
3168 \keys_define:nn
3169 { nV }
3170 \ExplSyntaxOff

```

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

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

```

3171 \ExplSyntaxOn
3172 \cs_new:Nn \@@_define_renderer_prototypes:
3173 {
3174 \seq_map_inline:Nn
3175 \g_@@_renderers_seq
3176 {
3177 \@@_define_renderer_prototype:n
3178 { ##1 }
3179 }
3180 }
3181 \cs_new:Nn \@@_define_renderer_prototype:n
3182 {
3183 \@@_renderer_prototype_tl_to_csname:nN
3184 { #1 }
3185 \l_tmpa_tl

```



```

3186 \prop_get:NnN
3187 \g_@@_renderer_arities_prop
3188 { #1 }
3189 \l_tmpb_tl
3190 \@@_define_renderer_prototype:ncV
3191 { #1 }
3192 { \l_tmpa_tl }
3193 \l_tmpb_tl
3194 }
3195 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3196 {
3197 \tl_set:Nn
3198 \l_tmpa_tl
3199 { \str_uppercase:n { #1 } }
3200 \tl_set:Nx
3201 #2
3202 {
3203 markdownRenderer
3204 \tl_head:f { \l_tmpa_tl }
3205 \tl_tail:n { #1 }
3206 Prototype
3207 }
3208 }
3209 \tl_new:N
3210 \l_@@_renderer_prototype_definition_tl
3211 \bool_new:N
3212 \g_@@_appending_renderer_prototype_bool
3213 \cs_new:Nn \@@_define_renderer_prototype:nNn
3214 {
3215 \keys_define:nn
3216 { markdown/options/renderer-prototypes }
3217 {
3218 #1 .code:n = {
3219 \tl_set:Nn
3220 \l_@@_renderer_prototype_definition_tl
3221 { ##1 }
3222 \regex_replace_all:nnN
3223 { \cP\#0 }
3224 { #1 }
3225 \l_@@_renderer_prototype_definition_tl
3226 \bool_if:NT
3227 \g_@@_appending_renderer_prototype_bool
3228 {
3229 \@@_tl_set_from_cs:NNn
3230 \l_tmpa_tl
3231 #2
3232 { #3 }

```

```

3233 \tl_put_left:NV
3234 \l_@@_renderer_prototype_definition_tl
3235 \l_tmpa_tl
3236 }
3237 \cs_generate_from_arg_count:NNnV
3238 #2
3239 \cs_set:Npn
3240 { #3 }
3241 \l_@@_renderer_prototype_definition_tl
3242 },
3243 }

```

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.

```

3244 \str_if_eq:nnF
3245 { #1 }
3246 { jekyllDataString }
3247 {
3248 \cs_if_free:NT
3249 #2
3250 {
3251 \cs_generate_from_arg_count:NNnn
3252 #2
3253 \cs_set:Npn
3254 { #3 }
3255 { }
3256 }
3257 }
3258 }
3259 \cs_generate_variant:Nn
3260 \@@_define_renderer_prototype:nNn
3261 { 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.
 }
}

```

```

3262 \keys_define:nn
3263 { markdown/options/renderer-prototypes }
3264 {
3265 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 += {...},
 },
 }
 },
 },
}

```

```

3266 \regex_match:NVTf
3267 \c_@@_appending_key_regex
3268 \l_keys_key_str
3269 {
3270 \bool_gset_true:N
3271 \g_@@_appending_renderer_prototype_bool
3272 \tl_set:Nv
3273 \l_tmpa_tl
3274 \l_keys_key_str
3275 \regex_replace_once:NnN

```

```

3276 \c_@@_appending_key_regex
3277 { }
3278 \l_tmpa_tl
3279 \tl_set:Nx
3280 \l_tmpb_tl
3281 { { \l_tmpa_tl } = }
3282 \tl_put_right:Nn
3283 \l_tmpb_tl
3284 { { #1 } }
3285 \keys_set:nV
3286 { markdown/options/renderer-prototypes }
3287 \l_tmpb_tl
3288 \bool_gset_false:N
3289 \g_@@_appending_renderer_prototype_bool
3290 }

```

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

```

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

```

Wildcards and enumerations can be combined:

```

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

```

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

```

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

```

```

3291 {
3292 \@@_glob_seq:VnN
3293 \l_keys_key_str
3294 { g_@@_renderers_seq }
3295 \l_@@_renderer_glob_results_seq
3296 \seq_if_empty:NTF
3297 \l_@@_renderer_glob_results_seq
3298 {
3299 \msg_error:nnV
3300 { markdown }
3301 { undefined-renderer-prototype }
3302 \l_keys_key_str
3303 }
3304 {
3305 \tl_set:Nn
3306 \l_@@_renderer_glob_definition_tl
3307 { \exp_not:n { #1 } }
3308 \seq_map_inline:Nn
3309 \l_@@_renderer_glob_results_seq
3310 {
3311 \tl_set:Nn
3312 \l_tmpa_tl
3313 { { ##1 } = }
3314 \tl_put_right:Nx
3315 \l_tmpa_tl
3316 { { \l_@@_renderer_glob_definition_tl } }
3317 \keys_set:nV
3318 { markdown/options/renderer-prototypes }
3319 \l_tmpa_tl
3320 }
3321 }
3322 }
3323 },
3324 }
3325 \msg_new:nnn
3326 { markdown }
3327 { undefined-renderer-prototype }
3328 {
3329 Renderer~prototype~#1~is~undefined.
3330 }
3331 \@@_with_various_cases:nn
3332 { rendererPrototypes }
3333 {
3334 \keys_define:nn
3335 { markdown/options }
3336 {
3337 #1 .code:n = {

```

```

3338 \keys_set:nn
3339 { markdown/options/renderer-prototypes }
3340 { ##1 }
3341 },
3342 }
3343 }

```

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

```

3344 \str_if_eq:VVT
3345 \c_@@_top_layer_tl
3346 \c_@@_option_layer_plain_tex_tl
3347 {
3348 \@@_define_renderer_prototypes:
3349 }
3350 \ExplSyntaxOff

```

## 2.2.7 Logging Facilities

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

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

## 2.2.8 Miscellanea

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

```

3351 \let\markdownMakeOther\relax

```

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

```

3352 \let\markdownReadAndConvert\relax
3353 \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*.

```

3354 \catcode`\|=0\catcode`\=12%
3355 |gdef|markdownBegin{%
3356 |markdownReadAndConvert{\markdownEnd}%
3357 {\markdownEnd}}%
3358 |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.

```

3359 \ExplSyntaxOn
3360 \keys_define:nn
3361 { markdown/options }
3362 {
3363 code .code:n = { #1 },
3364 }
3365 \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.

```

3366 \ExplSyntaxOn
3367 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3368 \cs_generate_variant:Nn
3369 \tl_const:Nn
3370 { NV }
3371 \tl_if_exist:NF
3372 \c_@@_top_layer_tl
3373 {
3374 \tl_const:NV
3375 \c_@@_top_layer_tl
3376 \c_@@_option_layer_latex_tl
3377 }
3378 \ExplSyntaxOff

```

3379 `\input markdown/markdown`

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

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

where *<options>* are the  $\LaTeX$  interface options (see Section 2.3.2). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single  $\LaTeX$  theme (see Section 2.3.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way  $\LaTeX 2_\epsilon$  parses package options.

### 2.3.1 Typesetting Markdown

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

#### 2.3.1.1 The `markdown` and `markdown*` $\LaTeX$ environments

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

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

```
3380 \newenvironment{markdown}\relax\relax
3381 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\markdownEnd` macro to produce special effects before and after the `markdown`  $\LaTeX$  environment (and likewise for the starred version).

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

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

```
\documentclass{article} \documentclass{article}
\usepackage{markdown} \usepackage{markdown}
\begin{document} \begin{document}
% ... % ...
\begin{markdown}[smartEllipses] \begin{markdown*}{smartEllipses}
```



<code>_Hello_ **world** ...</code>	<code>_Hello_ **world** ...</code>
<code>\end{markdown}</code>	<code>\end{markdown*}</code>
<code>% ...</code>	<code>% ...</code>
<code>\end{document}</code>	<code>\end{document}</code>

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 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 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 is 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}
```

### 2.3.1.2 The `\markinline` and `\markdownInput` macros

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 2.3.2) as its optional argument. These options will only influence this markdown content.

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

The following example  $\text{\LaTeX}$  code showcases the usage of the `\markdownInput` macro:

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

## 2.3.2 Options

The  $\text{\LaTeX}$  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.

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

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

### 2.3.2.1 Finalizing and Freezing the Cache

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

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

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

```
3382 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3383 \DeclareOption{frozenscache}{\markdownSetup{frozensCache}}
```

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

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

```
3384 \ExplSyntaxOn
3385 \str_if_eq:VVT
3386 \c_@@_top_layer_tl
3387 \c_@@_option_layer_latex_tl
3388 {
3389 \@@_define_option_commands_and_keyvals:
3390 \@@_define_renderers:
3391 \@@_define_renderer_prototypes:
3392 }
3393 \ExplSyntaxOff
```

The following example  $\LaTeX$  code showcases a possible configuration of plain  $\TeX$  interface options `hybrid`, `smartEllipses`, and `cacheDir`.

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

### 2.3.3 Themes

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

Having the Markdown package automatically load either the generic `.tex theme file` or the  $\LaTeX$ -specific `.sty` theme file allows developers to have a single *theme file*, when the theme is small or the difference between  $\TeX$  formats is unimportant, and

scale up to separate theme files native to different T<sub>E</sub>X formats for large multi-format themes, where different code is needed for different T<sub>E</sub>X formats. To enable code reuse, developers can load the `.tex` theme file from the `.sty` theme file using the `\markdownLoadPlainTeXTheme` macro.

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

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

```
3394 \newif\ifmarkdownLaTeXLoaded
3395 \markdownLaTeXLoadedfalse
```

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

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

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

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

  latex -> pmml;
```

```

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

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

```

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

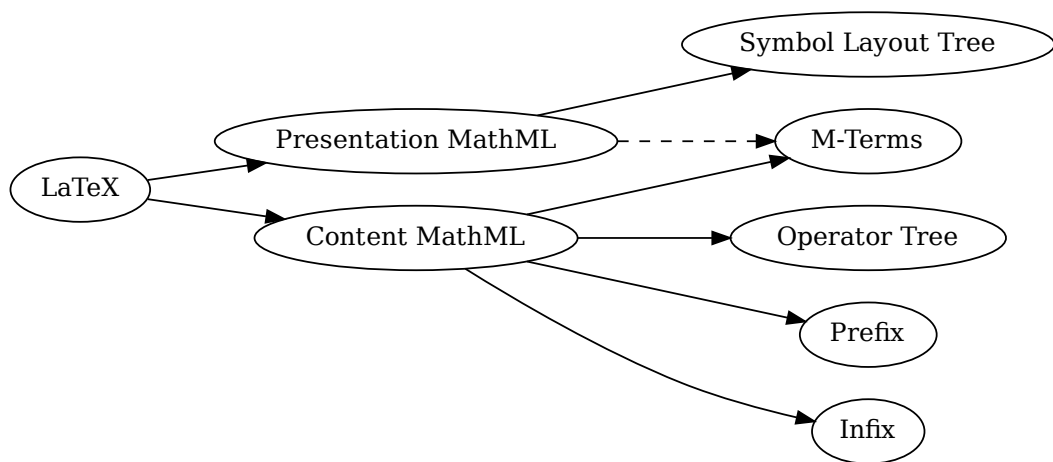


Figure 4: Various formats of mathematical formulae

The theme requires a Unix-like operating system with GNU Diffutils and

Graphviz installed. The theme also requires shell access unless the `frozenCache` plain `TEX` option is enabled.

```
3396 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%
```

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

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

Typesetting the above document produces the output shown in Figure 5. The

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

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

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



Chapter 1

Introduction

1.1 Section

1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

Figure 5: The banner of the Markdown package

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

```
3397 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

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

```
3398 \AtEndOfPackage{
3399   \markdownLaTeXLoadedtrue
```

At the end of the L^AT_EX module, we load the **witiko/markdown/defaults** L^AT_EX 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).

```
3400   \markdownIfOption{noDefaults}{-}{
3401     \markdownSetup{theme=witiko/markdown/defaults}
3402   }
3403 }
```

```
3404 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%
```

Please, see Section 3.3.2 for implementation details of the built-in L^AT_EX themes.

2.4 ConT_EXt Interface

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

```
3405 \ExplSyntaxOn
3406 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3407 \cs_generate_variant:Nn
3408   \tl_const:Nn
3409   { NV }
3410 \tl_if_exist:NF
3411   \c_@@_top_layer_tl
3412   {
3413     \tl_const:NV
3414       \c_@@_top_layer_tl
3415       \c_@@_option_layer_context_tl
3416   }
3417 \ExplSyntaxOff
```

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

```
3418 \writestatus{loading}{ConTeXt User Module / markdown}%
3419 \startmodule[markdown]
3420 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3421   \do#\do\^\do\_do\%do\~}%

```

3422 `\input markdown/markdown`

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

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

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

2.4.1 Typesetting Markdown

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

3423 `\let\startmarkdown\relax`

3424 `\let\stopmarkdown\relax`

3425 `\let\inputmarkdown\relax`

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

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

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

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

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

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


```

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

```

2.4.2 Options

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

ConT_EXt options map directly to the options recognized by the plain T_EX interface (see Section 2.2.2).

The ConT_EXt 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}`.

```

3426 \ExplSyntaxOn
3427 \cs_new:Npn
3428   \setupmarkdown
3429   [ #1 ]
3430   {
3431     \@@_setup:n
3432     { #1 }
3433   }
3434 \ExplSyntaxOff

```

2.4.2.1 Generating Plain T_EX Option Macros and Key-Values

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

```

3435 \ExplSyntaxOn
3436 \cs_new:Nn \@@_caseless:N
3437   {
3438     \regex_replace_all:nnN
3439     { ([a-z])([A-Z]) }
3440     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3441     #1
3442     \tl_set:Nx
3443     #1
3444     { #1 }
3445   }
3446 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

If ConT_EXt is the top layer, we use the `\@@_define_option_commands_and_keyvals:`, `\@@_define_renderers:`, and `\@@_define_renderer_prototypes:` macro to de-

fine 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.

```

3447 \str_if_eq:VVT
3448   \c_@@_top_layer_tl
3449   \c_@@_option_layer_context_tl
3450   {
3451     \@@_define_option_commands_and_keyvals:
3452     \@@_define_renderers:
3453     \@@_define_renderer_prototypes:
3454   }
3455 \ExplSyntaxOff

```

2.4.3 Themes

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

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

For example, to load a theme named `witiko/tilde` in your document:

```

\usemodule[t][markdown]
\setupmarkdown[import=witiko/tilde]

```

Built-in Con \TeX t themes provided with the Markdown package include:

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

```

3456 \startmodule[markdownthemewitiko_markdown_defaults]
3457 \unprotect

```

Please, see Section 3.4.2 for implementation details of the built-in Con \TeX t 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 $\text{T}_{\text{E}}\text{X}$ *token renderers* is performed by the Lua layer. The plain $\text{T}_{\text{E}}\text{X}$ layer provides default definitions for the token renderers. The \LaTeX and $\text{ConT}_{\text{E}}\text{Xt}$ layers correct idiosyncrasies of the respective $\text{T}_{\text{E}}\text{X}$ formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

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

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

```
3458 local upper, format, length =
3459   string.upper, string.format, string.len
3460 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3461   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3462   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 $\text{T}_{\text{E}}\text{X}$ writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
3463 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.

```
3464 function util.err(msg, exit_code)
3465   io.stderr:write("markdown.lua: " .. msg .. "\n")
3466   os.exit(exit_code or 1)
3467 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. Regardless, the pathname is then returned.

```

3468 function util.cache(dir, string, salt, transform, suffix)
3469   local digest = md5.sumhexa(string .. (salt or ""))
3470   local name = util.pathname(dir, digest .. suffix)
3471   local file = io.open(name, "r")
3472   if file == nil then -- If no cache entry exists, create a new one.
3473     file = assert(io.open(name, "w"),
3474       [[Could not open file ]] .. name .. [[ for writing]])
3475     local result = string
3476     if transform ~= nil then
3477       result = transform(result)
3478     end
3479     assert(file:write(result))
3480     assert(file:close())
3481   end
3482   return name
3483 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.

```

3484 function util.cache_verbatim(dir, string)
3485   local name = util.cache(dir, string, nil, nil, ".verbatim")
3486   return name
3487 end

```

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

```

3488 function util.table_copy(t)
3489   local u = { }
3490   for k, v in pairs(t) do u[k] = v end
3491   return setmetatable(u, getmetatable(t))
3492 end

```

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

```

3493 function util.encode_json_string(s)
3494   s = s:gsub([[\\]], [[\\]])
3495   s = s:gsub([[\"]], [[\"]])
3496   return [[\"]] .. s .. [[\"]]
3497 end

```

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

```

3498 function util.expand_tabs_in_line(s, tabstop)
3499   local tab = tabstop or 4
3500   local corr = 0
3501   return (s:gsub(")\t", function(p)
3502     local sp = tab - (p - 1 + corr) % tab

```

```

3503         corr = corr - 1 + sp
3504         return string.rep(" ", sp)
3505     end))
3506 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.

```

3507 function util.walk(t, f)
3508     local typ = type(t)
3509     if typ == "string" then
3510         f(t)
3511     elseif typ == "table" then
3512         local i = 1
3513         local n
3514         n = t[i]
3515         while n do
3516             util.walk(n, f)
3517             i = i + 1
3518             n = t[i]
3519         end
3520     elseif typ == "function" then
3521         local ok, val = pcall(t)
3522         if ok then
3523             util.walk(val, f)
3524         end
3525     else
3526         f(tostring(t))
3527     end
3528 end

```

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

```

3529 function util.flatten(ary)
3530     local new = {}
3531     for _,v in ipairs(ary) do
3532         if type(v) == "table" then
3533             for _,w in ipairs(util.flatten(v)) do
3534                 new[#new + 1] = w
3535             end
3536         else
3537             new[#new + 1] = v
3538         end
3539     end
3540     return new
3541 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.

```
3542 function util.rope_to_string(rope)
3543   local buffer = {}
3544   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3545   return table.concat(buffer)
3546 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.

```
3547 function util.rope_last(rope)
3548   if #rope == 0 then
3549     return nil
3550   else
3551     local l = rope[#rope]
3552     if type(l) == "table" then
3553       return util.rope_last(l)
3554     else
3555       return l
3556     end
3557   end
3558 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 $1 \leq i \leq \#ary$.

```
3559 function util.intersperse(ary, x)
3560   local new = {}
3561   local l = #ary
3562   for i,v in ipairs(ary) do
3563     local n = #new
3564     new[n + 1] = v
3565     if i ~= l then
3566       new[n + 2] = x
3567     end
3568   end
3569   return new
3570 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 $1 \leq i \leq \#ary$.

```
3571 function util.map(ary, f)
3572   local new = {}
3573   for i,v in ipairs(ary) do
3574     new[i] = f(v)
3575   end
3576   return new
3577 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.

```
3578 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3579 local char_escapes_list = ""
3580 for i,_ in pairs(char_escapes) do
3581   char_escapes_list = char_escapes_list .. i
3582 end
```

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

```
3583 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.

```
3584 if string_escapes then
3585   for k,v in pairs(string_escapes) do
3586     escapable = P(k) / v + escapable
3587   end
3588 end
```

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

```
3589 local escape_string = Cs((escapable + any)^0)
```

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

```
3590 return function(s)
3591   return lpeg.match(escape_string, s)
3592 end
3593 end
```

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

```
3594 function util.pathname(dir, file)
3595   if #dir == 0 then
3596     return file
```

```

3597     else
3598         return dir .. "/" .. file
3599     end
3600 end

```

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

```

3601 function util.salt(options)
3602     local opt_string = {}
3603     for k, _ in pairs(defaultOptions) do
3604         local v = options[k]
3605         if type(v) == "table" then
3606             for _, i in ipairs(v) do
3607                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
3608             end

```

The `cacheDir` option is disregarded.

```

3609         elseif k ~= "cacheDir" then
3610             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
3611         end
3612     end
3613     table.sort(opt_string)
3614     local salt = table.concat(opt_string, ",")
3615                 .. "," .. metadata.version
3616     return salt
3617 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.

```

3618 function util.warning(s)
3619     io.stderr:write("Warning: " .. s .. "\n")
3620 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.

```

3621 local entities = {}
3622
3623 local character_entities = {
3624     ["Tab"] = 9,
3625     ["NewLine"] = 10,
3626     ["excl"] = 33,
3627     ["QUOT"] = 34,
3628     ["quot"] = 34,

```


3629 ["num"] = 35,
3630 ["dollar"] = 36,
3631 ["percent"] = 37,
3632 ["AMP"] = 38,
3633 ["amp"] = 38,
3634 ["apos"] = 39,
3635 ["lpar"] = 40,
3636 ["rpar"] = 41,
3637 ["ast"] = 42,
3638 ["midast"] = 42,
3639 ["plus"] = 43,
3640 ["comma"] = 44,
3641 ["period"] = 46,
3642 ["sol"] = 47,
3643 ["colon"] = 58,
3644 ["semi"] = 59,
3645 ["LT"] = 60,
3646 ["lt"] = 60,
3647 ["nvlt"] = {60, 8402},
3648 ["bne"] = {61, 8421},
3649 ["equals"] = 61,
3650 ["GT"] = 62,
3651 ["gt"] = 62,
3652 ["nvgt"] = {62, 8402},
3653 ["quest"] = 63,
3654 ["commat"] = 64,
3655 ["lbrack"] = 91,
3656 ["lsqb"] = 91,
3657 ["bsol"] = 92,
3658 ["rbrack"] = 93,
3659 ["rsqb"] = 93,
3660 ["Hat"] = 94,
3661 ["UnderBar"] = 95,
3662 ["lowbar"] = 95,
3663 ["DiacriticalGrave"] = 96,
3664 ["grave"] = 96,
3665 ["fjlig"] = {102, 106},
3666 ["lbrace"] = 123,
3667 ["lcub"] = 123,
3668 ["VerticalLine"] = 124,
3669 ["verbar"] = 124,
3670 ["vert"] = 124,
3671 ["rbrace"] = 125,
3672 ["rcub"] = 125,
3673 ["NonBreakingSpace"] = 160,
3674 ["nbsp"] = 160,
3675 ["iexcl"] = 161,

3676 ["cent"] = 162,
3677 ["pound"] = 163,
3678 ["curren"] = 164,
3679 ["yen"] = 165,
3680 ["brvbar"] = 166,
3681 ["sect"] = 167,
3682 ["Dot"] = 168,
3683 ["DoubleDot"] = 168,
3684 ["die"] = 168,
3685 ["uml"] = 168,
3686 ["COPY"] = 169,
3687 ["copy"] = 169,
3688 ["ordf"] = 170,
3689 ["laquo"] = 171,
3690 ["not"] = 172,
3691 ["shy"] = 173,
3692 ["REG"] = 174,
3693 ["circledR"] = 174,
3694 ["reg"] = 174,
3695 ["macr"] = 175,
3696 ["strns"] = 175,
3697 ["deg"] = 176,
3698 ["PlusMinus"] = 177,
3699 ["plusmn"] = 177,
3700 ["pm"] = 177,
3701 ["sup2"] = 178,
3702 ["sup3"] = 179,
3703 ["DiacriticalAcute"] = 180,
3704 ["acute"] = 180,
3705 ["micro"] = 181,
3706 ["para"] = 182,
3707 ["CenterDot"] = 183,
3708 ["centerdot"] = 183,
3709 ["middot"] = 183,
3710 ["Cedilla"] = 184,
3711 ["cedil"] = 184,
3712 ["sup1"] = 185,
3713 ["ordm"] = 186,
3714 ["raquo"] = 187,
3715 ["frac14"] = 188,
3716 ["frac12"] = 189,
3717 ["half"] = 189,
3718 ["frac34"] = 190,
3719 ["iquest"] = 191,
3720 ["Agrave"] = 192,
3721 ["Aacute"] = 193,
3722 ["Acirc"] = 194,

3723 ["Atilde"] = 195,
3724 ["Auml"] = 196,
3725 ["Aring"] = 197,
3726 ["angst"] = 197,
3727 ["AElig"] = 198,
3728 ["Ccedil"] = 199,
3729 ["Egrave"] = 200,
3730 ["Eacute"] = 201,
3731 ["Ecirc"] = 202,
3732 ["Euml"] = 203,
3733 ["Igrave"] = 204,
3734 ["Iacute"] = 205,
3735 ["Icirc"] = 206,
3736 ["Iuml"] = 207,
3737 ["ETH"] = 208,
3738 ["Ntilde"] = 209,
3739 ["Ograve"] = 210,
3740 ["Oacute"] = 211,
3741 ["Ocirc"] = 212,
3742 ["Otilde"] = 213,
3743 ["Ouml"] = 214,
3744 ["times"] = 215,
3745 ["Oslash"] = 216,
3746 ["Ugrave"] = 217,
3747 ["Uacute"] = 218,
3748 ["Ucirc"] = 219,
3749 ["Uuml"] = 220,
3750 ["Yacute"] = 221,
3751 ["THORN"] = 222,
3752 ["szlig"] = 223,
3753 ["agrave"] = 224,
3754 ["aacute"] = 225,
3755 ["acirc"] = 226,
3756 ["atilde"] = 227,
3757 ["auml"] = 228,
3758 ["aring"] = 229,
3759 ["aelig"] = 230,
3760 ["ccedil"] = 231,
3761 ["egrave"] = 232,
3762 ["eacute"] = 233,
3763 ["ecirc"] = 234,
3764 ["euml"] = 235,
3765 ["igrave"] = 236,
3766 ["iacute"] = 237,
3767 ["icirc"] = 238,
3768 ["iuml"] = 239,
3769 ["eth"] = 240,

3770 ["ntilde"] = 241,
3771 ["ograve"] = 242,
3772 ["oacute"] = 243,
3773 ["ocirc"] = 244,
3774 ["otilde"] = 245,
3775 ["ouml"] = 246,
3776 ["div"] = 247,
3777 ["divide"] = 247,
3778 ["oslash"] = 248,
3779 ["ugrave"] = 249,
3780 ["uacute"] = 250,
3781 ["ucirc"] = 251,
3782 ["uuml"] = 252,
3783 ["yacute"] = 253,
3784 ["thorn"] = 254,
3785 ["yuml"] = 255,
3786 ["Amacr"] = 256,
3787 ["amacr"] = 257,
3788 ["Abreve"] = 258,
3789 ["abreve"] = 259,
3790 ["Aogon"] = 260,
3791 ["aogon"] = 261,
3792 ["Cacute"] = 262,
3793 ["cacute"] = 263,
3794 ["Ccirc"] = 264,
3795 ["ccirc"] = 265,
3796 ["Cdot"] = 266,
3797 ["cdot"] = 267,
3798 ["Ccaron"] = 268,
3799 ["ccaron"] = 269,
3800 ["Dcaron"] = 270,
3801 ["dcaron"] = 271,
3802 ["Dstrok"] = 272,
3803 ["dstrok"] = 273,
3804 ["Emacr"] = 274,
3805 ["emacr"] = 275,
3806 ["Edot"] = 278,
3807 ["edot"] = 279,
3808 ["Eogon"] = 280,
3809 ["eogon"] = 281,
3810 ["Ecaron"] = 282,
3811 ["ecaron"] = 283,
3812 ["Gcirc"] = 284,
3813 ["gcirc"] = 285,
3814 ["Gbreve"] = 286,
3815 ["gbreve"] = 287,
3816 ["Gdot"] = 288,

3817 ["gdot"] = 289,
3818 ["Gcedil"] = 290,
3819 ["Hcirc"] = 292,
3820 ["hcirc"] = 293,
3821 ["Hstrook"] = 294,
3822 ["hstrook"] = 295,
3823 ["Itilde"] = 296,
3824 ["itilde"] = 297,
3825 ["Imacr"] = 298,
3826 ["imacr"] = 299,
3827 ["Iogon"] = 302,
3828 ["iogon"] = 303,
3829 ["Idot"] = 304,
3830 ["imath"] = 305,
3831 ["inodot"] = 305,
3832 ["IJlig"] = 306,
3833 ["ijlig"] = 307,
3834 ["Jcirc"] = 308,
3835 ["jcirc"] = 309,
3836 ["Kcedil"] = 310,
3837 ["kcedil"] = 311,
3838 ["kgreen"] = 312,
3839 ["Lacute"] = 313,
3840 ["lacute"] = 314,
3841 ["Lcedil"] = 315,
3842 ["lcedil"] = 316,
3843 ["Lcaron"] = 317,
3844 ["lcaron"] = 318,
3845 ["Lmidot"] = 319,
3846 ["lmidot"] = 320,
3847 ["Lstrook"] = 321,
3848 ["lstrook"] = 322,
3849 ["Nacute"] = 323,
3850 ["nacute"] = 324,
3851 ["Ncedil"] = 325,
3852 ["ncedil"] = 326,
3853 ["Ncaron"] = 327,
3854 ["ncaron"] = 328,
3855 ["napos"] = 329,
3856 ["ENG"] = 330,
3857 ["eng"] = 331,
3858 ["Omacr"] = 332,
3859 ["omacr"] = 333,
3860 ["Odblac"] = 336,
3861 ["odblac"] = 337,
3862 ["OElig"] = 338,
3863 ["oelig"] = 339,

3864 ["Racute"] = 340,
3865 ["racute"] = 341,
3866 ["Rcedil"] = 342,
3867 ["rcedil"] = 343,
3868 ["Rcaron"] = 344,
3869 ["rcaron"] = 345,
3870 ["Sacute"] = 346,
3871 ["sacute"] = 347,
3872 ["Scirc"] = 348,
3873 ["scirc"] = 349,
3874 ["Scedil"] = 350,
3875 ["scedil"] = 351,
3876 ["Scaron"] = 352,
3877 ["scaron"] = 353,
3878 ["Tcedil"] = 354,
3879 ["tcedil"] = 355,
3880 ["Tcaron"] = 356,
3881 ["tcaron"] = 357,
3882 ["Tstrok"] = 358,
3883 ["tstrok"] = 359,
3884 ["Utilde"] = 360,
3885 ["utilde"] = 361,
3886 ["Umacr"] = 362,
3887 ["umacr"] = 363,
3888 ["Ubreve"] = 364,
3889 ["ubreve"] = 365,
3890 ["Uring"] = 366,
3891 ["uring"] = 367,
3892 ["Udblac"] = 368,
3893 ["udblac"] = 369,
3894 ["Uogon"] = 370,
3895 ["uogon"] = 371,
3896 ["Wcirc"] = 372,
3897 ["wcirc"] = 373,
3898 ["Ycirc"] = 374,
3899 ["ycirc"] = 375,
3900 ["Yuml"] = 376,
3901 ["Zacute"] = 377,
3902 ["zacute"] = 378,
3903 ["Zdot"] = 379,
3904 ["zdot"] = 380,
3905 ["Zcaron"] = 381,
3906 ["zcaron"] = 382,
3907 ["fnof"] = 402,
3908 ["imped"] = 437,
3909 ["gacute"] = 501,
3910 ["jmath"] = 567,

3911 ["circ"] = 710,
3912 ["Hacek"] = 711,
3913 ["caron"] = 711,
3914 ["Breve"] = 728,
3915 ["breve"] = 728,
3916 ["DiacriticalDot"] = 729,
3917 ["dot"] = 729,
3918 ["ring"] = 730,
3919 ["ogon"] = 731,
3920 ["DiacriticalTilde"] = 732,
3921 ["tilde"] = 732,
3922 ["DiacriticalDoubleAcute"] = 733,
3923 ["dblac"] = 733,
3924 ["DownBreve"] = 785,
3925 ["Alpha"] = 913,
3926 ["Beta"] = 914,
3927 ["Gamma"] = 915,
3928 ["Delta"] = 916,
3929 ["Epsilon"] = 917,
3930 ["Zeta"] = 918,
3931 ["Eta"] = 919,
3932 ["Theta"] = 920,
3933 ["Iota"] = 921,
3934 ["Kappa"] = 922,
3935 ["Lambda"] = 923,
3936 ["Mu"] = 924,
3937 ["Nu"] = 925,
3938 ["Xi"] = 926,
3939 ["Omicron"] = 927,
3940 ["Pi"] = 928,
3941 ["Rho"] = 929,
3942 ["Sigma"] = 931,
3943 ["Tau"] = 932,
3944 ["Upsilon"] = 933,
3945 ["Phi"] = 934,
3946 ["Chi"] = 935,
3947 ["Psi"] = 936,
3948 ["Omega"] = 937,
3949 ["ohm"] = 937,
3950 ["alpha"] = 945,
3951 ["beta"] = 946,
3952 ["gamma"] = 947,
3953 ["delta"] = 948,
3954 ["epsi"] = 949,
3955 ["epsilon"] = 949,
3956 ["zeta"] = 950,
3957 ["eta"] = 951,

3958 ["theta"] = 952,
 3959 ["iota"] = 953,
 3960 ["kappa"] = 954,
 3961 ["lambda"] = 955,
 3962 ["mu"] = 956,
 3963 ["nu"] = 957,
 3964 ["xi"] = 958,
 3965 ["omicron"] = 959,
 3966 ["pi"] = 960,
 3967 ["rho"] = 961,
 3968 ["sigmaf"] = 962,
 3969 ["sigmav"] = 962,
 3970 ["varsigma"] = 962,
 3971 ["sigma"] = 963,
 3972 ["tau"] = 964,
 3973 ["upsilon"] = 965,
 3974 ["upsilon"] = 965,
 3975 ["phi"] = 966,
 3976 ["chi"] = 967,
 3977 ["psi"] = 968,
 3978 ["omega"] = 969,
 3979 ["thetasym"] = 977,
 3980 ["thetav"] = 977,
 3981 ["vartheta"] = 977,
 3982 ["Upsilon"] = 978,
 3983 ["upsih"] = 978,
 3984 ["phiv"] = 981,
 3985 ["straightphi"] = 981,
 3986 ["varphi"] = 981,
 3987 ["piv"] = 982,
 3988 ["varpi"] = 982,
 3989 ["Gammad"] = 988,
 3990 ["digamma"] = 989,
 3991 ["gammad"] = 989,
 3992 ["kappav"] = 1008,
 3993 ["varkappa"] = 1008,
 3994 ["rhov"] = 1009,
 3995 ["varrho"] = 1009,
 3996 ["epsiv"] = 1013,
 3997 ["straightepsilon"] = 1013,
 3998 ["varepsilon"] = 1013,
 3999 ["backepsilon"] = 1014,
 4000 ["bepsi"] = 1014,
 4001 ["IOcy"] = 1025,
 4002 ["DJcy"] = 1026,
 4003 ["GJcy"] = 1027,
 4004 ["Jukcy"] = 1028,

4005 ["DScy"] = 1029,
4006 ["Iukcy"] = 1030,
4007 ["YIcy"] = 1031,
4008 ["Jsercy"] = 1032,
4009 ["LJcy"] = 1033,
4010 ["NJcy"] = 1034,
4011 ["TSHcy"] = 1035,
4012 ["KJcy"] = 1036,
4013 ["Ubrcy"] = 1038,
4014 ["DZcy"] = 1039,
4015 ["Acy"] = 1040,
4016 ["Bcy"] = 1041,
4017 ["Vcy"] = 1042,
4018 ["Gcy"] = 1043,
4019 ["Dcy"] = 1044,
4020 ["IEcy"] = 1045,
4021 ["ZHcy"] = 1046,
4022 ["Zcy"] = 1047,
4023 ["Icy"] = 1048,
4024 ["Jcy"] = 1049,
4025 ["Kcy"] = 1050,
4026 ["Lcy"] = 1051,
4027 ["Mcy"] = 1052,
4028 ["Ncy"] = 1053,
4029 ["Ocy"] = 1054,
4030 ["Pcy"] = 1055,
4031 ["Rcy"] = 1056,
4032 ["Scy"] = 1057,
4033 ["Tcy"] = 1058,
4034 ["Ucy"] = 1059,
4035 ["Fcy"] = 1060,
4036 ["KHcy"] = 1061,
4037 ["TScy"] = 1062,
4038 ["CHcy"] = 1063,
4039 ["SHcy"] = 1064,
4040 ["SHCHcy"] = 1065,
4041 ["HARDcy"] = 1066,
4042 ["Ycy"] = 1067,
4043 ["SOFTcy"] = 1068,
4044 ["Ecy"] = 1069,
4045 ["YUcy"] = 1070,
4046 ["YAcy"] = 1071,
4047 ["acy"] = 1072,
4048 ["bcy"] = 1073,
4049 ["vcy"] = 1074,
4050 ["gcy"] = 1075,
4051 ["dcy"] = 1076,

4052 ["iecy"] = 1077,
4053 ["zhcy"] = 1078,
4054 ["zcy"] = 1079,
4055 ["icy"] = 1080,
4056 ["jcy"] = 1081,
4057 ["kcy"] = 1082,
4058 ["lcy"] = 1083,
4059 ["mcy"] = 1084,
4060 ["ncy"] = 1085,
4061 ["ocy"] = 1086,
4062 ["pcy"] = 1087,
4063 ["rcy"] = 1088,
4064 ["scy"] = 1089,
4065 ["tcy"] = 1090,
4066 ["ucy"] = 1091,
4067 ["fcy"] = 1092,
4068 ["khcy"] = 1093,
4069 ["tscy"] = 1094,
4070 ["chcy"] = 1095,
4071 ["shcy"] = 1096,
4072 ["shchcy"] = 1097,
4073 ["hardcy"] = 1098,
4074 ["ycy"] = 1099,
4075 ["softcy"] = 1100,
4076 ["ecy"] = 1101,
4077 ["yucy"] = 1102,
4078 ["yacy"] = 1103,
4079 ["iocy"] = 1105,
4080 ["djcy"] = 1106,
4081 ["gjcy"] = 1107,
4082 ["jukcy"] = 1108,
4083 ["dscy"] = 1109,
4084 ["iukcy"] = 1110,
4085 ["yicy"] = 1111,
4086 ["jsercy"] = 1112,
4087 ["ljcy"] = 1113,
4088 ["njcy"] = 1114,
4089 ["tshcy"] = 1115,
4090 ["kjcy"] = 1116,
4091 ["ubrcy"] = 1118,
4092 ["dzcy"] = 1119,
4093 ["ensp"] = 8194,
4094 ["emsp"] = 8195,
4095 ["emsp13"] = 8196,
4096 ["emsp14"] = 8197,
4097 ["numsp"] = 8199,
4098 ["puncsp"] = 8200,

4099 ["ThinSpace"] = 8201,
4100 ["thinsp"] = 8201,
4101 ["VeryThinSpace"] = 8202,
4102 ["hairsp"] = 8202,
4103 ["NegativeMediumSpace"] = 8203,
4104 ["NegativeThickSpace"] = 8203,
4105 ["NegativeThinSpace"] = 8203,
4106 ["NegativeVeryThinSpace"] = 8203,
4107 ["ZeroWidthSpace"] = 8203,
4108 ["zwnj"] = 8204,
4109 ["zwj"] = 8205,
4110 ["lrm"] = 8206,
4111 ["rlm"] = 8207,
4112 ["dash"] = 8208,
4113 ["hyphen"] = 8208,
4114 ["ndash"] = 8211,
4115 ["mdash"] = 8212,
4116 ["horbar"] = 8213,
4117 ["Verbar"] = 8214,
4118 ["Vert"] = 8214,
4119 ["OpenCurlyQuote"] = 8216,
4120 ["lsquo"] = 8216,
4121 ["CloseCurlyQuote"] = 8217,
4122 ["rsquo"] = 8217,
4123 ["rsquor"] = 8217,
4124 ["lsquor"] = 8218,
4125 ["sbquo"] = 8218,
4126 ["OpenCurlyDoubleQuote"] = 8220,
4127 ["ldquo"] = 8220,
4128 ["CloseCurlyDoubleQuote"] = 8221,
4129 ["rdquo"] = 8221,
4130 ["rdquor"] = 8221,
4131 ["bdquo"] = 8222,
4132 ["ldquor"] = 8222,
4133 ["dagger"] = 8224,
4134 ["Dagger"] = 8225,
4135 ["ddagger"] = 8225,
4136 ["bull"] = 8226,
4137 ["bullet"] = 8226,
4138 ["nldr"] = 8229,
4139 ["hellip"] = 8230,
4140 ["mldr"] = 8230,
4141 ["permil"] = 8240,
4142 ["pertenk"] = 8241,
4143 ["prime"] = 8242,
4144 ["Prime"] = 8243,
4145 ["tprime"] = 8244,

4146 ["backprime"] = 8245,
 4147 ["bprime"] = 8245,
 4148 ["lsaquo"] = 8249,
 4149 ["rsaquo"] = 8250,
 4150 ["OverBar"] = 8254,
 4151 ["oline"] = 8254,
 4152 ["caret"] = 8257,
 4153 ["hybull"] = 8259,
 4154 ["frasl"] = 8260,
 4155 ["bsemi"] = 8271,
 4156 ["qprime"] = 8279,
 4157 ["MediumSpace"] = 8287,
 4158 ["ThickSpace"] = {8287, 8202},
 4159 ["NoBreak"] = 8288,
 4160 ["ApplyFunction"] = 8289,
 4161 ["af"] = 8289,
 4162 ["InvisibleTimes"] = 8290,
 4163 ["it"] = 8290,
 4164 ["InvisibleComma"] = 8291,
 4165 ["ic"] = 8291,
 4166 ["euro"] = 8364,
 4167 ["TripleDot"] = 8411,
 4168 ["tdot"] = 8411,
 4169 ["DotDot"] = 8412,
 4170 ["Copf"] = 8450,
 4171 ["complexes"] = 8450,
 4172 ["incare"] = 8453,
 4173 ["gscr"] = 8458,
 4174 ["HilbertSpace"] = 8459,
 4175 ["Hscr"] = 8459,
 4176 ["hamilt"] = 8459,
 4177 ["Hfr"] = 8460,
 4178 ["Poincareplane"] = 8460,
 4179 ["Hopf"] = 8461,
 4180 ["quaternions"] = 8461,
 4181 ["planckh"] = 8462,
 4182 ["hbar"] = 8463,
 4183 ["hslash"] = 8463,
 4184 ["planck"] = 8463,
 4185 ["plankv"] = 8463,
 4186 ["Iscr"] = 8464,
 4187 ["imagline"] = 8464,
 4188 ["Ifr"] = 8465,
 4189 ["Im"] = 8465,
 4190 ["image"] = 8465,
 4191 ["imagpart"] = 8465,
 4192 ["Laplacetrnf"] = 8466,

4193 ["Lscr"] = 8466,
4194 ["lagran"] = 8466,
4195 ["ell"] = 8467,
4196 ["Nopf"] = 8469,
4197 ["naturals"] = 8469,
4198 ["numero"] = 8470,
4199 ["copysr"] = 8471,
4200 ["weierp"] = 8472,
4201 ["wp"] = 8472,
4202 ["Popf"] = 8473,
4203 ["primes"] = 8473,
4204 ["Qopf"] = 8474,
4205 ["rationals"] = 8474,
4206 ["Rscr"] = 8475,
4207 ["realine"] = 8475,
4208 ["Re"] = 8476,
4209 ["Rfr"] = 8476,
4210 ["real"] = 8476,
4211 ["realpart"] = 8476,
4212 ["Ropf"] = 8477,
4213 ["reals"] = 8477,
4214 ["rx"] = 8478,
4215 ["TRADE"] = 8482,
4216 ["trade"] = 8482,
4217 ["Zopf"] = 8484,
4218 ["integers"] = 8484,
4219 ["mho"] = 8487,
4220 ["Zfr"] = 8488,
4221 ["zeetrf"] = 8488,
4222 ["iiota"] = 8489,
4223 ["Bernoullis"] = 8492,
4224 ["Bscr"] = 8492,
4225 ["bernou"] = 8492,
4226 ["Cayleys"] = 8493,
4227 ["Cfr"] = 8493,
4228 ["escr"] = 8495,
4229 ["Escr"] = 8496,
4230 ["expectation"] = 8496,
4231 ["Fouriertrf"] = 8497,
4232 ["Fscr"] = 8497,
4233 ["Mellintrf"] = 8499,
4234 ["Mscr"] = 8499,
4235 ["phmmat"] = 8499,
4236 ["order"] = 8500,
4237 ["orderof"] = 8500,
4238 ["oscr"] = 8500,
4239 ["alefsym"] = 8501,

4240 ["aleph"] = 8501,
 4241 ["beth"] = 8502,
 4242 ["gimel"] = 8503,
 4243 ["daleth"] = 8504,
 4244 ["CapitalDifferentialD"] = 8517,
 4245 ["DD"] = 8517,
 4246 ["DifferentialD"] = 8518,
 4247 ["dd"] = 8518,
 4248 ["ExponentialE"] = 8519,
 4249 ["ee"] = 8519,
 4250 ["exponentiale"] = 8519,
 4251 ["ImaginaryI"] = 8520,
 4252 ["ii"] = 8520,
 4253 ["frac13"] = 8531,
 4254 ["frac23"] = 8532,
 4255 ["frac15"] = 8533,
 4256 ["frac25"] = 8534,
 4257 ["frac35"] = 8535,
 4258 ["frac45"] = 8536,
 4259 ["frac16"] = 8537,
 4260 ["frac56"] = 8538,
 4261 ["frac18"] = 8539,
 4262 ["frac38"] = 8540,
 4263 ["frac58"] = 8541,
 4264 ["frac78"] = 8542,
 4265 ["LeftArrow"] = 8592,
 4266 ["ShortLeftArrow"] = 8592,
 4267 ["larr"] = 8592,
 4268 ["leftarrow"] = 8592,
 4269 ["slarr"] = 8592,
 4270 ["ShortUpArrow"] = 8593,
 4271 ["UpArrow"] = 8593,
 4272 ["uarr"] = 8593,
 4273 ["uparrow"] = 8593,
 4274 ["RightArrow"] = 8594,
 4275 ["ShortRightArrow"] = 8594,
 4276 ["rarr"] = 8594,
 4277 ["rightarrow"] = 8594,
 4278 ["srarr"] = 8594,
 4279 ["DownArrow"] = 8595,
 4280 ["ShortDownArrow"] = 8595,
 4281 ["darr"] = 8595,
 4282 ["downarrow"] = 8595,
 4283 ["LeftRightArrow"] = 8596,
 4284 ["harr"] = 8596,
 4285 ["leftrightarrow"] = 8596,
 4286 ["UpDownArrow"] = 8597,

4287 ["updownarrow"] = 8597,
 4288 ["varr"] = 8597,
 4289 ["UpperLeftArrow"] = 8598,
 4290 ["nwarr"] = 8598,
 4291 ["nwarrow"] = 8598,
 4292 ["UpperRightArrow"] = 8599,
 4293 ["nearr"] = 8599,
 4294 ["nearrow"] = 8599,
 4295 ["LowerRightArrow"] = 8600,
 4296 ["searr"] = 8600,
 4297 ["searrow"] = 8600,
 4298 ["LowerLeftArrow"] = 8601,
 4299 ["swarr"] = 8601,
 4300 ["swarrow"] = 8601,
 4301 ["nlarr"] = 8602,
 4302 ["nleftarrow"] = 8602,
 4303 ["nrarr"] = 8603,
 4304 ["nrightarrow"] = 8603,
 4305 ["nrarrw"] = {8605, 824},
 4306 ["rarrw"] = 8605,
 4307 ["rightsquigarrow"] = 8605,
 4308 ["Larr"] = 8606,
 4309 ["twoheadleftarrow"] = 8606,
 4310 ["Uarr"] = 8607,
 4311 ["Rarr"] = 8608,
 4312 ["twoheadrightarrow"] = 8608,
 4313 ["Darr"] = 8609,
 4314 ["larrtl"] = 8610,
 4315 ["leftarrowtail"] = 8610,
 4316 ["rarrtl"] = 8611,
 4317 ["rightarrowtail"] = 8611,
 4318 ["LeftTeeArrow"] = 8612,
 4319 ["mapstoleft"] = 8612,
 4320 ["UpTeeArrow"] = 8613,
 4321 ["mapstoup"] = 8613,
 4322 ["RightTeeArrow"] = 8614,
 4323 ["map"] = 8614,
 4324 ["mapsto"] = 8614,
 4325 ["DownTeeArrow"] = 8615,
 4326 ["mapstodown"] = 8615,
 4327 ["hookleftarrow"] = 8617,
 4328 ["larrhk"] = 8617,
 4329 ["hookrightarrow"] = 8618,
 4330 ["rarrhk"] = 8618,
 4331 ["larrlp"] = 8619,
 4332 ["looparrowleft"] = 8619,
 4333 ["looparrowright"] = 8620,

4334 ["rarrlp"] = 8620,
 4335 ["harrw"] = 8621,
 4336 ["leftrightsquigarrow"] = 8621,
 4337 ["nharr"] = 8622,
 4338 ["nleftrightarrow"] = 8622,
 4339 ["Lsh"] = 8624,
 4340 ["lsh"] = 8624,
 4341 ["Rsh"] = 8625,
 4342 ["rsh"] = 8625,
 4343 ["ldsh"] = 8626,
 4344 ["rdsh"] = 8627,
 4345 ["crarr"] = 8629,
 4346 ["cularr"] = 8630,
 4347 ["curvearrowleft"] = 8630,
 4348 ["curarr"] = 8631,
 4349 ["curvearrowright"] = 8631,
 4350 ["circlearrowleft"] = 8634,
 4351 ["olarr"] = 8634,
 4352 ["circlearrowright"] = 8635,
 4353 ["orarr"] = 8635,
 4354 ["LeftVector"] = 8636,
 4355 ["leftharpoonup"] = 8636,
 4356 ["lharu"] = 8636,
 4357 ["DownLeftVector"] = 8637,
 4358 ["leftharpoondown"] = 8637,
 4359 ["lhard"] = 8637,
 4360 ["RightUpVector"] = 8638,
 4361 ["uharr"] = 8638,
 4362 ["upharpoonright"] = 8638,
 4363 ["LeftUpVector"] = 8639,
 4364 ["uharl"] = 8639,
 4365 ["upharpoonleft"] = 8639,
 4366 ["RightVector"] = 8640,
 4367 ["rharu"] = 8640,
 4368 ["rightharpoonup"] = 8640,
 4369 ["DownRightVector"] = 8641,
 4370 ["rhard"] = 8641,
 4371 ["rightharpoondown"] = 8641,
 4372 ["RightDownVector"] = 8642,
 4373 ["dharr"] = 8642,
 4374 ["downharpoonright"] = 8642,
 4375 ["LeftDownVector"] = 8643,
 4376 ["dharl"] = 8643,
 4377 ["downharpoonleft"] = 8643,
 4378 ["RightArrowLeftArrow"] = 8644,
 4379 ["rightleftarrows"] = 8644,
 4380 ["rlarr"] = 8644,

4381 ["UpArrowDownArrow"] = 8645,
 4382 ["udarr"] = 8645,
 4383 ["LeftArrowRightArrow"] = 8646,
 4384 ["leftrightarrows"] = 8646,
 4385 ["lrarr"] = 8646,
 4386 ["leftleftarrows"] = 8647,
 4387 ["llarr"] = 8647,
 4388 ["upuparrows"] = 8648,
 4389 ["uuarr"] = 8648,
 4390 ["rightrightarrows"] = 8649,
 4391 ["rrarr"] = 8649,
 4392 ["ddarr"] = 8650,
 4393 ["downdownarrows"] = 8650,
 4394 ["ReverseEquilibrium"] = 8651,
 4395 ["leftrightharpoons"] = 8651,
 4396 ["lrhar"] = 8651,
 4397 ["Equilibrium"] = 8652,
 4398 ["rightleftharpoons"] = 8652,
 4399 ["rlhar"] = 8652,
 4400 ["nLeftarrow"] = 8653,
 4401 ["nlArr"] = 8653,
 4402 ["nLeftrightarrow"] = 8654,
 4403 ["nhArr"] = 8654,
 4404 ["nRrightarrow"] = 8655,
 4405 ["nrArr"] = 8655,
 4406 ["DoubleLeftArrow"] = 8656,
 4407 ["Leftarrow"] = 8656,
 4408 ["lArr"] = 8656,
 4409 ["DoubleUpArrow"] = 8657,
 4410 ["Uparrow"] = 8657,
 4411 ["uArr"] = 8657,
 4412 ["DoubleRightArrow"] = 8658,
 4413 ["Implies"] = 8658,
 4414 ["Rrightarrow"] = 8658,
 4415 ["rArr"] = 8658,
 4416 ["DoubleDownArrow"] = 8659,
 4417 ["Downarrow"] = 8659,
 4418 ["dArr"] = 8659,
 4419 ["DoubleLeftRightArrow"] = 8660,
 4420 ["Leftrightarrow"] = 8660,
 4421 ["hArr"] = 8660,
 4422 ["iff"] = 8660,
 4423 ["DoubleUpDownArrow"] = 8661,
 4424 ["Updownarrow"] = 8661,
 4425 ["vArr"] = 8661,
 4426 ["nwArr"] = 8662,
 4427 ["neArr"] = 8663,

4428 ["seArr"] = 8664,
 4429 ["swArr"] = 8665,
 4430 ["Lleftarrow"] = 8666,
 4431 ["lAarr"] = 8666,
 4432 ["Rightarrow"] = 8667,
 4433 ["rAarr"] = 8667,
 4434 ["zigrarr"] = 8669,
 4435 ["LeftArrowBar"] = 8676,
 4436 ["larrb"] = 8676,
 4437 ["RightArrowBar"] = 8677,
 4438 ["rarrb"] = 8677,
 4439 ["DownArrowUpArrow"] = 8693,
 4440 ["duarr"] = 8693,
 4441 ["loarr"] = 8701,
 4442 ["roarr"] = 8702,
 4443 ["hoarr"] = 8703,
 4444 ["ForAll"] = 8704,
 4445 ["forall"] = 8704,
 4446 ["comp"] = 8705,
 4447 ["complement"] = 8705,
 4448 ["PartialD"] = 8706,
 4449 ["npart"] = {8706, 824},
 4450 ["part"] = 8706,
 4451 ["Exists"] = 8707,
 4452 ["exist"] = 8707,
 4453 ["NotExists"] = 8708,
 4454 ["nexist"] = 8708,
 4455 ["nexists"] = 8708,
 4456 ["empty"] = 8709,
 4457 ["emptyset"] = 8709,
 4458 ["emptyv"] = 8709,
 4459 ["varnothing"] = 8709,
 4460 ["Del"] = 8711,
 4461 ["nabla"] = 8711,
 4462 ["Element"] = 8712,
 4463 ["in"] = 8712,
 4464 ["isin"] = 8712,
 4465 ["isinv"] = 8712,
 4466 ["NotElement"] = 8713,
 4467 ["notin"] = 8713,
 4468 ["notinva"] = 8713,
 4469 ["ReverseElement"] = 8715,
 4470 ["SuchThat"] = 8715,
 4471 ["ni"] = 8715,
 4472 ["niv"] = 8715,
 4473 ["NotReverseElement"] = 8716,
 4474 ["notni"] = 8716,

4475 ["notniva"] = 8716,
 4476 ["Product"] = 8719,
 4477 ["prod"] = 8719,
 4478 ["Coproduct"] = 8720,
 4479 ["coprod"] = 8720,
 4480 ["Sum"] = 8721,
 4481 ["sum"] = 8721,
 4482 ["minus"] = 8722,
 4483 ["MinusPlus"] = 8723,
 4484 ["mnplus"] = 8723,
 4485 ["mp"] = 8723,
 4486 ["dotplus"] = 8724,
 4487 ["plusdo"] = 8724,
 4488 ["Backslash"] = 8726,
 4489 ["setminus"] = 8726,
 4490 ["setmn"] = 8726,
 4491 ["smallsetminus"] = 8726,
 4492 ["ssetmn"] = 8726,
 4493 ["lowast"] = 8727,
 4494 ["SmallCircle"] = 8728,
 4495 ["compfn"] = 8728,
 4496 ["Sqrt"] = 8730,
 4497 ["radic"] = 8730,
 4498 ["Proportional"] = 8733,
 4499 ["prop"] = 8733,
 4500 ["propto"] = 8733,
 4501 ["varpropto"] = 8733,
 4502 ["vprop"] = 8733,
 4503 ["infin"] = 8734,
 4504 ["angrt"] = 8735,
 4505 ["ang"] = 8736,
 4506 ["angle"] = 8736,
 4507 ["nang"] = {8736, 8402},
 4508 ["angmsd"] = 8737,
 4509 ["measuredangle"] = 8737,
 4510 ["angsph"] = 8738,
 4511 ["VerticalBar"] = 8739,
 4512 ["mid"] = 8739,
 4513 ["shortmid"] = 8739,
 4514 ["smid"] = 8739,
 4515 ["NotVerticalBar"] = 8740,
 4516 ["nmid"] = 8740,
 4517 ["nshortmid"] = 8740,
 4518 ["nsmid"] = 8740,
 4519 ["DoubleVerticalBar"] = 8741,
 4520 ["par"] = 8741,
 4521 ["parallel"] = 8741,

4522 ["shortparallel"] = 8741,
4523 ["spar"] = 8741,
4524 ["NotDoubleVerticalBar"] = 8742,
4525 ["npar"] = 8742,
4526 ["nparallel"] = 8742,
4527 ["nshortparallel"] = 8742,
4528 ["nspar"] = 8742,
4529 ["and"] = 8743,
4530 ["wedge"] = 8743,
4531 ["or"] = 8744,
4532 ["vee"] = 8744,
4533 ["cap"] = 8745,
4534 ["caps"] = {8745, 65024},
4535 ["cup"] = 8746,
4536 ["cups"] = {8746, 65024},
4537 ["Integral"] = 8747,
4538 ["int"] = 8747,
4539 ["Int"] = 8748,
4540 ["iiint"] = 8749,
4541 ["tint"] = 8749,
4542 ["ContourIntegral"] = 8750,
4543 ["conint"] = 8750,
4544 ["oint"] = 8750,
4545 ["Conint"] = 8751,
4546 ["DoubleContourIntegral"] = 8751,
4547 ["Cconint"] = 8752,
4548 ["cwint"] = 8753,
4549 ["ClockwiseContourIntegral"] = 8754,
4550 ["cwconint"] = 8754,
4551 ["CounterClockwiseContourIntegral"] = 8755,
4552 ["awconint"] = 8755,
4553 ["Therefore"] = 8756,
4554 ["there4"] = 8756,
4555 ["therefore"] = 8756,
4556 ["Because"] = 8757,
4557 ["because"] = 8757,
4558 ["because"] = 8757,
4559 ["ratio"] = 8758,
4560 ["Colon"] = 8759,
4561 ["Proportion"] = 8759,
4562 ["dotminus"] = 8760,
4563 ["minusd"] = 8760,
4564 ["mDDot"] = 8762,
4565 ["homtht"] = 8763,
4566 ["Tilde"] = 8764,
4567 ["nvsim"] = {8764, 8402},
4568 ["sim"] = 8764,

4569 ["thicksim"] = 8764,
 4570 ["thksim"] = 8764,
 4571 ["backsim"] = 8765,
 4572 ["bsim"] = 8765,
 4573 ["race"] = {8765, 817},
 4574 ["ac"] = 8766,
 4575 ["acE"] = {8766, 819},
 4576 ["mstpos"] = 8766,
 4577 ["acd"] = 8767,
 4578 ["VerticalTilde"] = 8768,
 4579 ["wr"] = 8768,
 4580 ["wreath"] = 8768,
 4581 ["NotTilde"] = 8769,
 4582 ["nsim"] = 8769,
 4583 ["EqualTilde"] = 8770,
 4584 ["NotEqualTilde"] = {8770, 824},
 4585 ["eqsim"] = 8770,
 4586 ["esim"] = 8770,
 4587 ["nesim"] = {8770, 824},
 4588 ["TildeEqual"] = 8771,
 4589 ["sime"] = 8771,
 4590 ["simeq"] = 8771,
 4591 ["NotTildeEqual"] = 8772,
 4592 ["nsime"] = 8772,
 4593 ["nsimeq"] = 8772,
 4594 ["TildeFullEqual"] = 8773,
 4595 ["cong"] = 8773,
 4596 ["simne"] = 8774,
 4597 ["NotTildeFullEqual"] = 8775,
 4598 ["ncong"] = 8775,
 4599 ["TildeTilde"] = 8776,
 4600 ["ap"] = 8776,
 4601 ["approx"] = 8776,
 4602 ["asyp"] = 8776,
 4603 ["thickapprox"] = 8776,
 4604 ["thkap"] = 8776,
 4605 ["NotTildeTilde"] = 8777,
 4606 ["nap"] = 8777,
 4607 ["napprox"] = 8777,
 4608 ["ape"] = 8778,
 4609 ["approxpeq"] = 8778,
 4610 ["apid"] = 8779,
 4611 ["napid"] = {8779, 824},
 4612 ["backcong"] = 8780,
 4613 ["bcong"] = 8780,
 4614 ["CupCap"] = 8781,
 4615 ["asympeq"] = 8781,

4616 ["nvap"] = {8781, 8402},
 4617 ["Bumpeq"] = 8782,
 4618 ["HumpDownHump"] = 8782,
 4619 ["NotHumpDownHump"] = {8782, 824},
 4620 ["bump"] = 8782,
 4621 ["nbump"] = {8782, 824},
 4622 ["HumpEqual"] = 8783,
 4623 ["NotHumpEqual"] = {8783, 824},
 4624 ["bumpe"] = 8783,
 4625 ["bumpeq"] = 8783,
 4626 ["nbumpe"] = {8783, 824},
 4627 ["DotEqual"] = 8784,
 4628 ["doteq"] = 8784,
 4629 ["esdot"] = 8784,
 4630 ["nedot"] = {8784, 824},
 4631 ["doteqdot"] = 8785,
 4632 ["eDot"] = 8785,
 4633 ["efDot"] = 8786,
 4634 ["fallingdotseq"] = 8786,
 4635 ["erDot"] = 8787,
 4636 ["risingdotseq"] = 8787,
 4637 ["Assign"] = 8788,
 4638 ["colone"] = 8788,
 4639 ["coloneq"] = 8788,
 4640 ["ecolon"] = 8789,
 4641 ["eqcolon"] = 8789,
 4642 ["ecir"] = 8790,
 4643 ["eqcirc"] = 8790,
 4644 ["circeq"] = 8791,
 4645 ["cire"] = 8791,
 4646 ["wedgeq"] = 8793,
 4647 ["veeeq"] = 8794,
 4648 ["triangleq"] = 8796,
 4649 ["trie"] = 8796,
 4650 ["equest"] = 8799,
 4651 ["questeq"] = 8799,
 4652 ["NotEqual"] = 8800,
 4653 ["ne"] = 8800,
 4654 ["Congruent"] = 8801,
 4655 ["bnequiv"] = {8801, 8421},
 4656 ["equiv"] = 8801,
 4657 ["NotCongruent"] = 8802,
 4658 ["nequiv"] = 8802,
 4659 ["le"] = 8804,
 4660 ["leq"] = 8804,
 4661 ["nvle"] = {8804, 8402},
 4662 ["GreaterEqual"] = 8805,

4663 ["ge"] = 8805,
 4664 ["geq"] = 8805,
 4665 ["nvge"] = {8805, 8402},
 4666 ["LessFullEqual"] = 8806,
 4667 ["lE"] = 8806,
 4668 ["leqq"] = 8806,
 4669 ["nlE"] = {8806, 824},
 4670 ["nleqq"] = {8806, 824},
 4671 ["GreaterFullEqual"] = 8807,
 4672 ["NotGreaterFullEqual"] = {8807, 824},
 4673 ["gE"] = 8807,
 4674 ["geqq"] = 8807,
 4675 ["ngE"] = {8807, 824},
 4676 ["ngeqq"] = {8807, 824},
 4677 ["lnE"] = 8808,
 4678 ["lneqq"] = 8808,
 4679 ["lvertneqq"] = {8808, 65024},
 4680 ["lvnE"] = {8808, 65024},
 4681 ["gnE"] = 8809,
 4682 ["gneqq"] = 8809,
 4683 ["gvertneqq"] = {8809, 65024},
 4684 ["gvnE"] = {8809, 65024},
 4685 ["Lt"] = 8810,
 4686 ["NestedLessLess"] = 8810,
 4687 ["NotLessLess"] = {8810, 824},
 4688 ["l1"] = 8810,
 4689 ["nLt"] = {8810, 8402},
 4690 ["nLtv"] = {8810, 824},
 4691 ["Gt"] = 8811,
 4692 ["NestedGreaterGreater"] = 8811,
 4693 ["NotGreaterGreater"] = {8811, 824},
 4694 ["gg"] = 8811,
 4695 ["nGt"] = {8811, 8402},
 4696 ["nGtv"] = {8811, 824},
 4697 ["between"] = 8812,
 4698 ["twixt"] = 8812,
 4699 ["NotCupCap"] = 8813,
 4700 ["NotLess"] = 8814,
 4701 ["nless"] = 8814,
 4702 ["nlt"] = 8814,
 4703 ["NotGreater"] = 8815,
 4704 ["ngt"] = 8815,
 4705 ["ngtr"] = 8815,
 4706 ["NotLessEqual"] = 8816,
 4707 ["nle"] = 8816,
 4708 ["nleq"] = 8816,
 4709 ["NotGreaterEqual"] = 8817,

4710 ["nge"] = 8817,
 4711 ["ngeq"] = 8817,
 4712 ["LessTilde"] = 8818,
 4713 ["lesssim"] = 8818,
 4714 ["lsim"] = 8818,
 4715 ["GreaterTilde"] = 8819,
 4716 ["gsim"] = 8819,
 4717 ["gtrsim"] = 8819,
 4718 ["NotLessTilde"] = 8820,
 4719 ["nlsim"] = 8820,
 4720 ["NotGreaterTilde"] = 8821,
 4721 ["ngsim"] = 8821,
 4722 ["LessGreater"] = 8822,
 4723 ["lessgtr"] = 8822,
 4724 ["lg"] = 8822,
 4725 ["GreaterLess"] = 8823,
 4726 ["gl"] = 8823,
 4727 ["gtrless"] = 8823,
 4728 ["NotLessGreater"] = 8824,
 4729 ["ntlgl"] = 8824,
 4730 ["NotGreaterLess"] = 8825,
 4731 ["ntgl"] = 8825,
 4732 ["Precedes"] = 8826,
 4733 ["pr"] = 8826,
 4734 ["prec"] = 8826,
 4735 ["Succeeds"] = 8827,
 4736 ["sc"] = 8827,
 4737 ["succ"] = 8827,
 4738 ["PrecedesSlantEqual"] = 8828,
 4739 ["prcue"] = 8828,
 4740 ["preccurlyeq"] = 8828,
 4741 ["SucceedsSlantEqual"] = 8829,
 4742 ["sccue"] = 8829,
 4743 ["succcurlyeq"] = 8829,
 4744 ["PrecedesTilde"] = 8830,
 4745 ["precsim"] = 8830,
 4746 ["prsim"] = 8830,
 4747 ["NotSucceedsTilde"] = {8831, 824},
 4748 ["SucceedsTilde"] = 8831,
 4749 ["scsim"] = 8831,
 4750 ["succsim"] = 8831,
 4751 ["NotPrecedes"] = 8832,
 4752 ["npr"] = 8832,
 4753 ["nprec"] = 8832,
 4754 ["NotSucceeds"] = 8833,
 4755 ["nsc"] = 8833,
 4756 ["nsucc"] = 8833,

4757 ["NotSubset"] = {8834, 8402},
4758 ["nsubset"] = {8834, 8402},
4759 ["sub"] = 8834,
4760 ["subset"] = 8834,
4761 ["vnsup"] = {8834, 8402},
4762 ["NotSuperset"] = {8835, 8402},
4763 ["Superset"] = 8835,
4764 ["nsupset"] = {8835, 8402},
4765 ["sup"] = 8835,
4766 ["supset"] = 8835,
4767 ["vnsup"] = {8835, 8402},
4768 ["nsub"] = 8836,
4769 ["nsup"] = 8837,
4770 ["SubsetEqual"] = 8838,
4771 ["sube"] = 8838,
4772 ["subseteq"] = 8838,
4773 ["SupersetEqual"] = 8839,
4774 ["supe"] = 8839,
4775 ["supseteq"] = 8839,
4776 ["NotSubsetEqual"] = 8840,
4777 ["nsube"] = 8840,
4778 ["nsubseteq"] = 8840,
4779 ["NotSupersetEqual"] = 8841,
4780 ["nsupe"] = 8841,
4781 ["nsupseteq"] = 8841,
4782 ["subne"] = 8842,
4783 ["subsetneq"] = 8842,
4784 ["varsubsetneq"] = {8842, 65024},
4785 ["vsubne"] = {8842, 65024},
4786 ["supne"] = 8843,
4787 ["supsetneq"] = 8843,
4788 ["varsupsetneq"] = {8843, 65024},
4789 ["vsupne"] = {8843, 65024},
4790 ["cupdot"] = 8845,
4791 ["UnionPlus"] = 8846,
4792 ["uplus"] = 8846,
4793 ["NotSquareSubset"] = {8847, 824},
4794 ["SquareSubset"] = 8847,
4795 ["sqsub"] = 8847,
4796 ["sqsubset"] = 8847,
4797 ["NotSquareSuperset"] = {8848, 824},
4798 ["SquareSuperset"] = 8848,
4799 ["sqsup"] = 8848,
4800 ["sqsupset"] = 8848,
4801 ["SquareSubsetEqual"] = 8849,
4802 ["sqsube"] = 8849,
4803 ["sqsubseteq"] = 8849,

4804 ["SquareSupersetEqual"] = 8850,
 4805 ["sqsupe"] = 8850,
 4806 ["sqsupseteq"] = 8850,
 4807 ["SquareIntersection"] = 8851,
 4808 ["sqcap"] = 8851,
 4809 ["sqcaps"] = {8851, 65024},
 4810 ["SquareUnion"] = 8852,
 4811 ["sqcup"] = 8852,
 4812 ["sqcups"] = {8852, 65024},
 4813 ["CirclePlus"] = 8853,
 4814 ["oplus"] = 8853,
 4815 ["CircleMinus"] = 8854,
 4816 ["ominus"] = 8854,
 4817 ["CircleTimes"] = 8855,
 4818 ["otimes"] = 8855,
 4819 ["osol"] = 8856,
 4820 ["CircleDot"] = 8857,
 4821 ["odot"] = 8857,
 4822 ["circledcirc"] = 8858,
 4823 ["ocir"] = 8858,
 4824 ["circledast"] = 8859,
 4825 ["oast"] = 8859,
 4826 ["circleddash"] = 8861,
 4827 ["odash"] = 8861,
 4828 ["boxplus"] = 8862,
 4829 ["plusb"] = 8862,
 4830 ["boxminus"] = 8863,
 4831 ["minusb"] = 8863,
 4832 ["boxtimes"] = 8864,
 4833 ["timesb"] = 8864,
 4834 ["dotsquare"] = 8865,
 4835 ["sdotb"] = 8865,
 4836 ["RightTee"] = 8866,
 4837 ["vdash"] = 8866,
 4838 ["LeftTee"] = 8867,
 4839 ["dashv"] = 8867,
 4840 ["DownTee"] = 8868,
 4841 ["top"] = 8868,
 4842 ["UpTee"] = 8869,
 4843 ["bot"] = 8869,
 4844 ["bottom"] = 8869,
 4845 ["perp"] = 8869,
 4846 ["models"] = 8871,
 4847 ["DoubleRightTee"] = 8872,
 4848 ["vDash"] = 8872,
 4849 ["Vdash"] = 8873,
 4850 ["Vvdash"] = 8874,

4851 ["VDash"] = 8875,
 4852 ["nvdash"] = 8876,
 4853 ["nvDash"] = 8877,
 4854 ["nVdash"] = 8878,
 4855 ["nVDash"] = 8879,
 4856 ["prurel"] = 8880,
 4857 ["LeftTriangle"] = 8882,
 4858 ["vartriangleleft"] = 8882,
 4859 ["vltri"] = 8882,
 4860 ["RightTriangle"] = 8883,
 4861 ["vartriangleright"] = 8883,
 4862 ["vrtri"] = 8883,
 4863 ["LeftTriangleEqual"] = 8884,
 4864 ["ltrie"] = 8884,
 4865 ["nvltrie"] = {8884, 8402},
 4866 ["trianglelefteq"] = 8884,
 4867 ["RightTriangleEqual"] = 8885,
 4868 ["nvrtrie"] = {8885, 8402},
 4869 ["rtrie"] = 8885,
 4870 ["trianglerighteq"] = 8885,
 4871 ["origof"] = 8886,
 4872 ["imof"] = 8887,
 4873 ["multimap"] = 8888,
 4874 ["mumap"] = 8888,
 4875 ["hercon"] = 8889,
 4876 ["intcal"] = 8890,
 4877 ["intercal"] = 8890,
 4878 ["veebar"] = 8891,
 4879 ["barvee"] = 8893,
 4880 ["angrtvb"] = 8894,
 4881 ["ltri"] = 8895,
 4882 ["Wedge"] = 8896,
 4883 ["bigwedge"] = 8896,
 4884 ["xwedge"] = 8896,
 4885 ["Vee"] = 8897,
 4886 ["bigvee"] = 8897,
 4887 ["xvee"] = 8897,
 4888 ["Intersection"] = 8898,
 4889 ["bigcap"] = 8898,
 4890 ["xcap"] = 8898,
 4891 ["Union"] = 8899,
 4892 ["bigcup"] = 8899,
 4893 ["xcup"] = 8899,
 4894 ["Diamond"] = 8900,
 4895 ["diam"] = 8900,
 4896 ["diamond"] = 8900,
 4897 ["sdot"] = 8901,

4898 ["Star"] = 8902,
 4899 ["sstarf"] = 8902,
 4900 ["divideontimes"] = 8903,
 4901 ["divonx"] = 8903,
 4902 ["bowtie"] = 8904,
 4903 ["ltimes"] = 8905,
 4904 ["rtimes"] = 8906,
 4905 ["leftthreetimes"] = 8907,
 4906 ["lthree"] = 8907,
 4907 ["rightthreetimes"] = 8908,
 4908 ["rthree"] = 8908,
 4909 ["backsimeq"] = 8909,
 4910 ["bsime"] = 8909,
 4911 ["curlyvee"] = 8910,
 4912 ["cuvee"] = 8910,
 4913 ["curlywedge"] = 8911,
 4914 ["cuwed"] = 8911,
 4915 ["Sub"] = 8912,
 4916 ["Subset"] = 8912,
 4917 ["Sup"] = 8913,
 4918 ["Supset"] = 8913,
 4919 ["Cap"] = 8914,
 4920 ["Cup"] = 8915,
 4921 ["fork"] = 8916,
 4922 ["pitchfork"] = 8916,
 4923 ["epar"] = 8917,
 4924 ["lessdot"] = 8918,
 4925 ["ltdot"] = 8918,
 4926 ["gtdot"] = 8919,
 4927 ["gtrdot"] = 8919,
 4928 ["Ll"] = 8920,
 4929 ["nLl"] = {8920, 824},
 4930 ["Gg"] = 8921,
 4931 ["ggg"] = 8921,
 4932 ["nGg"] = {8921, 824},
 4933 ["LessEqualGreater"] = 8922,
 4934 ["leg"] = 8922,
 4935 ["lesg"] = {8922, 65024},
 4936 ["lesseqgtr"] = 8922,
 4937 ["GreaterEqualLess"] = 8923,
 4938 ["gel"] = 8923,
 4939 ["gesl"] = {8923, 65024},
 4940 ["gtreqless"] = 8923,
 4941 ["cuepr"] = 8926,
 4942 ["curlyeqprec"] = 8926,
 4943 ["cuesc"] = 8927,
 4944 ["curlyeqsucc"] = 8927,

4945 ["NotPrecedesSlantEqual"] = 8928,
 4946 ["nprcue"] = 8928,
 4947 ["NotSucceedsSlantEqual"] = 8929,
 4948 ["nsccue"] = 8929,
 4949 ["NotSquareSubsetEqual"] = 8930,
 4950 ["nsqsube"] = 8930,
 4951 ["NotSquareSupersetEqual"] = 8931,
 4952 ["nsqsupe"] = 8931,
 4953 ["lnsim"] = 8934,
 4954 ["gnsim"] = 8935,
 4955 ["precnsim"] = 8936,
 4956 ["prnsim"] = 8936,
 4957 ["scnsim"] = 8937,
 4958 ["succnsim"] = 8937,
 4959 ["NotLeftTriangle"] = 8938,
 4960 ["nltri"] = 8938,
 4961 ["ntriangleleft"] = 8938,
 4962 ["NotRightTriangle"] = 8939,
 4963 ["nrtri"] = 8939,
 4964 ["ntriangleright"] = 8939,
 4965 ["NotLeftTriangleEqual"] = 8940,
 4966 ["nltrie"] = 8940,
 4967 ["ntrianglelefteq"] = 8940,
 4968 ["NotRightTriangleEqual"] = 8941,
 4969 ["nrtrie"] = 8941,
 4970 ["ntrianglerighteq"] = 8941,
 4971 ["vellip"] = 8942,
 4972 ["ctdot"] = 8943,
 4973 ["utdot"] = 8944,
 4974 ["dtdot"] = 8945,
 4975 ["disin"] = 8946,
 4976 ["isinsv"] = 8947,
 4977 ["isins"] = 8948,
 4978 ["isindot"] = 8949,
 4979 ["notinodot"] = {8949, 824},
 4980 ["notinvc"] = 8950,
 4981 ["notinvb"] = 8951,
 4982 ["isinE"] = 8953,
 4983 ["notinE"] = {8953, 824},
 4984 ["nisd"] = 8954,
 4985 ["xnis"] = 8955,
 4986 ["nis"] = 8956,
 4987 ["notnivc"] = 8957,
 4988 ["notnivb"] = 8958,
 4989 ["barwed"] = 8965,
 4990 ["barwedge"] = 8965,
 4991 ["Barwed"] = 8966,

4992 ["doublebarwedge"] = 8966,
4993 ["LeftCeiling"] = 8968,
4994 ["lceil"] = 8968,
4995 ["RightCeiling"] = 8969,
4996 ["rceil"] = 8969,
4997 ["LeftFloor"] = 8970,
4998 ["lfloor"] = 8970,
4999 ["RightFloor"] = 8971,
5000 ["rfloor"] = 8971,
5001 ["drcrop"] = 8972,
5002 ["dlcrop"] = 8973,
5003 ["urcrop"] = 8974,
5004 ["ulcrop"] = 8975,
5005 ["bnot"] = 8976,
5006 ["proflines"] = 8978,
5007 ["profsurf"] = 8979,
5008 ["telrec"] = 8981,
5009 ["target"] = 8982,
5010 ["ulcorn"] = 8988,
5011 ["ulcorner"] = 8988,
5012 ["urcorn"] = 8989,
5013 ["urcorner"] = 8989,
5014 ["dlcorn"] = 8990,
5015 ["llcorner"] = 8990,
5016 ["drcorn"] = 8991,
5017 ["lrcorn"] = 8991,
5018 ["frown"] = 8994,
5019 ["sfrown"] = 8994,
5020 ["smile"] = 8995,
5021 ["ssmile"] = 8995,
5022 ["cylcty"] = 9005,
5023 ["profalar"] = 9006,
5024 ["topbot"] = 9014,
5025 ["ovbar"] = 9021,
5026 ["solbar"] = 9023,
5027 ["angzarr"] = 9084,
5028 ["lmoust"] = 9136,
5029 ["lmoustache"] = 9136,
5030 ["rmoust"] = 9137,
5031 ["rmoustache"] = 9137,
5032 ["OverBracket"] = 9140,
5033 ["tbrk"] = 9140,
5034 ["UnderBracket"] = 9141,
5035 ["bbrk"] = 9141,
5036 ["bbrktbrk"] = 9142,
5037 ["OverParenthesis"] = 9180,
5038 ["UnderParenthesis"] = 9181,

5039 ["OverBrace"] = 9182,
5040 ["UnderBrace"] = 9183,
5041 ["trpezium"] = 9186,
5042 ["elinters"] = 9191,
5043 ["blank"] = 9251,
5044 ["circledS"] = 9416,
5045 ["oS"] = 9416,
5046 ["HorizontalLine"] = 9472,
5047 ["boxh"] = 9472,
5048 ["boxv"] = 9474,
5049 ["boxdr"] = 9484,
5050 ["boxdl"] = 9488,
5051 ["boxur"] = 9492,
5052 ["boxul"] = 9496,
5053 ["boxvr"] = 9500,
5054 ["boxvl"] = 9508,
5055 ["boxhd"] = 9516,
5056 ["boxhu"] = 9524,
5057 ["boxvh"] = 9532,
5058 ["boxH"] = 9552,
5059 ["boxV"] = 9553,
5060 ["boxdR"] = 9554,
5061 ["boxDr"] = 9555,
5062 ["boxDR"] = 9556,
5063 ["boxdL"] = 9557,
5064 ["boxDL"] = 9558,
5065 ["boxDL"] = 9559,
5066 ["boxuR"] = 9560,
5067 ["boxUr"] = 9561,
5068 ["boxUR"] = 9562,
5069 ["boxuL"] = 9563,
5070 ["boxUL"] = 9564,
5071 ["boxUL"] = 9565,
5072 ["boxvR"] = 9566,
5073 ["boxVr"] = 9567,
5074 ["boxVR"] = 9568,
5075 ["boxvL"] = 9569,
5076 ["boxVl"] = 9570,
5077 ["boxVL"] = 9571,
5078 ["boxHd"] = 9572,
5079 ["boxhD"] = 9573,
5080 ["boxHD"] = 9574,
5081 ["boxHu"] = 9575,
5082 ["boxhU"] = 9576,
5083 ["boxHU"] = 9577,
5084 ["boxvH"] = 9578,
5085 ["boxVh"] = 9579,

5086 ["boxVH"] = 9580,
5087 ["uhblk"] = 9600,
5088 ["lhblk"] = 9604,
5089 ["block"] = 9608,
5090 ["blk14"] = 9617,
5091 ["blk12"] = 9618,
5092 ["blk34"] = 9619,
5093 ["Square"] = 9633,
5094 ["squ"] = 9633,
5095 ["square"] = 9633,
5096 ["FilledVerySmallSquare"] = 9642,
5097 ["blacksquare"] = 9642,
5098 ["squares"] = 9642,
5099 ["sqf"] = 9642,
5100 ["EmptyVerySmallSquare"] = 9643,
5101 ["rect"] = 9645,
5102 ["marker"] = 9646,
5103 ["fltns"] = 9649,
5104 ["bigtriangleup"] = 9651,
5105 ["xutri"] = 9651,
5106 ["blacktriangle"] = 9652,
5107 ["utrif"] = 9652,
5108 ["triangle"] = 9653,
5109 ["utri"] = 9653,
5110 ["blacktriangleright"] = 9656,
5111 ["rtrif"] = 9656,
5112 ["rtri"] = 9657,
5113 ["triangleright"] = 9657,
5114 ["bigtriangledown"] = 9661,
5115 ["xdtri"] = 9661,
5116 ["blacktriangledown"] = 9662,
5117 ["dtrif"] = 9662,
5118 ["dtri"] = 9663,
5119 ["triangledown"] = 9663,
5120 ["blacktriangleleft"] = 9666,
5121 ["ltrif"] = 9666,
5122 ["ltri"] = 9667,
5123 ["triangleleft"] = 9667,
5124 ["loz"] = 9674,
5125 ["lozenge"] = 9674,
5126 ["cir"] = 9675,
5127 ["tridot"] = 9708,
5128 ["bigcirc"] = 9711,
5129 ["xcirc"] = 9711,
5130 ["ultri"] = 9720,
5131 ["urtri"] = 9721,
5132 ["lltri"] = 9722,

5133 ["EmptySmallSquare"] = 9723,
5134 ["FilledSmallSquare"] = 9724,
5135 ["bigstar"] = 9733,
5136 ["starf"] = 9733,
5137 ["star"] = 9734,
5138 ["phone"] = 9742,
5139 ["female"] = 9792,
5140 ["male"] = 9794,
5141 ["spades"] = 9824,
5142 ["spadesuit"] = 9824,
5143 ["clubs"] = 9827,
5144 ["clubsuit"] = 9827,
5145 ["hearts"] = 9829,
5146 ["heartsuit"] = 9829,
5147 ["diamondsuit"] = 9830,
5148 ["diams"] = 9830,
5149 ["sung"] = 9834,
5150 ["flat"] = 9837,
5151 ["natur"] = 9838,
5152 ["natural"] = 9838,
5153 ["sharp"] = 9839,
5154 ["check"] = 10003,
5155 ["checkmark"] = 10003,
5156 ["cross"] = 10007,
5157 ["malt"] = 10016,
5158 ["maltese"] = 10016,
5159 ["sext"] = 10038,
5160 ["VerticalSeparator"] = 10072,
5161 ["lbrk"] = 10098,
5162 ["rbrk"] = 10099,
5163 ["bsolhsub"] = 10184,
5164 ["suphsol"] = 10185,
5165 ["LeftDoubleBracket"] = 10214,
5166 ["lobrk"] = 10214,
5167 ["RightDoubleBracket"] = 10215,
5168 ["robrk"] = 10215,
5169 ["LeftAngleBracket"] = 10216,
5170 ["lang"] = 10216,
5171 ["langle"] = 10216,
5172 ["RightAngleBracket"] = 10217,
5173 ["rang"] = 10217,
5174 ["rangle"] = 10217,
5175 ["Lang"] = 10218,
5176 ["Rang"] = 10219,
5177 ["loang"] = 10220,
5178 ["roang"] = 10221,
5179 ["LongLeftArrow"] = 10229,

5180 ["longleftarrow"] = 10229,
5181 ["xlarr"] = 10229,
5182 ["LongRightArrow"] = 10230,
5183 ["longrightarrow"] = 10230,
5184 ["xrarr"] = 10230,
5185 ["LongLeftRightArrow"] = 10231,
5186 ["longlefttrightarrow"] = 10231,
5187 ["xharr"] = 10231,
5188 ["DoubleLongLeftArrow"] = 10232,
5189 ["Longleftarrow"] = 10232,
5190 ["xlArr"] = 10232,
5191 ["DoubleLongRightArrow"] = 10233,
5192 ["Longrightarrow"] = 10233,
5193 ["xrArr"] = 10233,
5194 ["DoubleLongLeftRightArrow"] = 10234,
5195 ["Longlefttrightarrow"] = 10234,
5196 ["xhArr"] = 10234,
5197 ["longmapsto"] = 10236,
5198 ["xmap"] = 10236,
5199 ["dzigrarr"] = 10239,
5200 ["nvlArr"] = 10498,
5201 ["nvrArr"] = 10499,
5202 ["nvHarr"] = 10500,
5203 ["Map"] = 10501,
5204 ["lbarr"] = 10508,
5205 ["bkarow"] = 10509,
5206 ["rbarr"] = 10509,
5207 ["lBarr"] = 10510,
5208 ["dbkarow"] = 10511,
5209 ["rBarr"] = 10511,
5210 ["RBarr"] = 10512,
5211 ["drbkarow"] = 10512,
5212 ["DDottrahd"] = 10513,
5213 ["UpArrowBar"] = 10514,
5214 ["DownArrowBar"] = 10515,
5215 ["Rarrtl"] = 10518,
5216 ["latail"] = 10521,
5217 ["ratail"] = 10522,
5218 ["lAtail"] = 10523,
5219 ["rAtail"] = 10524,
5220 ["larrfs"] = 10525,
5221 ["rarrfs"] = 10526,
5222 ["larrbfs"] = 10527,
5223 ["rarrbfs"] = 10528,
5224 ["nwarhk"] = 10531,
5225 ["nearhk"] = 10532,
5226 ["hksearrow"] = 10533,

5227 ["searhk"] = 10533,
5228 ["hkswarow"] = 10534,
5229 ["swarhk"] = 10534,
5230 ["nwnear"] = 10535,
5231 ["nesear"] = 10536,
5232 ["toea"] = 10536,
5233 ["seswar"] = 10537,
5234 ["tosa"] = 10537,
5235 ["swnwar"] = 10538,
5236 ["nrarrc"] = {10547, 824},
5237 ["rarrc"] = 10547,
5238 ["cudarr"] = 10549,
5239 ["ldca"] = 10550,
5240 ["rdca"] = 10551,
5241 ["cudarrr"] = 10552,
5242 ["larrpl"] = 10553,
5243 ["curarrm"] = 10556,
5244 ["cularrp"] = 10557,
5245 ["rarrpl"] = 10565,
5246 ["harrcir"] = 10568,
5247 ["Uarrocir"] = 10569,
5248 ["lurdshar"] = 10570,
5249 ["ldrushar"] = 10571,
5250 ["LeftRightVector"] = 10574,
5251 ["RightUpDownVector"] = 10575,
5252 ["DownLeftRightVector"] = 10576,
5253 ["LeftUpDownVector"] = 10577,
5254 ["LeftVectorBar"] = 10578,
5255 ["RightVectorBar"] = 10579,
5256 ["RightUpVectorBar"] = 10580,
5257 ["RightDownVectorBar"] = 10581,
5258 ["DownLeftVectorBar"] = 10582,
5259 ["DownRightVectorBar"] = 10583,
5260 ["LeftUpVectorBar"] = 10584,
5261 ["LeftDownVectorBar"] = 10585,
5262 ["LeftTeeVector"] = 10586,
5263 ["RightTeeVector"] = 10587,
5264 ["RightUpTeeVector"] = 10588,
5265 ["RightDownTeeVector"] = 10589,
5266 ["DownLeftTeeVector"] = 10590,
5267 ["DownRightTeeVector"] = 10591,
5268 ["LeftUpTeeVector"] = 10592,
5269 ["LeftDownTeeVector"] = 10593,
5270 ["lHar"] = 10594,
5271 ["uHar"] = 10595,
5272 ["rHar"] = 10596,
5273 ["dHar"] = 10597,

5274 ["luruhar"] = 10598,
5275 ["ldrdhar"] = 10599,
5276 ["ruluhar"] = 10600,
5277 ["rdldhar"] = 10601,
5278 ["lharul"] = 10602,
5279 ["llhard"] = 10603,
5280 ["rharul"] = 10604,
5281 ["lrhard"] = 10605,
5282 ["UpEquilibrium"] = 10606,
5283 ["udhar"] = 10606,
5284 ["ReverseUpEquilibrium"] = 10607,
5285 ["duhar"] = 10607,
5286 ["RoundImplies"] = 10608,
5287 ["erarr"] = 10609,
5288 ["simrarr"] = 10610,
5289 ["larrsim"] = 10611,
5290 ["rarrsim"] = 10612,
5291 ["rarrap"] = 10613,
5292 ["ltlarr"] = 10614,
5293 ["gtrarr"] = 10616,
5294 ["subrarr"] = 10617,
5295 ["suplarr"] = 10619,
5296 ["lfishr"] = 10620,
5297 ["rfishr"] = 10621,
5298 ["ufishr"] = 10622,
5299 ["dfishr"] = 10623,
5300 ["lopar"] = 10629,
5301 ["ropar"] = 10630,
5302 ["lbrke"] = 10635,
5303 ["rbrke"] = 10636,
5304 ["lbrkslu"] = 10637,
5305 ["rbrksld"] = 10638,
5306 ["lbrksld"] = 10639,
5307 ["rbrkslu"] = 10640,
5308 ["langd"] = 10641,
5309 ["rangd"] = 10642,
5310 ["lparlt"] = 10643,
5311 ["rpargt"] = 10644,
5312 ["gtlPar"] = 10645,
5313 ["ltrPar"] = 10646,
5314 ["vzigzag"] = 10650,
5315 ["vangrt"] = 10652,
5316 ["angrtvbd"] = 10653,
5317 ["ange"] = 10660,
5318 ["range"] = 10661,
5319 ["dwangle"] = 10662,
5320 ["uwangle"] = 10663,

5321 ["angmsdaa"] = 10664,
5322 ["angmsdab"] = 10665,
5323 ["angmsdac"] = 10666,
5324 ["angmsdad"] = 10667,
5325 ["angmsdae"] = 10668,
5326 ["angmsdaf"] = 10669,
5327 ["angmsdag"] = 10670,
5328 ["angmsdah"] = 10671,
5329 ["bemptyv"] = 10672,
5330 ["demptyv"] = 10673,
5331 ["cemptyv"] = 10674,
5332 ["raemptyv"] = 10675,
5333 ["laemptyv"] = 10676,
5334 ["ohbar"] = 10677,
5335 ["omid"] = 10678,
5336 ["opar"] = 10679,
5337 ["operp"] = 10681,
5338 ["olcross"] = 10683,
5339 ["odsold"] = 10684,
5340 ["olcir"] = 10686,
5341 ["ofcir"] = 10687,
5342 ["olt"] = 10688,
5343 ["ogt"] = 10689,
5344 ["cirscir"] = 10690,
5345 ["cirE"] = 10691,
5346 ["solb"] = 10692,
5347 ["bsolb"] = 10693,
5348 ["boxbox"] = 10697,
5349 ["trisb"] = 10701,
5350 ["rtriltri"] = 10702,
5351 ["LeftTriangleBar"] = 10703,
5352 ["NotLeftTriangleBar"] = {10703, 824},
5353 ["NotRightTriangleBar"] = {10704, 824},
5354 ["RightTriangleBar"] = 10704,
5355 ["iinfin"] = 10716,
5356 ["infintie"] = 10717,
5357 ["nvinfin"] = 10718,
5358 ["eparsl"] = 10723,
5359 ["smeparsl"] = 10724,
5360 ["eqvparsl"] = 10725,
5361 ["blacklozenge"] = 10731,
5362 ["lozf"] = 10731,
5363 ["RuleDelayed"] = 10740,
5364 ["dsol"] = 10742,
5365 ["bigodot"] = 10752,
5366 ["xodot"] = 10752,
5367 ["bigoplus"] = 10753,

5368 ["xoplus"] = 10753,
5369 ["bigotimes"] = 10754,
5370 ["xotime"] = 10754,
5371 ["biguplus"] = 10756,
5372 ["xuplus"] = 10756,
5373 ["bigsqcup"] = 10758,
5374 ["xsqlcup"] = 10758,
5375 ["iiiint"] = 10764,
5376 ["qint"] = 10764,
5377 ["fpartint"] = 10765,
5378 ["cirfnint"] = 10768,
5379 ["awint"] = 10769,
5380 ["rppolint"] = 10770,
5381 ["scpolint"] = 10771,
5382 ["npolint"] = 10772,
5383 ["pointint"] = 10773,
5384 ["quatint"] = 10774,
5385 ["intlarhk"] = 10775,
5386 ["pluscir"] = 10786,
5387 ["plusacir"] = 10787,
5388 ["simplus"] = 10788,
5389 ["plusdu"] = 10789,
5390 ["plussim"] = 10790,
5391 ["plustwo"] = 10791,
5392 ["mcomma"] = 10793,
5393 ["minusdu"] = 10794,
5394 ["loplus"] = 10797,
5395 ["roplus"] = 10798,
5396 ["Cross"] = 10799,
5397 ["timesd"] = 10800,
5398 ["timesbar"] = 10801,
5399 ["smashp"] = 10803,
5400 ["lotimes"] = 10804,
5401 ["rotimes"] = 10805,
5402 ["otimesas"] = 10806,
5403 ["Otimes"] = 10807,
5404 ["odiv"] = 10808,
5405 ["triplus"] = 10809,
5406 ["triminus"] = 10810,
5407 ["tritime"] = 10811,
5408 ["intprod"] = 10812,
5409 ["iproduct"] = 10812,
5410 ["amalg"] = 10815,
5411 ["capdot"] = 10816,
5412 ["ncup"] = 10818,
5413 ["ncap"] = 10819,
5414 ["capand"] = 10820,

5415 ["cupor"] = 10821,
5416 ["cupcap"] = 10822,
5417 ["capcup"] = 10823,
5418 ["cupbrcap"] = 10824,
5419 ["capbrcup"] = 10825,
5420 ["cupcup"] = 10826,
5421 ["capcap"] = 10827,
5422 ["ccups"] = 10828,
5423 ["ccaps"] = 10829,
5424 ["ccupssm"] = 10832,
5425 ["And"] = 10835,
5426 ["Or"] = 10836,
5427 ["andand"] = 10837,
5428 ["oror"] = 10838,
5429 ["orslope"] = 10839,
5430 ["andslope"] = 10840,
5431 ["andv"] = 10842,
5432 ["orv"] = 10843,
5433 ["andd"] = 10844,
5434 ["ord"] = 10845,
5435 ["wedbar"] = 10847,
5436 ["sdote"] = 10854,
5437 ["simdot"] = 10858,
5438 ["congdote"] = 10861,
5439 ["ncongdote"] = {10861, 824},
5440 ["easter"] = 10862,
5441 ["apacir"] = 10863,
5442 ["apE"] = 10864,
5443 ["napE"] = {10864, 824},
5444 ["eplus"] = 10865,
5445 ["pluse"] = 10866,
5446 ["Esim"] = 10867,
5447 ["Colone"] = 10868,
5448 ["Equal"] = 10869,
5449 ["ddotseq"] = 10871,
5450 ["eDDot"] = 10871,
5451 ["equivDD"] = 10872,
5452 ["ltcir"] = 10873,
5453 ["gtcir"] = 10874,
5454 ["ltquest"] = 10875,
5455 ["gtquest"] = 10876,
5456 ["LessSlantEqual"] = 10877,
5457 ["NotLessSlantEqual"] = {10877, 824},
5458 ["leqslant"] = 10877,
5459 ["les"] = 10877,
5460 ["nleqslant"] = {10877, 824},
5461 ["nles"] = {10877, 824},

5462 ["GreaterSlantEqual"] = 10878,
5463 ["NotGreaterSlantEqual"] = {10878, 824},
5464 ["geqslant"] = 10878,
5465 ["ges"] = 10878,
5466 ["ngeqslant"] = {10878, 824},
5467 ["nges"] = {10878, 824},
5468 ["lesdot"] = 10879,
5469 ["gesdot"] = 10880,
5470 ["lesdoto"] = 10881,
5471 ["gesdoto"] = 10882,
5472 ["lesdotor"] = 10883,
5473 ["gesdoto1"] = 10884,
5474 ["lap"] = 10885,
5475 ["lessapprox"] = 10885,
5476 ["gap"] = 10886,
5477 ["gtrapprox"] = 10886,
5478 ["lne"] = 10887,
5479 ["lneq"] = 10887,
5480 ["gne"] = 10888,
5481 ["gneq"] = 10888,
5482 ["lnap"] = 10889,
5483 ["lnapprox"] = 10889,
5484 ["gnap"] = 10890,
5485 ["gnapprox"] = 10890,
5486 ["lEg"] = 10891,
5487 ["lesseqgtr"] = 10891,
5488 ["gEl"] = 10892,
5489 ["gtreqless"] = 10892,
5490 ["lsime"] = 10893,
5491 ["gsime"] = 10894,
5492 ["lsimg"] = 10895,
5493 ["gsiml"] = 10896,
5494 ["lgE"] = 10897,
5495 ["glE"] = 10898,
5496 ["lesges"] = 10899,
5497 ["gesles"] = 10900,
5498 ["els"] = 10901,
5499 ["eqslantless"] = 10901,
5500 ["egs"] = 10902,
5501 ["eqslantgtr"] = 10902,
5502 ["elsdot"] = 10903,
5503 ["egsdot"] = 10904,
5504 ["el"] = 10905,
5505 ["eg"] = 10906,
5506 ["siml"] = 10909,
5507 ["simg"] = 10910,
5508 ["simlE"] = 10911,


```

5509 ["simgE"] = 10912,
5510 ["LessLess"] = 10913,
5511 ["NotNestedLessLess"] = {10913, 824},
5512 ["GreaterGreater"] = 10914,
5513 ["NotNestedGreaterGreater"] = {10914, 824},
5514 ["glj"] = 10916,
5515 ["gla"] = 10917,
5516 ["ltcc"] = 10918,
5517 ["gtcc"] = 10919,
5518 ["lescc"] = 10920,
5519 ["gescc"] = 10921,
5520 ["smt"] = 10922,
5521 ["lat"] = 10923,
5522 ["smte"] = 10924,
5523 ["smtes"] = {10924, 65024},
5524 ["late"] = 10925,
5525 ["lates"] = {10925, 65024},
5526 ["bumpE"] = 10926,
5527 ["NotPrecedesEqual"] = {10927, 824},
5528 ["PrecedesEqual"] = 10927,
5529 ["npre"] = {10927, 824},
5530 ["npreceq"] = {10927, 824},
5531 ["pre"] = 10927,
5532 ["preceq"] = 10927,
5533 ["NotSucceedsEqual"] = {10928, 824},
5534 ["SucceedsEqual"] = 10928,
5535 ["nsce"] = {10928, 824},
5536 ["nsucceq"] = {10928, 824},
5537 ["sce"] = 10928,
5538 ["succeq"] = 10928,
5539 ["prE"] = 10931,
5540 ["scE"] = 10932,
5541 ["precneqq"] = 10933,
5542 ["prnE"] = 10933,
5543 ["scnE"] = 10934,
5544 ["succneqq"] = 10934,
5545 ["prap"] = 10935,
5546 ["precapprox"] = 10935,
5547 ["scap"] = 10936,
5548 ["succapprox"] = 10936,
5549 ["precnapprox"] = 10937,
5550 ["prnap"] = 10937,
5551 ["scnap"] = 10938,
5552 ["succnapprox"] = 10938,
5553 ["Pr"] = 10939,
5554 ["Sc"] = 10940,
5555 ["subdot"] = 10941,

```

5556 ["supdot"] = 10942,
5557 ["subplus"] = 10943,
5558 ["supplus"] = 10944,
5559 ["submult"] = 10945,
5560 ["supmult"] = 10946,
5561 ["subedot"] = 10947,
5562 ["supedot"] = 10948,
5563 ["nsubE"] = {10949, 824},
5564 ["nsubseteqq"] = {10949, 824},
5565 ["subE"] = 10949,
5566 ["subseteqq"] = 10949,
5567 ["nsupE"] = {10950, 824},
5568 ["nsupseteqq"] = {10950, 824},
5569 ["supE"] = 10950,
5570 ["supseteqq"] = 10950,
5571 ["subsim"] = 10951,
5572 ["supsim"] = 10952,
5573 ["subnE"] = 10955,
5574 ["subsetneqq"] = 10955,
5575 ["varsubsetneqq"] = {10955, 65024},
5576 ["vsubnE"] = {10955, 65024},
5577 ["supnE"] = 10956,
5578 ["supsetneqq"] = 10956,
5579 ["varsupsetneqq"] = {10956, 65024},
5580 ["vsupnE"] = {10956, 65024},
5581 ["csub"] = 10959,
5582 ["csup"] = 10960,
5583 ["csube"] = 10961,
5584 ["csupe"] = 10962,
5585 ["subsup"] = 10963,
5586 ["supsub"] = 10964,
5587 ["subsub"] = 10965,
5588 ["supsup"] = 10966,
5589 ["suphsub"] = 10967,
5590 ["supdsub"] = 10968,
5591 ["forkv"] = 10969,
5592 ["topfork"] = 10970,
5593 ["mlcp"] = 10971,
5594 ["Dashv"] = 10980,
5595 ["DoubleLeftTee"] = 10980,
5596 ["Vdashl"] = 10982,
5597 ["Barv"] = 10983,
5598 ["vBar"] = 10984,
5599 ["vBarv"] = 10985,
5600 ["Vbar"] = 10987,
5601 ["Not"] = 10988,
5602 ["bNot"] = 10989,

5603 ["rnmid"] = 10990,
5604 ["cirmid"] = 10991,
5605 ["midcir"] = 10992,
5606 ["topcir"] = 10993,
5607 ["nhpar"] = 10994,
5608 ["parsim"] = 10995,
5609 ["nparsl"] = {11005, 8421},
5610 ["parsl"] = 11005,
5611 ["fflig"] = 64256,
5612 ["filig"] = 64257,
5613 ["fllig"] = 64258,
5614 ["ffilig"] = 64259,
5615 ["ffllig"] = 64260,
5616 ["Ascr"] = 119964,
5617 ["Cscr"] = 119966,
5618 ["Dscr"] = 119967,
5619 ["Gscr"] = 119970,
5620 ["Jscr"] = 119973,
5621 ["Kscr"] = 119974,
5622 ["Nscr"] = 119977,
5623 ["Oscr"] = 119978,
5624 ["Pscr"] = 119979,
5625 ["Qscr"] = 119980,
5626 ["Sscr"] = 119982,
5627 ["Tscr"] = 119983,
5628 ["Uscr"] = 119984,
5629 ["Vscr"] = 119985,
5630 ["Wscr"] = 119986,
5631 ["Xscr"] = 119987,
5632 ["Yscr"] = 119988,
5633 ["Zscr"] = 119989,
5634 ["ascr"] = 119990,
5635 ["bscr"] = 119991,
5636 ["cscr"] = 119992,
5637 ["dscr"] = 119993,
5638 ["fscr"] = 119995,
5639 ["hscr"] = 119997,
5640 ["iscr"] = 119998,
5641 ["jscr"] = 119999,
5642 ["kscr"] = 120000,
5643 ["lscr"] = 120001,
5644 ["mscr"] = 120002,
5645 ["nscr"] = 120003,
5646 ["pscr"] = 120005,
5647 ["qscr"] = 120006,
5648 ["rscr"] = 120007,
5649 ["sscr"] = 120008,

5650 ["tscr"] = 120009,
5651 ["uscr"] = 120010,
5652 ["vscr"] = 120011,
5653 ["wscr"] = 120012,
5654 ["xscr"] = 120013,
5655 ["yscr"] = 120014,
5656 ["zscr"] = 120015,
5657 ["Afr"] = 120068,
5658 ["Bfr"] = 120069,
5659 ["Dfr"] = 120071,
5660 ["Efr"] = 120072,
5661 ["Ffr"] = 120073,
5662 ["Gfr"] = 120074,
5663 ["Jfr"] = 120077,
5664 ["Kfr"] = 120078,
5665 ["Lfr"] = 120079,
5666 ["Mfr"] = 120080,
5667 ["Nfr"] = 120081,
5668 ["Ofr"] = 120082,
5669 ["Pfr"] = 120083,
5670 ["Qfr"] = 120084,
5671 ["Sfr"] = 120086,
5672 ["Tfr"] = 120087,
5673 ["Ufr"] = 120088,
5674 ["Vfr"] = 120089,
5675 ["Wfr"] = 120090,
5676 ["Xfr"] = 120091,
5677 ["Yfr"] = 120092,
5678 ["afr"] = 120094,
5679 ["bfr"] = 120095,
5680 ["cfr"] = 120096,
5681 ["dfr"] = 120097,
5682 ["efr"] = 120098,
5683 ["ffr"] = 120099,
5684 ["gfr"] = 120100,
5685 ["hfr"] = 120101,
5686 ["ifr"] = 120102,
5687 ["jfr"] = 120103,
5688 ["kfr"] = 120104,
5689 ["lfr"] = 120105,
5690 ["mfr"] = 120106,
5691 ["nfr"] = 120107,
5692 ["ofr"] = 120108,
5693 ["pfr"] = 120109,
5694 ["qfr"] = 120110,
5695 ["rfr"] = 120111,
5696 ["sfr"] = 120112,

5697 ["tfr"] = 120113,
5698 ["ufr"] = 120114,
5699 ["vfr"] = 120115,
5700 ["wfr"] = 120116,
5701 ["xfr"] = 120117,
5702 ["yfr"] = 120118,
5703 ["zfr"] = 120119,
5704 ["Aopf"] = 120120,
5705 ["Bopf"] = 120121,
5706 ["Dopf"] = 120123,
5707 ["Eopf"] = 120124,
5708 ["Fopf"] = 120125,
5709 ["Gopf"] = 120126,
5710 ["Iopf"] = 120128,
5711 ["Jopf"] = 120129,
5712 ["Kopf"] = 120130,
5713 ["Lopf"] = 120131,
5714 ["Mopf"] = 120132,
5715 ["Oopf"] = 120134,
5716 ["Sopf"] = 120138,
5717 ["Topf"] = 120139,
5718 ["Uopf"] = 120140,
5719 ["Vopf"] = 120141,
5720 ["Wopf"] = 120142,
5721 ["Xopf"] = 120143,
5722 ["Yopf"] = 120144,
5723 ["aopf"] = 120146,
5724 ["bopf"] = 120147,
5725 ["copf"] = 120148,
5726 ["dopf"] = 120149,
5727 ["eopf"] = 120150,
5728 ["fopf"] = 120151,
5729 ["gopf"] = 120152,
5730 ["hopf"] = 120153,
5731 ["iopf"] = 120154,
5732 ["jopf"] = 120155,
5733 ["kopf"] = 120156,
5734 ["lopf"] = 120157,
5735 ["mopf"] = 120158,
5736 ["nopf"] = 120159,
5737 ["oopf"] = 120160,
5738 ["popf"] = 120161,
5739 ["qopf"] = 120162,
5740 ["ropf"] = 120163,
5741 ["sopf"] = 120164,
5742 ["topf"] = 120165,
5743 ["uopf"] = 120166,

```

5744 ["vopf"] = 120167,
5745 ["wopf"] = 120168,
5746 ["xopf"] = 120169,
5747 ["yopf"] = 120170,
5748 ["zopf"] = 120171,
5749 }

```

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

```

5750 function entities.dec_entity(s)
5751   local n = tonumber(s)
5752   if n == nil then
5753     return "&#" .. s .. ";" -- fallback for unknown entities
5754   end
5755   return unicode.utf8.char(n)
5756 end

```

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

```

5757 function entities.hex_entity(s)
5758   local n = tonumber("0x"..s)
5759   if n == nil then
5760     return "&#x" .. s .. ";" -- fallback for unknown entities
5761   end
5762   return unicode.utf8.char(n)
5763 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.

```

5764 function entities.hex_entity_with_x_char(x, s)
5765   local n = tonumber("0x"..s)
5766   if n == nil then
5767     return "&#" .. x .. s .. ";" -- fallback for unknown entities
5768   end
5769   return unicode.utf8.char(n)
5770 end

```

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

```

5771 function entities.char_entity(s)
5772   local code_points = character_entities[s]
5773   if code_points == nil then
5774     return "&" .. s .. ";"
5775   end
5776   if type(code_points) ~= 'table' then
5777     code_points = {code_points}
5778   end

```

```

5779 local char_table = {}
5780     for _, code_point in ipairs(code_points) do
5781         table.insert(char_table, unicode.utf8.char(code_point))
5782     end
5783 return table.concat(char_table)
5784 end

```

3.1.3 Plain T_EX Writer

This section documents the `writer` object, which implements the routines for producing the T_EX output. The object is an amalgamate of the generic, T_EX, L^AT_EX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark Lua module.

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

```

5785 M.writer = {}

```

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

```

5786 function M.writer.new(options)
5787     local self = {}

```

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

```

5788     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.

```

5789     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.

```

5790     local slice_specifiers = {}
5791     for specifier in options.slice:gmatch("[^%s]+") do
5792         table.insert(slice_specifiers, specifier)
5793     end
5794

```

```

5795 if #slice_specifiers == 2 then
5796     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5797     local slice_begin_type = self.slice_begin:sub(1, 1)
5798     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5799         self.slice_begin = "^" .. self.slice_begin
5800     end
5801     local slice_end_type = self.slice_end:sub(1, 1)
5802     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5803         self.slice_end = "$" .. self.slice_end
5804     end
5805 elseif #slice_specifiers == 1 then
5806     self.slice_begin = "^" .. slice_specifiers[1]
5807     self.slice_end = "$" .. slice_specifiers[1]
5808 end
5809
5810 self.slice_begin_type = self.slice_begin:sub(1, 1)
5811 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5812 self.slice_end_type = self.slice_end:sub(1, 1)
5813 self.slice_end_identifier = self.slice_end:sub(2) or ""
5814
5815 if self.slice_begin == "^" and self.slice_end ~= "^" then
5816     self.is_writing = true
5817 else
5818     self.is_writing = false
5819 end

```

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

```
5820 self.space = " "
```

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

```
5821 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5822 function self.plain(s)
5823     return s
5824 end

```

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

```
5825 function self.paragraph(s)
5826     if not self.is_writing then return "" end
5827     return s
5828 end

```

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

```
5829 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
5830 function self.interblocksep()
5831     if not self.is_writing then return "" end

```



```

5832     return self.interblocksep_text
5833 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.

```

5834 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n}"
5835 function self.paragraphsep()
5836     if not self.is_writing then return "" end
5837     return self.paragraphsep_text
5838 end

```

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

```

5839 self.undosep_text = "\\markdownRendererUndoSeparator\n}"
5840 function self.undosep()
5841     if not self.is_writing then return "" end
5842     return self.undosep_text
5843 end

```

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

```

5844 self.soft_line_break = function()
5845     if self.flatten_inlines then return "\n" end
5846     return "\\markdownRendererSoftLineBreak\n}"
5847 end

```

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

```

5848 self.hard_line_break = function()
5849     if self.flatten_inlines then return "\n" end
5850     return "\\markdownRendererHardLineBreak\n}"
5851 end

```

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

```

5852 self.ellipsis = "\\markdownRendererEllipsis}"

```

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

```

5853 function self.thematic_break()
5854     if not self.is_writing then return "" end
5855     return "\\markdownRendererThematicBreak}"
5856 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.

```

5857 self.escaped_uri_chars = {
5858     [{""] = "\\markdownRendererLeftBrace}",
5859     ["}"] = "\\markdownRendererRightBrace}",
5860     [{"\\"}] = "\\markdownRendererBackslash}",

```

```

5861     ["\r"] = " ",
5862     ["\n"] = " ",
5863   }
5864   self.escaped_minimal_strings = {
5865     ["^"] = "\\markdownRendererCircumflex",
5866     .. "\\markdownRendererCircumflex ",
5867     ["☒"] = "\\markdownRendererTickedBox{}",
5868     ["☐"] = "\\markdownRendererHalfTickedBox{}",
5869     ["□"] = "\\markdownRendererUntickedBox{}",
5870     [entities.hex_entity('FFFD')]
5871       = "\\markdownRendererReplacementCharacter{}",
5872   }

```

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

```

5873   self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5874   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.

```

5875   self.escaped_chars = {
5876     [{""] = "\\markdownRendererLeftBrace{}",
5877     ["}"] = "\\markdownRendererRightBrace{}",
5878     [%"] = "\\markdownRendererPercentSign{}",
5879     [{"\\"}] = "\\markdownRendererBackslash{}",
5880     [{"#"}] = "\\markdownRendererHash{}",
5881     [{"$"}] = "\\markdownRendererDollarSign{}",
5882     [{"&"}] = "\\markdownRendererAmpersand{}",
5883     [{"_"}] = "\\markdownRendererUnderscore{}",
5884     [{"^"}] = "\\markdownRendererCircumflex{}",
5885     [{"~"}] = "\\markdownRendererTilde{}",
5886     [{"|"}] = "\\markdownRendererPipe{}",
5887     [entities.hex_entity('0000')]
5888       = "\\markdownRendererReplacementCharacter{}",
5889   }

```

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.

```

5890   local function create_escaper(char_escapes, string_escapes)
5891     local escape = util.escaper(char_escapes, string_escapes)
5892     return function(s)
5893       if self.flatten_inlines then return s end
5894       return escape(s)
5895     end
5896   end
5897   local escape_typographic_text = create_escaper(

```

```

5898     self.escaped_chars, self.escaped_strings)
5899     local escape_programmatic_text = create_escaper(
5900         self.escaped_uri_chars, self.escaped_minimal_strings)
5901     local escape_minimal = create_escaper(
5902         {}, 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.

```

5903     self.escape = escape_typographic_text
5904     self.math = escape_minimal
5905     if options.hybrid then
5906         self.identifier = escape_minimal
5907         self.string = escape_minimal
5908         self.uri = escape_minimal
5909         self.infostring = escape_minimal
5910     else
5911         self.identifier = escape_programmatic_text
5912         self.string = escape_typographic_text
5913         self.uri = escape_programmatic_text
5914         self.infostring = escape_programmatic_text
5915     end

```

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

```

5916     function self.warning(t, m)
5917         return {"\markdownRendererWarning{" , self.escape(t), "}{" ,
5918             escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
5919             escape_minimal(m or ""), "}"}
5920     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.

```

5921     function self.error(t, m)
5922         return {"\markdownRendererError{" , self.escape(t), "}{" ,
5923             escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
5924             escape_minimal(m or ""), "}"}
5925     end

```

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

```
5926 function self.code(s, attributes)
5927   if self.flatten_inlines then return s end
5928   local buf = {}
5929   if attributes ~= nil then
5930     table.insert(buf,
5931                 "\\markdownRendererCodeSpanAttributeContextBegin\n")
5932     table.insert(buf, self.attributes(attributes))
5933   end
5934   table.insert(buf,
5935               {"\\markdownRendererCodeSpan{" , self.escape(s), "}"})
5936   if attributes ~= nil then
5937     table.insert(buf,
5938                 "\\markdownRendererCodeSpanAttributeContextEnd{")
5939   end
5940   return buf
5941 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.

```
5942 function self.link(lab, src, tit, attributes)
5943   if self.flatten_inlines then return lab end
5944   local buf = {}
5945   if attributes ~= nil then
5946     table.insert(buf,
5947                 "\\markdownRendererLinkAttributeContextBegin\n")
5948     table.insert(buf, self.attributes(attributes))
5949   end
5950   table.insert(buf, {"\\markdownRendererLink{" , lab, "}" ,
5951                   {" ,self.escape(src), "}" ,
5952                   {" ,self.uri(src), "}" ,
5953                   {" ,self.string(tit or "") , "}"})
5954   if attributes ~= nil then
5955     table.insert(buf,
5956                 "\\markdownRendererLinkAttributeContextEnd{")
5957   end
5958   return buf
5959 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.

```
5960 function self.image(lab, src, tit, attributes)
5961   if self.flatten_inlines then return lab end
5962   local buf = {}
```

```

5963     if attributes ~= nil then
5964         table.insert(buf,
5965             "\\markdownRendererImageAttributeContextBegin\n")
5966         table.insert(buf, self.attributes(attributes))
5967     end
5968     table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
5969         {" ,self.string(src),"} ",
5970         {" ,self.uri(src),"} ",
5971         {" ,self.string(tit or ""),"} })
5972     if attributes ~= nil then
5973         table.insert(buf,
5974             "\\markdownRendererImageAttributeContextEnd{ }")
5975     end
5976     return buf
5977 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.

```

5978     function self.bulletlist(items,tight)
5979         if not self.is_writing then return "" end
5980         local buffer = {}
5981         for _,item in ipairs(items) do
5982             if item ~= "" then
5983                 buffer[#buffer + 1] = self.bulletitem(item)
5984             end
5985         end
5986         local contents = util.intersperse(buffer,"\n")
5987         if tight and options.tightLists then
5988             return {"\\markdownRendererUlBeginTight\n",contents,
5989                 "\n\\markdownRendererUlEndTight "}
5990         else
5991             return {"\\markdownRendererUlBegin\n",contents,
5992                 "\n\\markdownRendererUlEnd "}
5993         end
5994     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.

```

5995     function self.bulletitem(s)
5996         return {"\\markdownRendererUlItem ",s,
5997             "\\markdownRendererUlItemEnd "}
5998     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.

```

5999 function self.orderedlist(items,tight,startnum)
6000     if not self.is_writing then return "" end
6001     local buffer = {}
6002     local num = startnum
6003     for _,item in ipairs(items) do
6004         if item ~= "" then
6005             buffer[#buffer + 1] = self.ordereditem(item,num)
6006         end
6007         if num ~= nil and item ~= "" then
6008             num = num + 1
6009         end
6010     end
6011     local contents = util.intersperse(buffer,"\n")
6012     if tight and options.tightLists then
6013         return {"\\markdownRenderer01BeginTight\n",contents,
6014             "\n\\markdownRenderer01EndTight "}
6015     else
6016         return {"\\markdownRenderer01Begin\n",contents,
6017             "\n\\markdownRenderer01End "}
6018     end
6019 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.

```

6020 function self.ordereditem(s,num)
6021     if num ~= nil then
6022         return {"\\markdownRenderer01ItemWithNumber{" ,num,"} ",s,
6023             "\\markdownRenderer01ItemEnd "}
6024     else
6025         return {"\\markdownRenderer01Item ",s,
6026             "\\markdownRenderer01ItemEnd "}
6027     end
6028 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.

```

6029 function self.inline_html_comment(contents)
6030     if self.flatten_inlines then return contents end
6031     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
6032 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.

```

6033 function self.inline_html_tag(contents)
6034     if self.flatten_inlines then return contents end

```

```

6035     return {"\markdownRendererInlineHtmlTag{" ,
6036             self.string(contents), "}" }
6037     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.

```

6038     function self.block_html_element(s)
6039         if not self.is_writing then return "" end
6040         local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6041         return {"\markdownRendererInputBlockHtmlElement{" , name, "}" }
6042     end

```

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

```

6043     function self.emphasis(s)
6044         if self.flatten_inlines then return s end
6045         return {"\markdownRendererEmphasis{" , s, "}" }
6046     end

```

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

```

6047     function self.checkbox(f)
6048         if f == 1.0 then
6049             return "☒ "
6050         elseif f == 0.0 then
6051             return "☐ "
6052         else
6053             return "◻ "
6054         end
6055     end

```

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

```

6056     function self.strong(s)
6057         if self.flatten_inlines then return s end
6058         return {"\markdownRendererStrongEmphasis{" , s, "}" }
6059     end

```

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

```

6060     function self.blockquote(s)
6061         if not self.is_writing then return "" end
6062         return {"\markdownRendererBlockQuoteBegin\n" , s,
6063             "\markdownRendererBlockQuoteEnd "}
6064     end

```

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

```

6065 function self.verbatim(s)
6066     if not self.is_writing then return "" end
6067     s = s:gsub("\n$", "")
6068     local name = util.cache_verbatim(options.cacheDir, s)
6069     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
6070 end

```

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

```

6071 function self.document(d)
6072     local buf = {"\\markdownRendererDocumentBegin\n"}
6073
6074     -- warn against the `hybrid` option
6075     if options.hybrid then
6076         local text = "The `hybrid` option has been soft-deprecated."
6077         local more = "Consider using one of the following better options "
6078             .. "for mixing TeX and markdown: `contentBlocks`, "
6079             .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6080             .. "`texMathSingleBackslash`, and "
6081             .. "`texMathDoubleBackslash`. "
6082             .. "For more information, see the user manual at "
6083             .. "<https://witiko.github.io/markdown/>."
6084         table.insert(buf, self.warning(text, more))
6085     end
6086
6087     -- insert the text of the document
6088     table.insert(buf, d)
6089
6090     -- pop all attributes
6091     table.insert(buf, self.pop_attributes())
6092
6093     table.insert(buf, "\\markdownRendererDocumentEnd")
6094
6095     return buf
6096 end

```

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

```

6097 local seen_identifiers = {}
6098 local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
6099 local function normalize_attributes(attributes, auto_identifiers)
6100     -- normalize attributes
6101     local normalized_attributes = {}
6102     local has_explicit_identifiers = false
6103     local key, value
6104     for _, attribute in ipairs(attributes or {}) do
6105         if attribute:sub(1, 1) == "#" then
6106             table.insert(normalized_attributes, attribute)

```



```

6107     has_explicit_identifiers = true
6108     seen_identifiers[attribute:sub(2)] = true
6109 elseif attribute:sub(1, 1) == "." then
6110     table.insert(normalized_attributes, attribute)
6111 else
6112     key, value = attribute:match(key_value_regex)
6113     if key:lower() == "id" then
6114         table.insert(normalized_attributes, "#" .. value)
6115     elseif key:lower() == "class" then
6116         local classes = {}
6117         for class in value:gmatch("%S+") do
6118             table.insert(classes, class)
6119         end
6120         table.sort(classes)
6121         for _, class in ipairs(classes) do
6122             table.insert(normalized_attributes, "." .. class)
6123         end
6124     else
6125         table.insert(normalized_attributes, attribute)
6126     end
6127 end
6128 end
6129
6130 -- if no explicit identifiers exist, add auto identifiers
6131 if not has_explicit_identifiers and auto_identifiers ~= nil then
6132     local seen_auto_identifiers = {}
6133     for _, auto_identifier in ipairs(auto_identifiers) do
6134         if seen_auto_identifiers[auto_identifier] == nil then
6135             seen_auto_identifiers[auto_identifier] = true
6136             if seen_identifiers[auto_identifier] == nil then
6137                 seen_identifiers[auto_identifier] = true
6138                 table.insert(normalized_attributes,
6139                     "#" .. auto_identifier)
6140             else
6141                 local auto_identifier_number = 1
6142                 while true do
6143                     local numbered_auto_identifier = auto_identifier .. "-"
6144                                             .. auto_identifier_number
6145                     if seen_identifiers[numbered_auto_identifier] == nil then
6146                         seen_identifiers[numbered_auto_identifier] = true
6147                         table.insert(normalized_attributes,
6148                             "#" .. numbered_auto_identifier)
6149                     break
6150                 end
6151                 auto_identifier_number = auto_identifier_number + 1
6152             end
6153         end
6154     end

```

```

6154         end
6155     end
6156 end
6157
6158 -- sort and deduplicate normalized attributes
6159 table.sort(normalized_attributes)
6160 local seen_normalized_attributes = {}
6161 local deduplicated_normalized_attributes = {}
6162 for _, attribute in ipairs(normalized_attributes) do
6163     if seen_normalized_attributes[attribute] == nil then
6164         seen_normalized_attributes[attribute] = true
6165         table.insert(deduplicated_normalized_attributes, attribute)
6166     end
6167 end
6168
6169 return deduplicated_normalized_attributes
6170 end
6171
6172 function self.attributes(attributes, should_normalize_attributes)
6173     local normalized_attributes
6174     if should_normalize_attributes == false then
6175         normalized_attributes = attributes
6176     else
6177         normalized_attributes = normalize_attributes(attributes)
6178     end
6179
6180     local buf = {}
6181     local key, value
6182     for _, attribute in ipairs(normalized_attributes) do
6183         if attribute:sub(1, 1) == "#" then
6184             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6185                 attribute:sub(2), "}"})
6186         elseif attribute:sub(1, 1) == "." then
6187             table.insert(buf, {"\\markdownRendererAttributeName{" ,
6188                 attribute:sub(2), "}"})
6189         else
6190             key, value = attribute:match(key_value_regex)
6191             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6192                 key, "}{" , value, "}"})
6193         end
6194     end
6195
6196     return buf
6197 end

```

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

```
6198 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`.

```
6199 self.attribute_type_levels = {}
6200 setmetatable(self.attribute_type_levels,
6201               { __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`.

```
6202 local function apply_attributes()
6203     local buf = {}
6204     for i = 1, #self.active_attributes do
6205         local start_output = self.active_attributes[i][3]
6206         if start_output ~= nil then
6207             table.insert(buf, start_output)
6208         end
6209     end
6210     return buf
6211 end
6212
6213 local function tear_down_attributes()
6214     local buf = {}
6215     for i = #self.active_attributes, 1, -1 do
6216         local end_output = self.active_attributes[i][4]
6217         if end_output ~= nil then
6218             table.insert(buf, end_output)
6219         end
6220     end
6221     return buf
6222 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.

```
6223 function self.push_attributes(attribute_type, attributes,
6224                               start_output, end_output)
6225     local attribute_type_level
6226     = self.attribute_type_levels[attribute_type]
6227     self.attribute_type_levels[attribute_type]
6228     = attribute_type_level + 1
6229
6230     -- index attributes in a hash table for easy lookup
6231     attributes = attributes or {}
```

```

6232     for i = 1, #attributes do
6233         attributes[attributes[i]] = true
6234     end
6235
6236     local buf = {}
6237     -- handle slicing
6238     if attributes["#" .. self.slice_end_identifer] ~= nil and
6239         self.slice_end_type == "^" then
6240         if self.is_writing then
6241             table.insert(buf, self.undosep())
6242             table.insert(buf, tear_down_attributes())
6243         end
6244         self.is_writing = false
6245     end
6246     if attributes["#" .. self.slice_begin_identifer] ~= nil and
6247         self.slice_begin_type == "^" then
6248         table.insert(buf, apply_attributes())
6249         self.is_writing = true
6250     end
6251     if self.is_writing and start_output ~= nil then
6252         table.insert(buf, start_output)
6253     end
6254     table.insert(self.active_attributes,
6255                 {attribute_type, attributes,
6256                  start_output, end_output})
6257     return buf
6258 end
6259

```

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`).

```

6260 function self.pop_attributes(attribute_type)
6261     local buf = {}
6262     -- pop attributes until we find attributes of correct type
6263     -- or until no attributes remain
6264     local current_attribute_type = false
6265     while current_attribute_type ~= attribute_type and
6266         #self.active_attributes > 0 do
6267         local attributes, _, end_output
6268         current_attribute_type, attributes, _, end_output = table.unpack(
6269             self.active_attributes[#self.active_attributes])
6270         local attribute_type_level
6271         = self.attribute_type_levels[current_attribute_type]

```

```

6272     self.attribute_type_levels[current_attribute_type]
6273         = attribute_type_level - 1
6274     if self.is_writing and end_output ~= nil then
6275         table.insert(buf, end_output)
6276     end
6277     table.remove(self.active_attributes, #self.active_attributes)
6278     -- handle slicing
6279     if attributes["#" .. self.slice_end_identifier] ~= nil
6280         and self.slice_end_type == "$" then
6281         if self.is_writing then
6282             table.insert(buf, self.undosep())
6283             table.insert(buf, tear_down_attributes())
6284         end
6285         self.is_writing = false
6286     end
6287     if attributes["#" .. self.slice_begin_identifier] ~= nil and
6288         self.slice_begin_type == "$" then
6289         self.is_writing = true
6290         table.insert(buf, apply_attributes())
6291     end
6292 end
6293 return buf
6294 end

```

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

```

6295 local function create_auto_identifier(s)
6296     local buffer = {}
6297     local prev_space = false
6298     local letter_found = false
6299     local normalized_s = s
6300     if not options.unicodeNormalization
6301         or options.unicodeNormalizationForm ~= "nfc" then
6302         normalized_s = uni_algos.normalize.NFC(normalized_s)
6303     end
6304
6305     for _, code in utf8.codes(normalized_s) do
6306         local char = utf8.char(code)
6307
6308         -- Remove everything up to the first letter.
6309         if not letter_found then
6310             local is_letter = unicode.utf8.match(char, "%a")
6311             if is_letter then
6312                 letter_found = true
6313             else
6314                 goto continue
6315             end
6316         end
6317     end

```

```

6318     -- Remove all non-alphanumeric characters, except underscores,
6319     -- hyphens, and periods.
6320     if not unicode.utf8.match(char, "[%w_-%.%s]") then
6321         goto continue
6322     end
6323
6324     -- Replace all spaces and newlines with hyphens.
6325     if unicode.utf8.match(char, "[%s\n]") then
6326         char = "-"
6327         if prev_space then
6328             goto continue
6329         else
6330             prev_space = true
6331         end
6332     else
6333         -- Convert all alphabetic characters to lowercase.
6334         char = unicode.utf8.lower(char)
6335         prev_space = false
6336     end
6337
6338     table.insert(buffer, char)
6339
6340     ::continue::
6341 end
6342
6343 if prev_space then
6344     table.remove(buffer)
6345 end
6346
6347 local identifier = #buffer == 0 and "section"
6348                  or table.concat(buffer, "")
6349 return identifier
6350 end

```

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

```

6351 local function create_gfm_auto_identifier(s)
6352     local buffer = {}
6353     local prev_space = false
6354     local letter_found = false
6355     local normalized_s = s
6356     if not options.unicodeNormalization
6357         or options.unicodeNormalizationForm ~= "nfc" then
6358         normalized_s = uni_algos.normalize.NFC(normalized_s)
6359     end
6360
6361     for _, code in utf8.codes(normalized_s) do

```

```

6362     local char = utf8.char(code)
6363
6364     -- Remove everything up to the first non-space.
6365     if not letter_found then
6366         local is_letter = unicode.utf8.match(char, "%S")
6367         if is_letter then
6368             letter_found = true
6369         else
6370             goto continue
6371         end
6372     end
6373
6374     -- Remove all non-alphanumeric characters, except underscores
6375     -- and hyphens.
6376     if not unicode.utf8.match(char, "[%w_-%s]") then
6377         prev_space = false
6378         goto continue
6379     end
6380
6381     -- Replace all spaces and newlines with hyphens.
6382     if unicode.utf8.match(char, "[%s\n]") then
6383         char = "-"
6384         if prev_space then
6385             goto continue
6386         else
6387             prev_space = true
6388         end
6389     else
6390         -- Convert all alphabetic characters to lowercase.
6391         char = unicode.utf8.lower(char)
6392         prev_space = false
6393     end
6394
6395     table.insert(buffer, char)
6396
6397     ::continue::
6398 end
6399
6400 if prev_space then
6401     table.remove(buffer)
6402 end
6403
6404 local identifier = #buffer == 0 and "section"
6405                 or table.concat(buffer, "")
6406 return identifier
6407 end

```

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

```
6408 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6409 self.secend_text = "\n\\markdownRendererSectionEnd "
6410 function self.heading(s, level, attributes)
6411   local buf = {}
6412   local flat_text, inlines = table.unpack(s)
6413
6414   -- push empty attributes for implied sections
6415   while self.attribute_type_levels["heading"] < level - 1 do
6416     table.insert(buf,
6417                 self.push_attributes("heading",
6418                                     nil,
6419                                     self.secbegin_text,
6420                                     self.secend_text))
6421   end
6422
6423   -- pop attributes for sections that have ended
6424   while self.attribute_type_levels["heading"] >= level do
6425     table.insert(buf, self.pop_attributes("heading"))
6426   end
6427
6428   -- construct attributes for the new section
6429   local auto_identifiers = {}
6430   if self.options.autoIdentifiers then
6431     table.insert(auto_identifiers, create_auto_identifier(flat_text))
6432   end
6433   if self.options.gfmAutoIdentifiers then
6434     table.insert(auto_identifiers,
6435                 create_gfm_auto_identifier(flat_text))
6436   end
6437   local normalized_attributes = normalize_attributes(attributes,
6438                                                     auto_identifiers)
6439
6440   -- push attributes for the new section
6441   local start_output = {}
6442   local end_output = {}
6443   table.insert(start_output, self.secbegin_text)
6444   table.insert(end_output, self.secend_text)
6445
6446   table.insert(buf, self.push_attributes("heading",
6447                                         normalized_attributes,
6448                                         start_output,
6449                                         end_output))
6450   assert(self.attribute_type_levels["heading"] == level)
6451
6452   -- render the heading and its attributes
```



```

6453     if self.is_writing and #normalized_attributes > 0 then
6454         table.insert(buf,
6455             "\\markdownRendererHeaderAttributeContextBegin\n")
6456         table.insert(buf, self.attributes(normalized_attributes, false))
6457     end
6458
6459     local cmd
6460     level = level + options.shiftHeadings
6461     if level <= 1 then
6462         cmd = "\\markdownRendererHeadingOne"
6463     elseif level == 2 then
6464         cmd = "\\markdownRendererHeadingTwo"
6465     elseif level == 3 then
6466         cmd = "\\markdownRendererHeadingThree"
6467     elseif level == 4 then
6468         cmd = "\\markdownRendererHeadingFour"
6469     elseif level == 5 then
6470         cmd = "\\markdownRendererHeadingFive"
6471     elseif level >= 6 then
6472         cmd = "\\markdownRendererHeadingSix"
6473     else
6474         cmd = ""
6475     end
6476     if self.is_writing then
6477         table.insert(buf, {cmd, "{", inlines, "}"})
6478     end
6479
6480     if self.is_writing and #normalized_attributes > 0 then
6481         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6482     end
6483
6484     return buf
6485 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.

```

6486 function self.get_state()
6487     return {
6488         is_writing=self.is_writing,
6489         flatten_inlines=self.flatten_inlines,
6490         active_attributes={table.unpack(self.active_attributes)},
6491     }
6492 end

```

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

```

6493 function self.set_state(s)
6494     local previous_state = self.get_state()

```

```

6495     for key, value in pairs(s) do
6496         self[key] = value
6497     end
6498     return previous_state
6499 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`.

```

6500 function self.defer_call(f)
6501     local previous_state = self.get_state()
6502     return function(...)
6503         local state = self.set_state(previous_state)
6504         local return_value = f(...)
6505         self.set_state(state)
6506         return return_value
6507     end
6508 end
6509
6510 return self
6511 end

```

3.1.4 Parsers

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

```

6512 local parsers = {}

```

3.1.4.1 Basic Parsers

```

6513 parsers.percent = P("%")
6514 parsers.at = P("@")
6515 parsers.comma = P(",")
6516 parsers.asterisk = P("*")
6517 parsers.dash = P("-")
6518 parsers.plus = P("+")
6519 parsers.underscore = P("_")
6520 parsers.period = P(".")
6521 parsers.hash = P("#")
6522 parsers.dollar = P("$")
6523 parsers.ampersand = P("&")
6524 parsers.backtick = P("`")
6525 parsers.less = P("<")
6526 parsers.more = P(">")
6527 parsers.space = P(" ")
6528 parsers.squote = P("'")
6529 parsers.dquote = P('"')

```

```

6530 parsers.lparent      = P("(")
6531 parsers.rparent      = P(")")
6532 parsers.lbracket     = P("[")
6533 parsers.rbracket     = P("]")
6534 parsers.lbrace       = P("{")
6535 parsers.rbrace       = P("}")
6536 parsers.circumflex   = P("^")
6537 parsers.slash        = P("/")
6538 parsers.equal        = P("=")
6539 parsers.colon        = P(":")
6540 parsers.semicolon    = P(";")
6541 parsers.exclamation  = P("!")
6542 parsers.pipe         = P("|")
6543 parsers.tilde        = P("~")
6544 parsers.backslash    = P("\\")
6545 parsers.tab          = P("\t")
6546 parsers.newline     = P("\n")
6547
6548 parsers.digit        = R("09")
6549 parsers.hexdigit     = R("09", "af", "AF")
6550 parsers.letter       = R("AZ", "az")
6551 parsers.alphanumeric = R("AZ", "az", "09")
6552 parsers.keyword     = parsers.letter
6553                    * (parsers.alphanumeric + parsers.dash)^0
6554
6555 parsers.doubleasterisks = P("**")
6556 parsers.doubleunderscores = P("__")
6557 parsers.doubletildes   = P("~~")
6558 parsers.fourspaces    = P("    ")
6559
6560 parsers.any          = P(1)
6561 parsers.succeed     = P(true)
6562 parsers.fail        = P(false)
6563
6564 parsers.internal_punctuation = S(":,.?")
6565 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
6566

```

3.1.5 Unicode punctuation

This section documents the Unicode punctuation³³ recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

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

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

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.

```

6567 ;(function()
6568   local pathname = assert(kpse.find_file("UnicodeData.txt"),
6569     [[Could not locate file "UnicodeData.txt"]])
6570   local file = assert(io.open(pathname, "r"),
6571     [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct a prefix tree of UTF-8 encodings for all codepoints of a given code length.

```

6572   local prefix_trees = {}
6573   for line in file:lines() do
6574     local codepoint, major_category = line:match("^(%x+);[^\;]*;(%a)")
6575     if major_category == "P" or major_category == "S" then
6576       local code = unicode.utf8.char(tonumber(codepoint, 16))
6577       if prefix_trees[#code] == nil then
6578         prefix_trees[#code] = {}
6579       end
6580       local node = prefix_trees[#code]
6581       for i = 1, #code do
6582         local byte = code:sub(i, i)
6583         if i < #code then
6584           if node[byte] == nil then
6585             node[byte] = {}
6586           end
6587           node = node[byte]
6588         else
6589           table.insert(node, byte)
6590         end
6591       end
6592     end
6593   end
6594   assert(file:close())
6595

```

Next, we will construct a parser out of the prefix tree.

```

6596   local function depth_first_search(node, path, visit, leave)
6597     visit(node, path)
6598     for label, child in pairs(node) do
6599       if type(child) == "table" then
6600         depth_first_search(child, path .. label, visit, leave)
6601       else
6602         visit(child, path)
6603       end
6604     end
6605     leave(node, path)

```

```

6606 end
6607
6608 print("M.punctuation = {}")
6609 print("local S = lpeg.S")
6610 print("-- luacheck: push no max line length")
6611 for length, prefix_tree in pairs(prefix_trees) do
6612     local subparsers = {}
6613     depth_first_search(prefix_tree, "", function(node, path)
6614         if type(node) == "string" then
6615             local suffix
6616             if node == "]" then
6617                 suffix = "S('" .. node .. "'"")"
6618             else
6619                 suffix = "S([[ " .. node .. " ]])"
6620             end
6621             if subparsers[path] ~= nil then
6622                 subparsers[path] = subparsers[path] .. " + " .. suffix
6623             else
6624                 subparsers[path] = suffix
6625             end
6626         end
6627     end, function(_, path)
6628         if #path > 0 then
6629             local byte = path:sub(#path, #path)
6630             local parent_path = path:sub(1, #path-1)
6631             if subparsers[path] ~= nil then
6632                 local suffix
6633                 if byte == "]" then
6634                     suffix = "S('" .. byte .. "'"")"
6635                 else
6636                     suffix = "S([[ " .. byte .. " ]])"
6637                 end
6638                 suffix = suffix .. " * (" .. subparsers[path] .. ")"
6639                 if subparsers[parent_path] ~= nil then
6640                     subparsers[parent_path] = subparsers[parent_path]
6641                         .. " + " .. suffix
6642                 else
6643                     subparsers[parent_path] = suffix
6644                 end
6645             end
6646         else
6647             print("M.punctuation[" .. length .. "] = " .. subparsers[path])
6648         end
6649     end)
6650 end
6651 print("-- luacheck: pop")
6652 end)()

```

```
6653 print("return M")
```

Back in the Markdown package, we will load the precompiled parser of Unicode punctuation.

```
6654 local unicode_data = require("markdown-unicode-data")
6655 if metadata.version ~= unicode_data.metadata.version then
6656     util.warning(
6657         "markdown.lua " .. metadata.version .. " used with " ..
6658         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
6659     )
6660 end
6661 parsers.punctuation = unicode_data.punctuation
6662
6663 parsers.escapable           = parsers.ascii_punctuation
6664 parsers.anyescaped         = parsers.backslash / ""
6665                             * parsers.escapable
6666                             + parsers.any
6667
6668 parsers.spacechar          = S("\t ")
6669 parsers.spacing            = S(" \n\r\t")
6670 parsers.nonspacechar       = parsers.any - parsers.spacing
6671 parsers.optionalspace     = parsers.spacechar^0
6672
6673 parsers.normalchar         = parsers.any - (V("SpecialChar")
6674                                             + parsers.spacing)
6675 parsers.eof                 = -parsers.any
6676 parsers.nonindentSPACE     = parsers.space^-3 * - parsers.spacechar
6677 parsers.indent             = parsers.space^-3 * parsers.tab
6678                             + parsers.fourspaces / ""
6679 parsers.linechar           = P(1 - parsers.newline)
6680
6681 parsers.blankline          = parsers.optionalspace
6682                             * parsers.newline / "\n"
6683 parsers.blanklines         = parsers.blankline^0
6684 parsers.skipblanklines     = ( parsers.optionalspace
6685                             * parsers.newline)^0
6686 parsers.indentedline       = parsers.indent / ""
6687                             * C( parsers.linechar^1
6688                                 * parsers.newline^-1)
6689 parsers.optionallyindentedline = parsers.indent^-1 / ""
6690                             * C( parsers.linechar^1
6691                                 * parsers.newline^-1)
6692 parsers.sp                  = parsers.spacing^0
6693 parsers.spnl               = parsers.optionalspace
6694                             * ( parsers.newline
6695                                 * parsers.optionalspace)^-1
6696 parsers.line                = parsers.linechar^0 * parsers.newline
```

```
6697 parsers.nonemptyline          = parsers.line - parsers.blankline
```

3.1.5.1 Parsers Used for Indentation

```
6698  
6699 parsers.leader          = parsers.space^-3  
6700
```

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

```
6701 local function has_trail(indent_table)  
6702   return indent_table ~= nil and  
6703     indent_table.trail ~= nil and  
6704     next(indent_table.trail) ~= nil  
6705 end  
6706
```

Check if indent table `indent_table` has any indents.

```
6707 local function has_indents(indent_table)  
6708   return indent_table ~= nil and  
6709     indent_table.indents ~= nil and  
6710     next(indent_table.indents) ~= nil  
6711 end  
6712
```

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

```
6713 local function add_trail(indent_table, trail_info)  
6714   indent_table.trail = trail_info  
6715   return indent_table  
6716 end  
6717
```

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

```
6718 local function remove_trail(indent_table)  
6719   indent_table.trail = nil  
6720   return indent_table  
6721 end  
6722
```

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

```
6723 local function update_indent_table(indent_table, new_indent, add)  
6724   indent_table = remove_trail(indent_table)  
6725  
6726   if not has_indents(indent_table) then  
6727     indent_table.indents = {}  
6728   end  
6729  
6730  
6731   if add then  
6732     indent_table.indents[#indent_table.indents + 1] = new_indent
```

```

6733 else
6734     if indent_table.indents[#indent_table.indents].name
6735         == new_indent.name then
6736         indent_table.indents[#indent_table.indents] = nil
6737     end
6738 end
6739
6740 return indent_table
6741 end
6742

```

Remove an indent by its name `name`.

```

6743 local function remove_indent(name)
6744     local remove_indent_level =
6745         function(s, i, indent_table) -- luacheck: ignore s i
6746             indent_table = update_indent_table(indent_table, {name=name},
6747                 false)
6748             return true, indent_table
6749         end
6750
6751     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6752 end
6753

```

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`.

```

6754 local function process_starter_spacing(indent, spacing,
6755                                     minimum, left_strip_length)
6756     left_strip_length = left_strip_length or 0
6757
6758     local count = 0
6759     local tab_value = 4 - (indent) % 4
6760
6761     local code_started, minimum_found = false, false
6762     local code_start, minimum_remainder = "", ""
6763
6764     local left_total_stripped = 0
6765     local full_remainder = ""
6766
6767     if spacing ~= nil then
6768         for i = 1, #spacing do
6769             local character = spacing:sub(i, i)
6770
6771             if character == "\t" then

```



```

6772         count = count + tab_value
6773         tab_value = 4
6774     elseif character == " " then
6775         count = count + 1
6776         tab_value = 4 - (1 - tab_value) % 4
6777     end
6778
6779     if (left_strip_length ~= 0) then
6780         local possible_to_strip = math.min(count, left_strip_length)
6781         count = count - possible_to_strip
6782         left_strip_length = left_strip_length - possible_to_strip
6783         left_total_stripped = left_total_stripped + possible_to_strip
6784     else
6785         full_remainder = full_remainder .. character
6786     end
6787
6788     if (minimum_found) then
6789         minimum_remainder = minimum_remainder .. character
6790     elseif (count >= minimum) then
6791         minimum_found = true
6792         minimum_remainder = minimum_remainder
6793             .. string.rep(" ", count - minimum)
6794     end
6795
6796     if (code_started) then
6797         code_start = code_start .. character
6798     elseif (count >= minimum + 4) then
6799         code_started = true
6800         code_start = code_start
6801             .. string.rep(" ", count - (minimum + 4))
6802     end
6803 end
6804 end
6805
6806 local remainder
6807 if (code_started) then
6808     remainder = code_start
6809 else
6810     remainder = string.rep(" ", count - minimum)
6811 end
6812
6813 local is_minimum = count >= minimum
6814 return {
6815     is_code = code_started,
6816     remainder = remainder,
6817     left_total_stripped = left_total_stripped,
6818     is_minimum = is_minimum,

```

```

6819     minimum_remainder = minimum_remainder,
6820     total_length = count,
6821     full_remainder = full_remainder
6822 }
6823 end
6824

```

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

```

6825 local function count_indent_tab_level(indent_table)
6826     local count = 0
6827     if not has_indents(indent_table) then
6828         return count
6829     end
6830
6831     for i=1, #indent_table.indents do
6832         count = count + indent_table.indents[i].length
6833     end
6834     return count
6835 end
6836

```

Count the total width of a delimiter `delimiter`.

```

6837 local function total_delimiter_length(delimiter)
6838     local count = 0
6839     if type(delimiter) == "string" then return #delimiter end
6840     for _, value in pairs(delimiter) do
6841         count = count + total_delimiter_length(value)
6842     end
6843     return count
6844 end
6845

```

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`.

```

6846 local function process_starter_indent(_, _, indent_table, starter,
6847                                     is_blank, indent_type, breakable)
6848     local last_trail = starter[1]
6849     local delimiter = starter[2]
6850     local raw_new_trail = starter[3]
6851
6852     if indent_type == "bq" and not breakable then
6853         indent_table.ignore_blockquote_blank = true
6854     end
6855
6856     if has_trail(indent_table) then
6857         local trail = indent_table.trail
6858         if trail.is_code then
6859             return false

```

```

6860     end
6861     last_trail = trail.remainder
6862 else
6863     local sp = process_starter_spacing(0, last_trail, 0, 0)
6864
6865     if sp.is_code then
6866         return false
6867     end
6868     last_trail = sp.remainder
6869 end
6870
6871 local preceding_indentation = count_indent_tab_level(indent_table) % 4
6872 local last_trail_length = #last_trail
6873 local delimiter_length = total_delimiter_length(delimiter)
6874
6875 local total_indent_level = preceding_indentation + last_trail_length
6876                        + delimiter_length
6877
6878 local sp = {}
6879 if not is_blank then
6880     sp = process_starter_spacing(total_indent_level, raw_new_trail,
6881                                0, 1)
6882 end
6883
6884 local del_trail_length = sp.left_total_stripped
6885 if is_blank then
6886     del_trail_length = 1
6887 elseif not sp.is_code then
6888     del_trail_length = del_trail_length + #sp.remainder
6889 end
6890
6891 local indent_length = last_trail_length + delimiter_length
6892                        + del_trail_length
6893 local new_indent_info = {name=indent_type, length=indent_length}
6894
6895 indent_table = update_indent_table(indent_table, new_indent_info,
6896                                   true)
6897 indent_table = add_trail(indent_table,
6898                          {is_code=sp.is_code,
6899                           remainder=sp.remainder,
6900                           total_length=sp.total_length,
6901                           full_remainder=sp.full_remainder})
6902
6903 return true, indent_table
6904 end
6905

```

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

```
6906 local function decode_pattern(name)
6907   local delimiter = parsers.succeed
6908   if name == "bq" then
6909     delimiter = parsers.more
6910   end
6911
6912   return C(parsers.optionalspace) * C(delimiter)
6913         * C(parsers.optionalspace) * Cp()
6914 end
6915
```

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

```
6916 local function left_blank_starter(indent_table)
6917   local blank_starter_index
6918
6919   if not has_indents(indent_table) then
6920     return
6921   end
6922
6923   for i = #indent_table.indents,1,-1 do
6924     local value = indent_table.indents[i]
6925     if value.name == "li" then
6926       blank_starter_index = i
6927     else
6928       break
6929     end
6930   end
6931
6932   return blank_starter_index
6933 end
6934
```

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.

```
6935 local function traverse_indent(s, i, indent_table, is_optional,
6936                               is_blank, current_line_indents)
6937   local new_index = i
6938
6939   local preceding_indentation = 0
6940   local current_trail = {}
6941
6942   local blank_starter = left_blank_starter(indent_table)
```

```

6943
6944 if current_line_indents == nil then
6945     current_line_indents = {}
6946 end
6947
6948 for index = 1,#indent_table.indents do
6949     local value = indent_table.indents[index]
6950     local pattern = decode_pattern(value.name)
6951
6952     -- match decoded pattern
6953     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6954     if new_indent_info == nil then
6955         local blankline_end = lpeg.match(
6956             Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
6957         if is_optional or not indent_table.ignore_blockquote_blank
6958             or not blankline_end then
6959             return is_optional, new_index, current_trail,
6960                 current_line_indents
6961         end
6962
6963         return traverse_indent(s, tonumber(blankline_end.pos),
6964             indent_table, is_optional, is_blank,
6965             current_line_indents)
6966     end
6967
6968     local raw_last_trail = new_indent_info[1]
6969     local delimiter = new_indent_info[2]
6970     local raw_new_trail = new_indent_info[3]
6971     local next_index = new_indent_info[4]
6972
6973     local space_only = delimiter == ""
6974
6975     -- check previous trail
6976     if not space_only and next(current_trail) == nil then
6977         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6978         current_trail = {is_code=sp.is_code, remainder=sp.remainder,
6979             total_length=sp.total_length,
6980             full_remainder=sp.full_remainder}
6981     end
6982
6983     if next(current_trail) ~= nil then
6984         if not space_only and current_trail.is_code then
6985             return is_optional, new_index, current_trail,
6986                 current_line_indents
6987         end
6988         if current_trail.internal_remainder ~= nil then
6989             raw_last_trail = current_trail.internal_remainder

```

```

6990     end
6991 end
6992
6993 local raw_last_trail_length = 0
6994 local delimiter_length = 0
6995
6996 if not space_only then
6997     delimiter_length = #delimiter
6998     raw_last_trail_length = #raw_last_trail
6999 end
7000
7001 local total_indent_level = preceding_indentation
7002     + raw_last_trail_length + delimiter_length
7003
7004 local spacing_to_process
7005 local minimum = 0
7006 local left_strip_length = 0
7007
7008 if not space_only then
7009     spacing_to_process = raw_new_trail
7010     left_strip_length = 1
7011 else
7012     spacing_to_process = raw_last_trail
7013     minimum = value.length
7014 end
7015
7016 local sp = process_starter_spacing(total_indent_level,
7017     spacing_to_process, minimum,
7018     left_strip_length)
7019
7020 if space_only and not sp.is_minimum then
7021     return is_optional or (is_blank and blank_starter <= index),
7022     new_index, current_trail, current_line_indents
7023 end
7024
7025 local indent_length = raw_last_trail_length + delimiter_length
7026     + sp.left_total_stripped
7027
7028 -- update info for the next pattern
7029 if not space_only then
7030     preceding_indentation = preceding_indentation + indent_length
7031 else
7032     preceding_indentation = preceding_indentation + value.length
7033 end
7034
7035 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7036     internal_remainder=sp.minimum_remainder,

```

```

7037             total_length=sp.total_length,
7038             full_remainder=sp.full_remainder}
7039
7040     current_line_indents[#current_line_indents + 1] = new_indent_info
7041     new_index = next_index
7042 end
7043
7044 return true, new_index, current_trail, current_line_indents
7045 end
7046

```

Check if a code trail is expected.

```

7047 local function check_trail(expect_code, is_code)
7048     return (expect_code and is_code) or (not expect_code and not is_code)
7049 end
7050

```

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`.

```

7051 local check_trail_joined =
7052     function(s, i, indent_table, -- luacheck: ignore s i
7053             spacing, expect_code, omit_remainder)
7054         local is_code
7055         local remainder
7056
7057         if has_trail(indent_table) then
7058             local trail = indent_table.trail
7059             is_code = trail.is_code
7060             if is_code then
7061                 remainder = trail.remainder
7062             else
7063                 remainder = trail.full_remainder
7064             end
7065         else
7066             local sp = process_starter_spacing(0, spacing, 0, 0)
7067             is_code = sp.is_code
7068             if is_code then
7069                 remainder = sp.remainder
7070             else
7071                 remainder = sp.full_remainder
7072             end
7073         end
7074
7075         local result = check_trail(expect_code, is_code)
7076         if omit_remainder then
7077             return result
7078         end

```

```

7079     return result, remainder
7080 end
7081

```

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

```

7082 local check_trail_length =
7083   function(s, i, indent_table, -- luacheck: ignore s i
7084           spacing, min, max)
7085     local trail
7086
7087     if has_trail(indent_table) then
7088       trail = indent_table.trail
7089     else
7090       trail = process_starter_spacing(0, spacing, 0, 0)
7091     end
7092
7093     local total_length = trail.total_length
7094     if total_length == nil then
7095       return false
7096     end
7097
7098     return min <= total_length and total_length <= max
7099   end
7100

```

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

```

7101 local function check_continuation_indentation(s, i, indent_table,
7102                                             is_optional, is_blank)
7103   if not has_indents(indent_table) then
7104     return true
7105   end
7106
7107   local passes, new_index, current_trail, current_line_indents =
7108     traverse_indent(s, i, indent_table, is_optional, is_blank)
7109
7110   if passes then
7111     indent_table.current_line_indents = current_line_indents
7112     indent_table = add_trail(indent_table, current_trail)
7113     return new_index, indent_table
7114   end
7115   return false
7116 end
7117

```

Get name of the last indent from the `indent_table`.

```

7118 local function get_last_indent_name(indent_table)
7119   if has_indents(indent_table) then

```



```

7120     return indent_table.indents[#indent_table.indents].name
7121 end
7122 end
7123

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

7124 local function remove_remainder_if_blank(indent_table, remainder)
7125     if get_last_indent_name(indent_table) == "li" then
7126         return ""
7127     end
7128     return remainder
7129 end
7130

```

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.

```

7131 local check_trail_type =
7132     function(s, i, -- luacheck: ignore s i
7133             trail, spacing, trail_type)
7134     if trail == nil then
7135         trail = process_starter_spacing(0, spacing, 0, 0)
7136     end
7137
7138     if trail_type == "non-code" then
7139         return check_trail(false, trail.is_code)
7140     end
7141     if trail_type == "code" then
7142         return check_trail(true, trail.is_code)
7143     end
7144     if trail_type == "full-code" then
7145         if (trail.is_code) then
7146             return i, trail.remainder
7147         end
7148         return i, ""
7149     end
7150     if trail_type == "full-any" then
7151         return i, trail.internal_remainder
7152     end
7153 end
7154

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

7155 local trail_freezing =
7156     function(s, i, -- luacheck: ignore s i
7157             indent_table, is_freezing)
7158     if is_freezing then
7159         if indent_table.is_trail_frozen then

```

```

7160     indent_table.trail = indent_table.frozen_trail
7161   else
7162     indent_table.frozen_trail = indent_table.trail
7163     indent_table.is_trail_frozen = true
7164   end
7165   else
7166     indent_table.frozen_trail = nil
7167     indent_table.is_trail_frozen = false
7168   end
7169   return true, indent_table
7170 end
7171

```

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`.

```

7172 local check_continuation_indentation_and_trail =
7173 function (s, i, indent_table, is_optional, is_blank, trail_type,
7174         reset_rem, omit_remainder)
7175   if not has_indents(indent_table) then
7176     local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
7177                                           * Cp(), s, i)
7178     local result, remainder = check_trail_type(s, i,
7179       indent_table.trail, spacing, trail_type)
7180     if remainder == nil then
7181       if result then
7182         return new_index
7183       end
7184       return false
7185     end
7186     if result then
7187       return new_index, remainder
7188     end
7189     return false
7190   end
7191
7192   local passes, new_index, current_trail = traverse_indent(s, i,
7193     indent_table, is_optional, is_blank)
7194
7195   if passes then
7196     local spacing
7197     if current_trail == nil then
7198       local newer_spacing, newer_index = lpeg.match(
7199         C(parsers.spacechar^0) * Cp(), s, i)
7200       current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7201       new_index = newer_index
7202       spacing = newer_spacing

```

```

7203     else
7204         spacing = current_trail.remainer
7205     end
7206     local result, remainder = check_trail_type(s, new_index,
7207         current_trail, spacing, trail_type)
7208     if remainder == nil or omit_remainder then
7209         if result then
7210             return new_index
7211         end
7212         return false
7213     end
7214
7215     if is_blank and reset_rem then
7216         remainder = remove_remainder_if_blank(indent_table, remainder)
7217     end
7218     if result then
7219         return new_index, remainder
7220     end
7221     return false
7222 end
7223 return false
7224 end
7225

```

The following patterns check whitespace indentation at the start of a block.

```

7226 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar~0)
7227     * Cc(false), check_trail_joined)
7228
7229 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7230     * C(parsers.spacechar~0) * Cc(false)
7231     * Cc(true), check_trail_joined)
7232
7233 parsers.check_code_trail = Cmt( Cb("indent_info")
7234     * C(parsers.spacechar~0)
7235     * Cc(true), check_trail_joined)
7236
7237 parsers.check_trail_length_range = function(min, max)
7238     return Cmt( Cb("indent_info") * C(parsers.spacechar~0) * Cc(min)
7239         * Cc(max), check_trail_length)
7240 end
7241
7242 parsers.check_trail_length = function(n)
7243     return parsers.check_trail_length_range(n, n)
7244 end
7245

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7246 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
7247     * Cc(true), trail_freezing), "indent_info")
7248
7249 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7250     trail_freezing), "indent_info")
7251

```

The following patterns check indentation in continuation lines as defined by the container start.

```

7252 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7253     check_continuation_indentation)
7254
7255 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7256     check_continuation_indentation)
7257
7258 parsers.check_minimal_blank_indent
7259     = Cmt( Cb("indent_info") * Cc(false)
7260         * Cc(true)
7261         , check_continuation_indentation)
7262

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

7263
7264 parsers.check_minimal_indent_and_trail =
7265     Cmt( Cb("indent_info")
7266         * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7267         , check_continuation_indentation_and_trail)
7268
7269 parsers.check_minimal_indent_and_code_trail =
7270     Cmt( Cb("indent_info")
7271         * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7272         , check_continuation_indentation_and_trail)
7273
7274 parsers.check_minimal_blank_indent_and_full_code_trail =
7275     Cmt( Cb("indent_info")
7276         * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7277         , check_continuation_indentation_and_trail)
7278
7279 parsers.check_minimal_indent_and_any_trail =
7280     Cmt( Cb("indent_info")
7281         * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7282         , check_continuation_indentation_and_trail)
7283
7284 parsers.check_minimal_blank_indent_and_any_trail =
7285     Cmt( Cb("indent_info")
7286         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7287         , check_continuation_indentation_and_trail)

```

```

7288
7289 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7290   Cmt( Cb("indent_info")
7291     * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7292     , check_continuation_indentation_and_trail)
7293
7294 parsers.check_optional_indent_and_any_trail =
7295   Cmt( Cb("indent_info")
7296     * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7297     , check_continuation_indentation_and_trail)
7298
7299 parsers.check_optional_blank_indent_and_any_trail =
7300   Cmt( Cb("indent_info")
7301     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7302     , check_continuation_indentation_and_trail)
7303

```

The following patterns specify behaviour around newlines.

```

7304
7305 parsers.spnlc_noexc = parsers.optionalspace
7306                   * ( parsers.newline
7307                     * parsers.check_minimal_indent_and_any_trail)^-1
7308
7309 parsers.spnlc = parsers.optionalspace
7310               * (V("EndlineNoSub"))^-1
7311
7312 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7313                 + parsers.spacechar^1
7314
7315 parsers.only_blank = parsers.spacechar^0
7316                   * (parsers.newline + parsers.eof)
7317
7318 % \end{macrocode}
7319 % \begin{figure}
7320 % \hspace*{-0.1\textwidth}
7321 % \begin{minipage}{1.2\textwidth}
7322 % \centering
7323 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
7324 % \node[state, initial by diamond, accepting] (noop) {initial};
7325 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
7326 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
7327 % \node[state] (comment) [below=of noop] {comment};
7328 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
7329 % \node[state] (blank_line) [below right=of comment] {blank line};
7330 % \path[->]
7331 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^\
7332 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
7333 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl

```

```

7334 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [\$^\wedge\wedge\drsh$}
7335 %         edge [bend left=10] node {match $\drsh$} (leading_spaces)
7336 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$}
7337 %         edge [bend right=90] node [right] {match \textbackslash} (odd_backslash)
7338 %         edge [bend left=10] node {match \%} (comment)
7339 %         edge [bend right=10] node {${\epsilon}} (blank_line)
7340 %         edge [bend left=10] node [align=center, right=0.3cm] {match [^\wedge\wedge\drsh$}
7341 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$}
7342 %         edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$}
7343 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2cm]
7344 %         edge [bend right=10] node [align=center, above left=-0.3cm, xshift=0.1cm] {match [^\wedge\wedge\textbackslash}\for \%, capture \textbackslash}
7345 % (even_backslash) edge [bend left=10] node {${\epsilon}} (noop);
7346 % \end{tikzpicture}
7347 % \caption{A pushdown automaton that recognizes \TeX{} comments}
7348 % \label{fig:commented_line}
7349 % \end{minipage}
7350 % \end{figure}
7351 % \begin{markdown}
7352 %
7353 % The \luamdef{parsers.commented_line}^1 parser recognizes the regular
7354 % language of \TeX{} comments, see an equivalent finite automaton in Figure
7355 % <#fig:commented_line>.
7356 %
7357 % \end{markdown}
7358 % \begin{macrocode}
7359 parsers.comment_line_letter = parsers.linechar
7360                               + parsers.newline
7361                               - parsers.backslash
7362                               - parsers.percent
7363 parsers.comment_line = Cg(Cc(""), "backslashes")
7364                       * ((#(parsers.comment_line_letter
7365                           - parsers.newline)
7366                           * Cb("backslashes")
7367                           * Cs(parsers.comment_line_letter
7368                               - parsers.newline)^1 -- initial
7369                           * Cg(Cc(""), "backslashes"))
7370                       + #(parsers.backslash * parsers.backslash)
7371                       * Cg((parsers.backslash -- even backslash
7372                           * parsers.backslash)^1, "backslashes")
7373                       + (parsers.backslash
7374                           * (#parsers.percent
7375                             * Cb("backslashes")
7376                             / function(backslashes)
7377                               return string.rep("\\", #backslashes / 2)
7378                               end
7379                             * C(parsers.percent)

```

```

7380         + #parsers.commented_line_letter
7381         * Cb("backslashes")
7382         * Cc("\\")
7383         * C(parsers.commented_line_letter))
7384         * Cg(Cc(""), "backslashes"))^0
7385     * (#parsers.percent
7386     * Cb("backslashes")
7387     / function(backslashes)
7388     return string.rep("\\", #backslashes / 2)
7389     end
7390     * ((parsers.percent -- comment
7391     * parsers.line
7392     * #parsers.blankline) -- blank line
7393     / "\\n"
7394     + parsers.percent -- comment
7395     * parsers.line
7396     * parsers.optionalspace) -- leading spaces
7397     + #(parsers.newline)
7398     * Cb("backslashes")
7399     * C(parsers.newline))
7400
7401 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7402     - parsers.blankline)^0
7403
7404 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7405 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7406 parsers.attribute_key = (parsers.attribute_key_char
7407     - parsers.dash - parsers.digit)
7408     * parsers.attribute_key_char^0
7409 parsers.attribute_value = ( (parsers.dquote / "")
7410     * (parsers.anyescaped - parsers.dquote)^0
7411     * (parsers.dquote / ""))
7412 + ( (parsers.squote / "")
7413     * (parsers.anyescaped - parsers.squote)^0
7414     * (parsers.squote / ""))
7415 + ( parsers.anyescaped
7416     - parsers.dquote
7417     - parsers.rbrace
7418     - parsers.space)^0
7419 parsers.attribute_identfier = parsers.attribute_key_char^1
7420 parsers.attribute_classname = parsers.letter
7421     * parsers.attribute_key_char^0
7422 parsers.attribute_raw = parsers.attribute_raw_char^1
7423
7424 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7425     + C( parsers.hash
7426     * parsers.attribute_identfier)

```

```

7427         + C( parsers.period
7428             * parsers.attribute_classname)
7429         + Cs( parsers.attribute_key
7430             * parsers.optionalspace
7431             * parsers.equal
7432             * parsers.optionalspace
7433             * parsers.attribute_value)
7434 parsers.attributes = parsers.lbrace
7435                 * parsers.optionalspace
7436                 * parsers.attribute
7437                 * (parsers.spacechar^1
7438                 * parsers.attribute)^0
7439                 * parsers.optionalspace
7440                 * parsers.rbrace
7441
7442 parsers.raw_attribute = parsers.lbrace
7443                 * parsers.optionalspace
7444                 * parsers.equal
7445                 * C(parsers.attribute_raw)
7446                 * parsers.optionalspace
7447                 * parsers.rbrace
7448
7449 -- block followed by 0 or more optionally
7450 -- indented blocks with first line indented.
7451 parsers.indented_blocks = function(bl)
7452   return Cs( bl
7453             * ( parsers.blankline^1
7454             * parsers.indent
7455             * -parsers.blankline
7456             * bl)^0
7457             * (parsers.blankline^1 + parsers.eof) )
7458 end

```

3.1.5.2 Parsers Used for HTML Entities

```

7459 local function repeat_between(pattern, min, max)
7460   return -pattern^(max + 1) * pattern^min
7461 end
7462
7463 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7464                 * C(repeat_between(parsers.hexdigit, 1, 6))
7465                 * parsers.semicolon
7466 parsers.decentity = parsers.ampersand * parsers.hash
7467                 * C(repeat_between(parsers.digit, 1, 7))
7468                 * parsers.semicolon
7469 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7470                 * parsers.semicolon

```



```

7471
7472 parsers.html_entities
7473   = parsers.hexentity / entities.hex_entity_with_x_char
7474   + parsers.decentity / entities.dec_entity
7475   + parsers.tagentity / entities.char_entity

```

3.1.5.3 Parsers Used for Markdown Lists

```

7476 parsers.bullet = function(bullet_char, interrupting)
7477   local allowed_end
7478   if interrupting then
7479     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7480   else
7481     allowed_end = C(parsers.spacechar^1)
7482                 + #(parsers.newline + parsers.eof)
7483   end
7484   return parsers.check_trail
7485         * Ct(C(bullet_char) * Cc(""))
7486         * allowed_end
7487 end
7488
7489 local function tickbox(interior)
7490   return parsers.optionalspace * parsers.lbracket
7491         * interior * parsers.rbracket * parsers.spacechar^1
7492 end
7493
7494 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7495 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7496 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7497

```

3.1.5.4 Parsers Used for Markdown Code Spans

```

7498 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7499
7500 local function captures_equal_length(_,i,a,b)
7501   return #a == #b and i
7502 end
7503
7504 parsers.closeticks = Cmt(C(parsers.backtick^1)
7505                          * Cb("ticks"), captures_equal_length)
7506
7507 parsers.intickschar = (parsers.any - S("\n\r`"))
7508                      + V("NoSoftLineBreakEndline")
7509                      + (parsers.backtick^1 - parsers.closeticks)
7510
7511 local function process_inticks(s)
7512   s = s:gsub("\n", " ")

```

```

7513 s = s:gsub("^ (.*) $", "%1")
7514 return s
7515 end
7516
7517 parsers.inticks = parsers.openticks
7518                 * C(parsers.space^0)
7519                 * parsers.closeticks
7520                 + parsers.openticks
7521                 * Cs(Cs(parsers.intickschar^0) / process_inticks)
7522                 * parsers.closeticks
7523

```

3.1.5.5 Parsers Used for HTML

```

7524 -- case-insensitive match (we assume s is lowercase)
7525 -- must be single byte encoding
7526 parsers.keyword_exact = function(s)
7527   local parser = P(0)
7528   for i=1,#s do
7529     local c = s:sub(i,i)
7530     local m = c .. upper(c)
7531     parser = parser * S(m)
7532   end
7533   return parser
7534 end
7535
7536 parsers.special_block_keyword =
7537   parsers.keyword_exact("pre") +
7538   parsers.keyword_exact("script") +
7539   parsers.keyword_exact("style") +
7540   parsers.keyword_exact("textarea")
7541
7542 parsers.block_keyword =
7543   parsers.keyword_exact("address") +
7544   parsers.keyword_exact("article") +
7545   parsers.keyword_exact("aside") +
7546   parsers.keyword_exact("base") +
7547   parsers.keyword_exact("basefont") +
7548   parsers.keyword_exact("blockquote") +
7549   parsers.keyword_exact("body") +
7550   parsers.keyword_exact("caption") +
7551   parsers.keyword_exact("center") +
7552   parsers.keyword_exact("col") +
7553   parsers.keyword_exact("colgroup") +
7554   parsers.keyword_exact("dd") +
7555   parsers.keyword_exact("details") +
7556   parsers.keyword_exact("dialog") +

```

```
7557 parsers.keyword_exact("dir") +
7558 parsers.keyword_exact("div") +
7559 parsers.keyword_exact("dl") +
7560 parsers.keyword_exact("dt") +
7561 parsers.keyword_exact("fieldset") +
7562 parsers.keyword_exact("figcaption") +
7563 parsers.keyword_exact("figure") +
7564 parsers.keyword_exact("footer") +
7565 parsers.keyword_exact("form") +
7566 parsers.keyword_exact("frame") +
7567 parsers.keyword_exact("frameset") +
7568 parsers.keyword_exact("h1") +
7569 parsers.keyword_exact("h2") +
7570 parsers.keyword_exact("h3") +
7571 parsers.keyword_exact("h4") +
7572 parsers.keyword_exact("h5") +
7573 parsers.keyword_exact("h6") +
7574 parsers.keyword_exact("head") +
7575 parsers.keyword_exact("header") +
7576 parsers.keyword_exact("hr") +
7577 parsers.keyword_exact("html") +
7578 parsers.keyword_exact("iframe") +
7579 parsers.keyword_exact("legend") +
7580 parsers.keyword_exact("li") +
7581 parsers.keyword_exact("link") +
7582 parsers.keyword_exact("main") +
7583 parsers.keyword_exact("menu") +
7584 parsers.keyword_exact("menuitem") +
7585 parsers.keyword_exact("nav") +
7586 parsers.keyword_exact("noframes") +
7587 parsers.keyword_exact("ol") +
7588 parsers.keyword_exact("optgroup") +
7589 parsers.keyword_exact("option") +
7590 parsers.keyword_exact("p") +
7591 parsers.keyword_exact("param") +
7592 parsers.keyword_exact("section") +
7593 parsers.keyword_exact("source") +
7594 parsers.keyword_exact("summary") +
7595 parsers.keyword_exact("table") +
7596 parsers.keyword_exact("tbody") +
7597 parsers.keyword_exact("td") +
7598 parsers.keyword_exact("tfoot") +
7599 parsers.keyword_exact("th") +
7600 parsers.keyword_exact("thead") +
7601 parsers.keyword_exact("title") +
7602 parsers.keyword_exact("tr") +
7603 parsers.keyword_exact("track") +
```

```

7604     parsers.keyword_exact("ul")
7605
7606 -- end conditions
7607 parsers.html_blankline_end_condition
7608 = parsers.linechar^0
7609 * ( parsers.newline
7610   * (parsers.check_minimal_blank_indent_and_any_trail
7611     * #parsers.blankline
7612     + parsers.check_minimal_indent_and_any_trail)
7613   * parsers.linechar^1)^0
7614 * (parsers.newline^-1 / "")
7615
7616 local function remove_trailing_blank_lines(s)
7617 return s:gsub("[\n\r]+%s*$", "")
7618 end
7619
7620 parsers.html_until_end = function(end_marker)
7621 return Cs(Cs((parsers.newline
7622   * (parsers.check_minimal_blank_indent_and_any_trail
7623     * #parsers.blankline
7624     + parsers.check_minimal_indent_and_any_trail)
7625   + parsers.linechar - end_marker)^0
7626   * parsers.linechar^0 * parsers.newline^-1)
7627 / remove_trailing_blank_lines)
7628 end
7629
7630 -- attributes
7631 parsers.html_attribute_spacing = parsers.optionalspace
7632                               * V("NoSoftLineBreakEndline")
7633                               * parsers.optionalspace
7634                               + parsers.spacechar^1
7635
7636 parsers.html_attribute_name = ( parsers.letter
7637                               + parsers.colon
7638                               + parsers.underscore)
7639                               * ( parsers.alphanumeric
7640                               + parsers.colon
7641                               + parsers.underscore
7642                               + parsers.period
7643                               + parsers.dash)^0
7644
7645 parsers.html_attribute_value = parsers.squote
7646                               * (parsers.linechar - parsers.squote)^0
7647                               * parsers.squote
7648                               + parsers.dquote
7649                               * (parsers.linechar - parsers.dquote)^0
7650                               * parsers.dquote

```

```

7651         + ( parsers.any
7652           - parsers.spacechar
7653           - parsers.newline
7654           - parsers.dquote
7655           - parsers.squote
7656           - parsers.backtick
7657           - parsers.equal
7658           - parsers.less
7659           - parsers.more)^1
7660
7661 parsers.html_inline_attribute_value = parsers.squote
7662         * (V("NoSoftLineBreakEndline")
7663           + parsers.any
7664           - parsers.blankline^2
7665           - parsers.squote)^0
7666         * parsers.squote
7667         + parsers.dquote
7668         * (V("NoSoftLineBreakEndline")
7669           + parsers.any
7670           - parsers.blankline^2
7671           - parsers.dquote)^0
7672         * parsers.dquote
7673         + (parsers.any
7674           - parsers.spacechar
7675           - parsers.newline
7676           - parsers.dquote
7677           - parsers.squote
7678           - parsers.backtick
7679           - parsers.equal
7680           - parsers.less
7681           - parsers.more)^1
7682
7683 parsers.html_attribute_value_specification
7684 = parsers.optionalspace
7685 * parsers.equal
7686 * parsers.optionalspace
7687 * parsers.html_attribute_value
7688
7689 parsers.html_spnl = parsers.optionalspace
7690         * (V("NoSoftLineBreakEndline")
7691           * parsers.optionalspace)^-1
7692
7693 parsers.html_inline_attribute_value_specification
7694 = parsers.html_spnl
7695 * parsers.equal
7696 * parsers.html_spnl
7697 * parsers.html_inline_attribute_value

```

```

7698
7699 parsers.html_attribute
7700     = parsers.html_attribute_spacing
7701     * parsers.html_attribute_name
7702     * parsers.html_inline_attribute_value_specification^-1
7703
7704 parsers.html_non_newline_attribute
7705     = parsers.spacechar^1
7706     * parsers.html_attribute_name
7707     * parsers.html_attribute_value_specification^-1
7708
7709 parsers.nested_breaking_blank = parsers.newline
7710                               * parsers.check_minimal_blank_indent
7711                               * parsers.blankline
7712
7713 parsers.html_comment_start = P("<!--")
7714
7715 parsers.html_comment_end = P("-->")
7716
7717 parsers.html_comment
7718     = Cs( parsers.html_comment_start
7719           * parsers.html_until_end(parsers.html_comment_end))
7720
7721 parsers.html_inline_comment = (parsers.html_comment_start / "")
7722                             * -P(">") * -P("-->")
7723                             * Cs(( V("NoSoftLineBreakEndline")
7724                                   + parsers.any
7725                                   - parsers.nested_breaking_blank
7726                                   - parsers.html_comment_end)^0)
7727                             * (parsers.html_comment_end / "")
7728
7729 parsers.html_cdatasection_start = P("<![CDATA[")
7730
7731 parsers.html_cdatasection_end = P("]]>")
7732
7733 parsers.html_cdatasection
7734     = Cs( parsers.html_cdatasection_start
7735           * parsers.html_until_end(parsers.html_cdatasection_end))
7736
7737 parsers.html_inline_cdatasection
7738     = parsers.html_cdatasection_start
7739     * Cs(V("NoSoftLineBreakEndline") + parsers.any
7740         - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
7741     * parsers.html_cdatasection_end
7742
7743 parsers.html_declaration_start = P("<!") * parsers.letter
7744

```

```

7745 parsers.html_declaration_end = P(">")
7746
7747 parsers.html_declaration
7748   = Cs( parsers.html_declaration_start
7749         * parsers.html_until_end(parsers.html_declaration_end))
7750
7751 parsers.html_inline_declaration
7752   = parsers.html_declaration_start
7753     * Cs(V("NoSoftLineBreakEndline") + parsers.any
7754         - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
7755     * parsers.html_declaration_end
7756
7757 parsers.html_instruction_start = P("<?")
7758
7759 parsers.html_instruction_end = P("?>")
7760
7761 parsers.html_instruction
7762   = Cs( parsers.html_instruction_start
7763         * parsers.html_until_end(parsers.html_instruction_end))
7764
7765 parsers.html_inline_instruction = parsers.html_instruction_start
7766                                 * Cs( V("NoSoftLineBreakEndline")
7767                                       + parsers.any
7768                                       - parsers.nested_breaking_blank
7769                                       - parsers.html_instruction_end)^0
7770                                 * parsers.html_instruction_end
7771
7772 parsers.html_blankline = parsers.newline
7773                       * parsers.optionalspace
7774                       * parsers.newline
7775
7776 parsers.html_tag_start = parsers.less
7777
7778 parsers.html_tag_closing_start = parsers.less
7779                               * parsers.slash
7780
7781 parsers.html_tag_end = parsers.html_spnl
7782                    * parsers.more
7783
7784 parsers.html_empty_tag_end = parsers.html_spnl
7785                          * parsers.slash
7786                          * parsers.more
7787
7788 -- opening tags
7789 parsers.html_any_open_inline_tag = parsers.html_tag_start
7790                                 * parsers.keyword
7791                                 * parsers.html_attribute^0

```

```

7792             * parsers.html_tag_end
7793
7794 parsers.html_any_open_tag = parsers.html_tag_start
7795             * parsers.keyword
7796             * parsers.html_non_newline_attribute^0
7797             * parsers.html_tag_end
7798
7799 parsers.html_open_tag = parsers.html_tag_start
7800             * parsers.block_keyword
7801             * parsers.html_attribute^0
7802             * parsers.html_tag_end
7803
7804 parsers.html_open_special_tag = parsers.html_tag_start
7805             * parsers.special_block_keyword
7806             * parsers.html_attribute^0
7807             * parsers.html_tag_end
7808
7809 -- incomplete tags
7810 parsers.incomplete_tag_following = parsers.spacechar
7811             + parsers.more
7812             + parsers.slash * parsers.more
7813             + #(parsers.newline + parsers.eof)
7814
7815 parsers.incomplete_special_tag_following = parsers.spacechar
7816             + parsers.more
7817             + #( parsers.newline
7818                 + parsers.eof)
7819
7820 parsers.html_incomplete_open_tag = parsers.html_tag_start
7821             * parsers.block_keyword
7822             * parsers.incomplete_tag_following
7823
7824 parsers.html_incomplete_open_special_tag
7825 = parsers.html_tag_start
7826 * parsers.special_block_keyword
7827 * parsers.incomplete_special_tag_following
7828
7829 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7830             * parsers.block_keyword
7831             * parsers.incomplete_tag_following
7832
7833 parsers.html_incomplete_close_special_tag
7834 = parsers.html_tag_closing_start
7835 * parsers.special_block_keyword
7836 * parsers.incomplete_tag_following
7837
7838 -- closing tags

```



```

7839 parsers.html_close_tag = parsers.html_tag_closing_start
7840                        * parsers.block_keyword
7841                        * parsers.html_tag_end
7842
7843 parsers.html_any_close_tag = parsers.html_tag_closing_start
7844                        * parsers.keyword
7845                        * parsers.html_tag_end
7846
7847 parsers.html_close_special_tag = parsers.html_tag_closing_start
7848                        * parsers.special_block_keyword
7849                        * parsers.html_tag_end
7850
7851 -- empty tags
7852 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7853                        * parsers.keyword
7854                        * parsers.html_attribute^0
7855                        * parsers.html_empty_tag_end
7856
7857 parsers.html_any_empty_tag = parsers.html_tag_start
7858                        * parsers.keyword
7859                        * parsers.html_non_newline_attribute^0
7860                        * parsers.optionalspace
7861                        * parsers.slash
7862                        * parsers.more
7863
7864 parsers.html_empty_tag = parsers.html_tag_start
7865                        * parsers.block_keyword
7866                        * parsers.html_attribute^0
7867                        * parsers.html_empty_tag_end
7868
7869 parsers.html_empty_special_tag = parsers.html_tag_start
7870                        * parsers.special_block_keyword
7871                        * parsers.html_attribute^0
7872                        * parsers.html_empty_tag_end
7873
7874 parsers.html_incomplete_blocks
7875 = parsers.html_incomplete_open_tag
7876 + parsers.html_incomplete_open_special_tag
7877 + parsers.html_incomplete_close_tag
7878
7879 -- parse special html blocks
7880 parsers.html_blankline_ending_special_block_opening
7881 = ( parsers.html_close_special_tag
7882     + parsers.html_empty_special_tag)
7883 * #( parsers.optionalspace
7884     * (parsers.newline + parsers.eof))
7885

```

```

7886 parsers.html_blankline_ending_special_block
7887   = parsers.html_blankline_ending_special_block_opening
7888   * parsers.html_blankline_end_condition
7889
7890 parsers.html_special_block_opening
7891   = parsers.html_incomplete_open_special_tag
7892   - parsers.html_empty_special_tag
7893
7894 parsers.html_closing_special_block
7895   = parsers.html_special_block_opening
7896   * parsers.html_until_end(parsers.html_close_special_tag)
7897
7898 parsers.html_special_block
7899   = parsers.html_blankline_ending_special_block
7900   + parsers.html_closing_special_block
7901
7902 -- parse html blocks
7903 parsers.html_block_opening = parsers.html_incomplete_open_tag
7904                           + parsers.html_incomplete_close_tag
7905
7906 parsers.html_block = parsers.html_block_opening
7907                   * parsers.html_blankline_end_condition
7908
7909 -- parse any html blocks
7910 parsers.html_any_block_opening
7911   = ( parsers.html_any_open_tag
7912     + parsers.html_any_close_tag
7913     + parsers.html_any_empty_tag)
7914   * #(parsers.optionalspace * (parsers.newline + parsers.eof))
7915
7916 parsers.html_any_block = parsers.html_any_block_opening
7917                       * parsers.html_blankline_end_condition
7918
7919 parsers.html_inline_comment_full = parsers.html_comment_start
7920                                 * -P(">") * -P("->")
7921                                 * Cs(( V("NoSoftLineBreakEndline")
7922                                     + parsers.any - P("--")
7923                                     - parsers.nested_breaking_blank
7924                                     - parsers.html_comment_end)^0)
7925                                 * parsers.html_comment_end
7926
7927 parsers.html_inline_tags = parsers.html_inline_comment_full
7928                          + parsers.html_any_empty_inline_tag
7929                          + parsers.html_inline_instruction
7930                          + parsers.html_inline_cdatasection
7931                          + parsers.html_inline_declaration
7932                          + parsers.html_any_open_inline_tag

```

```
7933 + parsers.html_any_close_tag
7934
```

3.1.5.6 Parsers Used for Markdown Tags and Links

```
7935 parsers.urlchar = parsers.anyescaped
7936                 - parsers.newline
7937                 - parsers.more
7938
7939 parsers.auto_link_scheme_part = parsers.alphanumeric
7940                               + parsers.plus
7941                               + parsers.period
7942                               + parsers.dash
7943
7944 parsers.auto_link_scheme = parsers.letter
7945                          * parsers.auto_link_scheme_part
7946                          * parsers.auto_link_scheme_part^-30
7947
7948 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7949                  * ( parsers.any - parsers.spacing
7950                    - parsers.less - parsers.more)^0
7951
7952 parsers.printable_characters = S(" !#$%&'*/+=?^_`{|}~-")
7953
7954 parsers.email_address_local_part_char = parsers.alphanumeric
7955                                       + parsers.printable_characters
7956
7957 parsers.email_address_local_part
7958   = parsers.email_address_local_part_char^1
7959
7960 parsers.email_address_dns_label = parsers.alphanumeric
7961                               * ( parsers.alphanumeric
7962                                 + parsers.dash)^-62
7963                               * B(parsers.alphanumeric)
7964
7965 parsers.email_address_domain = parsers.email_address_dns_label
7966                             * ( parsers.period
7967                               * parsers.email_address_dns_label)^0
7968
7969 parsers.email_address = parsers.email_address_local_part
7970                       * parsers.at
7971                       * parsers.email_address_domain
7972
7973 parsers.auto_link_url = parsers.less
7974                      * C(parsers.absolute_uri)
7975                      * parsers.more
7976
```

```

7977 parsers.auto_link_email = parsers.less
7978                             * C(parsers.email_address)
7979                             * parsers.more
7980
7981 parsers.auto_link_relative_reference = parsers.less
7982                             * C(parsers.urlchar^1)
7983                             * parsers.more
7984
7985 parsers.autolink = parsers.auto_link_url
7986                 + parsers.auto_link_email
7987
7988 -- content in balanced brackets, parentheses, or quotes:
7989 parsers.bracketed = P{ parsers.lbracket
7990                       * (( parsers.backslash / "\"" * parsers.rbracket
7991                           + parsers.any - (parsers.lbracket
7992                                           + parsers.rbracket
7993                                           + parsers.blankline^2)
7994                       ) + V(1))^0
7995                       * parsers.rbracket }
7996
7997 parsers.inparens = P{ parsers.lparent
7998                       * ((parsers.anyescaped - (parsers.lparent
7999                                               + parsers.rparent
8000                                               + parsers.blankline^2)
8001                       ) + V(1))^0
8002                       * parsers.rparent }
8003
8004 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8005                       * ((parsers.anyescaped - (parsers.squote
8006                                               + parsers.blankline^2)
8007                       ) + V(1))^0
8008                       * parsers.squote }
8009
8010 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8011                       * ((parsers.anyescaped - (parsers.dquote
8012                                               + parsers.blankline^2)
8013                       ) + V(1))^0
8014                       * parsers.dquote }
8015
8016 parsers.link_text = parsers.lbracket
8017                   * Cs((parsers.alphanumeric^1
8018                       + parsers.bracketed
8019                       + parsers.inticks
8020                       + parsers.autolink
8021                       + V("InlineHtml"))
8022                   + ( parsers.backslash * parsers.backslash)
8023                   + ( parsers.backslash

```

```

8024         * ( parsers.lbracket
8025           + parsers.rbracket)
8026         + V("NoSoftLineBreakSpace")
8027         + V("NoSoftLineBreakEndline")
8028         + (parsers.any
8029           - ( parsers.newline
8030             + parsers.lbracket
8031             + parsers.rbracket
8032             + parsers.blankline^2))))^0)
8033     * parsers.rbracket
8034
8035 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
8036     * #( ( parsers.any
8037         - parsers.rbracket)^-999
8038     * parsers.rbracket)
8039     * Cs((parsers.alphanumeric^1
8040         + parsers.inticks
8041         + parsers.autolink
8042         + V("InlineHtml")
8043         + ( parsers.backslash * parsers.backslash)
8044         + ( parsers.backslash
8045           * ( parsers.lbracket
8046             + parsers.rbracket)
8047           + V("NoSoftLineBreakSpace")
8048           + V("NoSoftLineBreakEndline")
8049           + (parsers.any
8050             - ( parsers.newline
8051               + parsers.lbracket
8052               + parsers.rbracket
8053               + parsers.blankline^2))))^1)
8054
8055 parsers.link_label = parsers.lbracket
8056     * parsers.link_label_body
8057     * parsers.rbracket
8058
8059 parsers.inparens_url = P{ parsers.lparent
8060     * ((parsers.anyescaped - (parsers.lparent
8061         + parsers.rparent
8062         + parsers.spacing)
8063     ) + V(1))^0
8064     * parsers.rparent }
8065
8066 -- url for markdown links, allowing nested brackets:
8067 parsers.url = parsers.less * Cs((parsers.anyescaped
8068     - parsers.newline
8069     - parsers.less
8070     - parsers.more)^0)

```

```

8071                                     * parsers.more
8072 + -parsers.less
8073 * Cs((parsers.inparens_url + (parsers.anyescaped
8074                               - parsers.spacing
8075                               - parsers.lparent
8076                               - parsers.rparent))^1)
8077
8078 -- quoted text:
8079 parsers.title_s = parsers.squote
8080 * Cs((parsers.html_entities
8081       + V("NoSoftLineBreakSpace")
8082       + V("NoSoftLineBreakEndline")
8083       + ( parsers.anyescaped
8084         - parsers.newline
8085         - parsers.squote
8086         - parsers.blankline^2))^0)
8087 * parsers.squote
8088
8089 parsers.title_d = parsers.dquote
8090 * Cs((parsers.html_entities
8091       + V("NoSoftLineBreakSpace")
8092       + V("NoSoftLineBreakEndline")
8093       + ( parsers.anyescaped
8094         - parsers.newline
8095         - parsers.dquote
8096         - parsers.blankline^2))^0)
8097 * parsers.dquote
8098
8099 parsers.title_p = parsers.lparent
8100 * Cs((parsers.html_entities
8101       + V("NoSoftLineBreakSpace")
8102       + V("NoSoftLineBreakEndline")
8103       + ( parsers.anyescaped
8104         - parsers.newline
8105         - parsers.lparent
8106         - parsers.rparent
8107         - parsers.blankline^2))^0)
8108 * parsers.rparent
8109
8110 parsers.title
8111 = parsers.title_d + parsers.title_s + parsers.title_p
8112
8113 parsers.optionaltitle
8114 = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8115

```

3.1.5.7 Helpers for Links and Link Reference Definitions

```
8116 -- parse a reference definition: [foo]: /bar "title"
8117 parsers.define_reference_parser = (parsers.check_trail / "")
8118                               * parsers.link_label * parsers.colon
8119                               * parsers.spnlc * parsers.url
8120                               * ( parsers.spnlc_sep * parsers.title
8121                               * parsers.only_blank
8122                               + Cc("") * parsers.only_blank)
```

3.1.5.8 Inline Elements

```
8123 parsers.Inline              = V("Inline")
8124
8125 -- parse many p between starter and ender
8126 parsers.between = function(p, starter, ender)
8127   local ender2 = B(parsers.nonspacechar) * ender
8128   return ( starter
8129           * #parsers.nonspacechar
8130           * Ct(p * (p - ender2)^0)
8131           * ender2)
8132 end
8133
```

3.1.5.9 Block Elements

```
8134 parsers.lineof = function(c)
8135   return ( parsers.check_trail_no_rem
8136           * (P(c) * parsers.optionalspace)^3
8137           * (parsers.newline + parsers.eof))
8138 end
8139
8140 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8141                               + parsers.lineof(parsers.dash)
8142                               + parsers.lineof(parsers.underscore)
```

3.1.5.10 Headings

```
8143 -- parse Atx heading start and return level
8144 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8145                       * -parsers.hash / length
8146
8147 -- parse setext header ending and return level
8148 parsers.heading_level
8149   = parsers.nonindentSPACE * parsers.equal^1
8150   * parsers.optionalspace * #parsers.newline * Cc(1)
8151   + parsers.nonindentSPACE * parsers.dash^1
8152   * parsers.optionalspace * #parsers.newline * Cc(2)
8153
```

```

8154 local function strip_atx_end(s)
8155     return s:gsub("%s+##%s*\n$", "")
8156 end
8157
8158 parsers.atx_heading = parsers.check_trail_no_rem
8159                     * Cg(parsers.heading_start, "level")
8160                     * (C( parsers.optionalspace
8161                         * parsers.hash^0
8162                         * parsers.optionalspace
8163                         * parsers.newline)
8164                       + parsers.spacechar^1
8165                       * C(parsers.line))

```

3.1.6 Markdown Reader

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

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

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

```

8166 M.reader = {}
8167 function M.reader.new(writer, options)
8168     local self = {}

```

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

```

8169     self.writer = writer
8170     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.

```

8171     self.parsers = {}
8172     (function(parsers)
8173         setmetatable(self.parsers, {
8174             __index = function (_, key)
8175                 return parsers[key]
8176             end
8177         })
8178     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.

```
8179 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.

```
8180 function self.normalize_tag(tag)
8181     tag = util.ropes_to_string(tag)
8182     tag = tag:gsub("[\n\r\t]+", " ")
8183     tag = tag:gsub("^ ", ""):gsub(" $", "")
8184     tag = uni_algos.case.casefold(tag, true, false)
8185     return tag
8186 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.

```
8187 local function iterlines(s, f)
8188     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8189     return util.ropes_to_string(rope)
8190 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.

```
8191 if options.preserveTabs then
8192     self.expandtabs = function(s) return s end
8193 else
8194     self.expandtabs = function(s)
8195         if s:find("\t") then
8196             return iterlines(s, util.expand_tabs_in_line)
8197         else
8198             return s
8199         end
8200     end
8201 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 `oplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
8202 self.parser_functions = {}
```

```

8203 self.create_parser = function(name, grammar, toplevel)
8204     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.

```

8205     if toplevel and options.stripIndent then
8206         local min_prefix_length, min_prefix = nil, ''
8207         str = iterlines(str, function(line)
8208             if lpeg.match(parsers.nonemptyline, line) == nil then
8209                 return line
8210             end
8211             line = util.expand_tabs_in_line(line)
8212             local prefix = lpeg.match(C(parsers.optionalspace), line)
8213             local prefix_length = #prefix
8214             local is_shorter = min_prefix_length == nil
8215             if not is_shorter then
8216                 is_shorter = prefix_length < min_prefix_length
8217             end
8218             if is_shorter then
8219                 min_prefix_length, min_prefix = prefix_length, prefix
8220             end
8221             return line
8222         end)
8223         str = str:gsub('^' .. min_prefix, '')
8224     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.

```

8225     if toplevel and (options.texComments or options.hybrid) then
8226         str = lpeg.match(Ct(parsers.commented_line^1), str)
8227         str = util.rope_to_string(str)
8228     end
8229     local res = lpeg.match(grammar(), str)
8230     if res == nil then
8231         return writer.error(
8232             format("Parser `%s` failed to process the input text.", name),
8233             format("Here are the first 20 characters of the remaining "
8234                 .. "unprocessed text: `%s`.", str:sub(1,20))
8235         )
8236     else
8237         return res
8238     end
8239 end
8240 end

```

```

8241
8242 self.create_parser("parse_blocks",
8243                     function()
8244                         return parsers.blocks
8245                     end, true)
8246
8247 self.create_parser("parse_blocks_nested",
8248                     function()
8249                         return parsers.blocks_nested
8250                     end, false)
8251
8252 self.create_parser("parse_inlines",
8253                     function()
8254                         return parsers.inlines
8255                     end, false)
8256
8257 self.create_parser("parse_inlines_no_inline_note",
8258                     function()
8259                         return parsers.inlines_no_inline_note
8260                     end, false)
8261
8262 self.create_parser("parse_inlines_no_html",
8263                     function()
8264                         return parsers.inlines_no_html
8265                     end, false)
8266
8267 self.create_parser("parse_inlines_nbsp",
8268                     function()
8269                         return parsers.inlines_nbsp
8270                     end, false)
8271 self.create_parser("parse_inlines_no_link_or_emphasis",
8272                     function()
8273                         return parsers.inlines_no_link_or_emphasis
8274                     end, false)

```

3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8275 parsers.minimally_indented_blankline
8276     = parsers.check_minimal_indent * (parsers.blankline / "")
8277
8278 parsers.minimally_indented_block
8279     = parsers.check_minimal_indent * V("Block")
8280
8281 parsers.minimally_indented_block_or_paragraph
8282     = parsers.check_minimal_indent * V("BlockOrParagraph")
8283

```

```

8284 parsers.minimally_indented_paragraph
8285     = parsers.check_minimal_indent * V("Paragraph")
8286
8287 parsers.minimally_indented_plain
8288     = parsers.check_minimal_indent * V("Plain")
8289
8290 parsers.minimally_indented_par_or_plain
8291     = parsers.minimally_indented_paragraph
8292     + parsers.minimally_indented_plain
8293
8294 parsers.minimally_indented_par_or_plain_no_blank
8295     = parsers.minimally_indented_par_or_plain
8296     - parsers.minimally_indented_blankline
8297
8298 parsers.minimally_indented_ref
8299     = parsers.check_minimal_indent * V("Reference")
8300
8301 parsers.minimally_indented_blank
8302     = parsers.check_minimal_indent * V("Blank")
8303
8304 parsers.conditionally_indented_blankline
8305     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8306
8307 parsers.minimally_indented_ref_or_block
8308     = parsers.minimally_indented_ref
8309     + parsers.minimally_indented_block
8310     - parsers.minimally_indented_blankline
8311
8312 parsers.minimally_indented_ref_or_block_or_par
8313     = parsers.minimally_indented_ref
8314     + parsers.minimally_indented_block_or_paragraph
8315     - parsers.minimally_indented_blankline
8316

```

The following pattern parses the properly indented content that follows the initial container start.

```

8317
8318 function parsers.separator_loop(separated_block, paragraph,
8319                                block_separator, paragraph_separator)
8320     return separated_block
8321         + block_separator
8322         * paragraph
8323         * separated_block
8324         + paragraph_separator
8325         * paragraph
8326 end
8327

```

```

8328 function parsers.create_loop_body_pair(separated_block, paragraph,
8329                                     block_separator,
8330                                     paragraph_separator)
8331     return {
8332         block = parsers.separator_loop(separated_block, paragraph,
8333                                     block_separator, block_separator),
8334         par = parsers.separator_loop(separated_block, paragraph,
8335                                    block_separator, paragraph_separator)
8336     }
8337 end
8338
8339 parsers.block_sep_group = function(blank)
8340     return blank^0 * parsers.eof
8341           + ( blank^2 / writer.paragraphsep
8342             + blank^0 / writer.interblocksep
8343           )
8344 end
8345
8346 parsers.par_sep_group = function(blank)
8347     return blank^0 * parsers.eof
8348           + blank^0 / writer.paragraphsep
8349 end
8350
8351 parsers.sep_group_no_output = function(blank)
8352     return blank^0 * parsers.eof
8353           + blank^0
8354 end
8355
8356 parsers.content_blank = parsers.minimally_indented_blankline
8357
8358 parsers.ref_or_block_separated
8359     = parsers.sep_group_no_output(parsers.content_blank)
8360     * ( parsers.minimally_indented_ref
8361       - parsers.content_blank)
8362     + parsers.block_sep_group(parsers.content_blank)
8363     * ( parsers.minimally_indented_block
8364       - parsers.content_blank)
8365
8366 parsers.loop_body_pair =
8367     parsers.create_loop_body_pair(
8368         parsers.ref_or_block_separated,
8369         parsers.minimally_indented_par_or_plain_no_blank,
8370         parsers.block_sep_group(parsers.content_blank),
8371         parsers.par_sep_group(parsers.content_blank))
8372
8373 parsers.content_loop = ( V("Block")
8374                       * parsers.loop_body_pair.block^0

```

```

8375         + (V("Paragraph") + V("Plain"))
8376         * parsers.ref_or_block_separated
8377         * parsers.loop_body_pair.block^0
8378         + (V("Paragraph") + V("Plain"))
8379         * parsers.loop_body_pair.par^0
8380         * parsers.content_blank^0
8381
8382     parsers.indented_content = function()
8383         return Ct( (V("Reference") + (parsers.blankline / ""))
8384                 * parsers.content_blank^0
8385                 * parsers.check_minimal_indent
8386                 * parsers.content_loop
8387                 + (V("Reference") + (parsers.blankline / ""))
8388                 * parsers.content_blank^0
8389                 + parsers.content_loop)
8390     end
8391
8392     parsers.add_indent = function(pattern, name, breakable)
8393         return Cg(Cmt( Cb("indent_info")
8394                     * Ct(pattern)
8395                     * ( #parsers.linechar -- check if starter is blank
8396                       * Cc(false) + Cc(true))
8397                     * Cc(name)
8398                     * Cc(breakable),
8399                     process_starter_indent), "indent_info")
8400     end
8401
8402

```

3.1.6.4 Parsers Used for Markdown Lists (local)

```

8402     if options.hashEnumerators then
8403         parsers.dig = parsers.digit + parsers.hash
8404     else
8405         parsers.dig = parsers.digit
8406     end
8407
8408     parsers.enumerator = function(delimiter_type, interrupting)
8409         local delimiter_range
8410         local allowed_end
8411         if interrupting then
8412             delimiter_range = P("1")
8413             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8414         else
8415             delimiter_range = parsers.dig * parsers.dig^-8
8416             allowed_end = C(parsers.spacechar^1)
8417                         + #(parsers.newline + parsers.eof)
8418         end

```

```

8419
8420     return parsers.check_trail
8421         * Ct(C(delimiter_range) * C(delimiter_type))
8422         * allowed_end
8423     end
8424
8425     parsers.starter = parsers.bullet(parsers.dash)
8426         + parsers.bullet(parsers.asterisk)
8427         + parsers.bullet(parsers.plus)
8428         + parsers.enumerator(parsers.period)
8429         + parsers.enumerator(parsers.rparent)
8430

```

3.1.6.5 Parsers Used for Blockquotes (local)

```

8431     parsers.blockquote_start
8432     = parsers.check_trail
8433     * C(parsers.more)
8434     * C(parsers.spacechar^0)
8435
8436     parsers.blockquote_body
8437     = parsers.add_indent(parsers.blockquote_start, "bq", true)
8438     * parsers.indented_content()
8439     * remove_indent("bq")
8440
8441     if not options.breakableBlockquotes then
8442         parsers.blockquote_body
8443         = parsers.add_indent(parsers.blockquote_start, "bq", false)
8444         * parsers.indented_content()
8445         * remove_indent("bq")
8446     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.

```

8447     local function parse_content_part(content_part)
8448         local rope = util.rope_to_string(content_part)
8449         local parsed
8450             = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8451         parsed.indent_info = nil
8452         return parsed
8453     end
8454

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8455     local collect_emphasis_content =

```

```

8456     function(t, opening_index, closing_index)
8457         local content = {}
8458
8459         local content_part = {}
8460         for i = opening_index, closing_index do
8461             local value = t[i]
8462
8463             if value.rendered ~= nil then
8464                 content[#content + 1] = parse_content_part(content_part)
8465                 content_part = {}
8466                 content[#content + 1] = value.rendered
8467                 value.rendered = nil
8468             else
8469                 if value.type == "delimiter"
8470                     and value.element == "emphasis" then
8471                     if value.is_active then
8472                         content_part[#content_part + 1]
8473                             = string.rep(value.character, value.current_count)
8474                     end
8475                 else
8476                     content_part[#content_part + 1] = value.content
8477                 end
8478                 value.content = ''
8479                 value.is_active = false
8480             end
8481         end
8482
8483         if next(content_part) ~= nil then
8484             content[#content + 1] = parse_content_part(content_part)
8485         end
8486
8487         return content
8488     end
8489

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

8490     local function fill_emph(t, opening_index, closing_index)
8491         local content
8492             = collect_emphasis_content(t, opening_index + 1,
8493                                     closing_index - 1)
8494         t[opening_index + 1].is_active = true
8495         t[opening_index + 1].rendered = writer.emphasis(content)
8496     end
8497

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.


```

8498 local function fill_strong(t, opening_index, closing_index)
8499     local content
8500         = collect_emphasis_content(t, opening_index + 1,
8501                                     closing_index - 1)
8502     t[opening_index + 1].is_active = true
8503     t[opening_index + 1].rendered = writer.strong(content)
8504 end
8505

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

8506 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8507     return ( opening_delimiter.is_closing
8508             or closing_delimiter.is_opening)
8509     and (( opening_delimiter.original_count
8510           + closing_delimiter.original_count) % 3 == 0)
8511     and ( opening_delimiter.original_count % 3 ~= 0
8512           or closing_delimiter.original_count % 3 ~= 0)
8513 end
8514

```

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`.

```

8515 local find_emphasis_opener = function(t, bottom_index, latest_index,
8516                                     character, closing_delimiter)
8517     for i = latest_index, bottom_index, -1 do
8518         local value = t[i]
8519         if value.is_active and
8520            value.is_opening and
8521            value.type == "delimiter" and
8522            value.element == "emphasis" and
8523            (value.character == character) and
8524            (value.current_count > 0) then
8525             if not breaks_three_rule(value, closing_delimiter) then
8526                 return i
8527             end
8528         end
8529     end
8530 end
8531

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

8532 local function process_emphasis(t, opening_index, closing_index)
8533     for i = opening_index, closing_index do
8534         local value = t[i]
8535         if value.type == "delimiter" and value.element == "emphasis" then

```

```

8536         local delimiter_length = string.len(value.content)
8537         value.character = string.sub(value.content, 1, 1)
8538         value.current_count = delimiter_length
8539         value.original_count = delimiter_length
8540     end
8541 end
8542
8543 local openers_bottom = {
8544     ['*'] = {
8545         [true] = {opening_index, opening_index, opening_index},
8546         [false] = {opening_index, opening_index, opening_index}
8547     },
8548     ['_'] = {
8549         [true] = {opening_index, opening_index, opening_index},
8550         [false] = {opening_index, opening_index, opening_index}
8551     }
8552 }
8553
8554 local current_position = opening_index
8555 local max_position = closing_index
8556
8557 while current_position <= max_position do
8558     local value = t[current_position]
8559
8560     if value.type ~= "delimiter" or
8561        value.element ~= "emphasis" or
8562        not value.is_active or
8563        not value.is_closing or
8564        (value.current_count <= 0) then
8565         current_position = current_position + 1
8566         goto continue
8567     end
8568
8569     local character = value.character
8570     local is_opening = value.is_opening
8571     local closing_length_modulo_three = value.original_count % 3
8572
8573     local current_openers_bottom
8574         = openers_bottom[character][is_opening]
8575           [closing_length_modulo_three + 1]
8576
8577     local opener_position
8578         = find_emphasis_opener(t, current_openers_bottom,
8579                               current_position - 1, character, value)
8580
8581     if (opener_position == nil) then
8582         openers_bottom[character][is_opening]

```

```

8583             [closing_length_modulo_three + 1]
8584             = current_position
8585             current_position = current_position + 1
8586             goto continue
8587         end
8588
8589         local opening_delimiter = t[opener_position]
8590
8591         local current_opening_count = opening_delimiter.current_count
8592         local current_closing_count = t[current_position].current_count
8593
8594         if (current_opening_count >= 2)
8595             and (current_closing_count >= 2) then
8596             opening_delimiter.current_count = current_opening_count - 2
8597             t[current_position].current_count = current_closing_count - 2
8598             fill_strong(t, opener_position, current_position)
8599         else
8600             opening_delimiter.current_count = current_opening_count - 1
8601             t[current_position].current_count = current_closing_count - 1
8602             fill_emph(t, opener_position, current_position)
8603         end
8604
8605         ::continue::
8606     end
8607 end
8608
8609 local cont = lpeg.R("\128\191") -- continuation byte
8610

```

Match a UTF-8 character of byte length `n`.

```

8611 local function utf8_by_byte_count(n)
8612     if (n == 1) then
8613         return lpeg.R("\0\127")
8614     end
8615     if (n == 2) then
8616         return lpeg.R("\194\223") * cont
8617     end
8618     if (n == 3) then
8619         return lpeg.R("\224\239") * cont * cont
8620     end
8621     if (n == 4) then
8622         return lpeg.R("\240\244") * cont * cont * cont
8623     end
8624 end

```

Check if there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```

8625 local function check_unicode_type(s, i, start_pos, end_pos, chartype)

```

```

8626     local c
8627     local char_length
8628     for pos = start_pos, end_pos, 1 do
8629         if (start_pos < 0) then
8630             char_length = -pos
8631         else
8632             char_length = pos + 1
8633         end
8634
8635         if (chartype == "punctuation") then
8636             if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8637                 return i
8638             end
8639         else
8640             c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
8641             if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8642                 return i
8643             end
8644         end
8645     end
8646 end
8647
8648 local function check_preceding_unicode_punctuation(s, i)
8649     return check_unicode_type(s, i, -4, -1, "punctuation")
8650 end
8651
8652 local function check_preceding_unicode_whitespace(s, i)
8653     return check_unicode_type(s, i, -4, -1, "%s")
8654 end
8655
8656 local function check_following_unicode_punctuation(s, i)
8657     return check_unicode_type(s, i, 0, 3, "punctuation")
8658 end
8659
8660 local function check_following_unicode_whitespace(s, i)
8661     return check_unicode_type(s, i, 0, 3, "%s")
8662 end
8663
8664 parsers.unicode_preceding_punctuation
8665     = B(parsers.escapable)
8666     + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8667
8668 parsers.unicode_preceding_whitespace
8669     = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8670
8671 parsers.unicode_following_punctuation
8672     = #parsers.escapable

```

```

8673     + Cmt(parsers.succeed, check_following_unicode_punctuation)
8674
8675     parsers.unicode_following_whitespace
8676     = Cmt(parsers.succeed, check_following_unicode_whitespace)
8677
8678     parsers.delimiter_run = function(character)
8679     return (B(parsers.backslash * character) + -B(character))
8680             * character^1
8681             * -#character
8682     end
8683
8684     parsers.left_flanking_delimiter_run = function(character)
8685     return (B( parsers.any)
8686             * ( parsers.unicode_preceding_punctuation
8687               + parsers.unicode_preceding_whitespace)
8688             + -B(parsers.any))
8689             * parsers.delimiter_run(character)
8690             * parsers.unicode_following_punctuation
8691             + parsers.delimiter_run(character)
8692             * -( parsers.unicode_following_punctuation
8693               + parsers.unicode_following_whitespace
8694               + parsers.eof)
8695     end
8696
8697     parsers.right_flanking_delimiter_run = function(character)
8698     return parsers.unicode_preceding_punctuation
8699             * parsers.delimiter_run(character)
8700             * ( parsers.unicode_following_punctuation
8701               + parsers.unicode_following_whitespace
8702               + parsers.eof)
8703             + (B(parsers.any)
8704               * -( parsers.unicode_preceding_punctuation
8705                 + parsers.unicode_preceding_whitespace))
8706             * parsers.delimiter_run(character)
8707     end
8708
8709     if options.underscores then
8710     parsers.emph_start
8711     = parsers.left_flanking_delimiter_run(parsers.asterisk)
8712     + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
8713       + ( parsers.unicode_preceding_punctuation
8714         * #parsers.right_flanking_delimiter_run(parsers.underscore)))
8715     * parsers.left_flanking_delimiter_run(parsers.underscore)
8716
8717     parsers.emph_end
8718     = parsers.right_flanking_delimiter_run(parsers.asterisk)
8719     + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)

```

```

8720     + #( parsers.left_flanking_delimiter_run(parsers.underscore)
8721         * parsers.unicode_following_punctuation))
8722     * parsers.right_flanking_delimiter_run(parsers.underscore)
8723 else
8724     parsers.emph_start
8725     = parsers.left_flanking_delimiter_run(parsers.asterisk)
8726
8727     parsers.emph_end
8728     = parsers.right_flanking_delimiter_run(parsers.asterisk)
8729 end
8730
8731 parsers.emph_capturing_open_and_close
8732 = #parsers.emph_start * #parsers.emph_end
8733 * Ct( Cg(Cc("delimiter"), "type")
8734     * Cg(Cc("emphasis"), "element")
8735     * Cg(C(parsers.emph_start), "content")
8736     * Cg(Cc(true), "is_opening")
8737     * Cg(Cc(true), "is_closing"))
8738
8739 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
8740     * Cg(Cc("emphasis"), "element")
8741     * Cg(C(parsers.emph_start), "content")
8742     * Cg(Cc(true), "is_opening")
8743     * Cg(Cc(false), "is_closing"))
8744
8745 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
8746     * Cg(Cc("emphasis"), "element")
8747     * Cg(C(parsers.emph_end), "content")
8748     * Cg(Cc(false), "is_opening")
8749     * Cg(Cc(true), "is_closing"))
8750
8751 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8752     + parsers.emph_capturing_open
8753     + parsers.emph_capturing_close
8754
8755 parsers.emph_open = parsers.emph_capturing_open_and_close
8756     + parsers.emph_capturing_open
8757
8758 parsers.emph_close = parsers.emph_capturing_open_and_close
8759     + parsers.emph_capturing_close
8760

```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

8761 -- List of references defined in the document
8762 local references
8763

```

```

8764 -- List of note references defined in the document
8765 parsers.rawnotes = {}
8766

```

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.

```

8767 function self.register_link(_, tag, url, title,
8768                             attributes)
8769   local normalized_tag = self.normalize_tag(tag)
8770   if references[normalized_tag] == nil then
8771     references[normalized_tag] = {
8772       url = url,
8773       title = title,
8774       attributes = attributes
8775     }
8776   end
8777   return ""
8778 end
8779

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

8780 function self.lookup_reference(tag)
8781   return references[self.normalize_tag(tag)]
8782 end
8783

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

8784 function self.lookup_note_reference(tag)
8785   return parsers.rawnotes[self.normalize_tag(tag)]
8786 end
8787
8788 parsers.title_s_direct_ref = parsers.squote
8789                             * Cs((parsers.html_entities
8790                               + ( parsers.anyescaped
8791                                 - parsers.squote
8792                                   - parsers.blankline^2))^0)
8793                             * parsers.squote
8794
8795 parsers.title_d_direct_ref = parsers.dquote
8796                             * Cs((parsers.html_entities
8797                               + ( parsers.anyescaped
8798                                 - parsers.dquote
8799                                   - parsers.blankline^2))^0)
8800                             * parsers.dquote
8801
8802 parsers.title_p_direct_ref = parsers.lparent

```

```

8803         * Cs((parsers.html_entities
8804             + ( parsers.anyescaped
8805               - parsers.lparent
8806               - parsers.rparent
8807               - parsers.blankline^2))^0)
8808         * parsers.rparent
8809
8810 parsers.title_direct_ref = parsers.title_s_direct_ref
8811                          + parsers.title_d_direct_ref
8812                          + parsers.title_p_direct_ref
8813
8814 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8815                                   * Cg(parsers.url + Cc(""), "url")
8816                                   * parsers.spnl
8817                                   * Cg( parsers.title_direct_ref
8818                                       + Cc(""), "title")
8819                                   * parsers.spnl * parsers.rparent
8820
8821 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8822                                   * Cg(parsers.url + Cc(""), "url")
8823                                   * parsers.spnlc
8824                                   * Cg(parsers.title + Cc(""), "title")
8825                                   * parsers.spnlc * parsers.rparent
8826
8827 parsers.empty_link = parsers.lbracket
8828                   * parsers.rbracket
8829
8830 parsers.inline_link = parsers.link_text
8831                   * parsers.inline_direct_ref
8832
8833 parsers.full_link = parsers.link_text
8834                   * parsers.link_label
8835
8836 parsers.shortcut_link = parsers.link_label
8837                       * -(parsers.empty_link + parsers.link_label)
8838
8839 parsers.collapsed_link = parsers.link_label
8840                       * parsers.empty_link
8841
8842 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8843                       * Cg(Cc("inline"), "link_type")
8844                       + #(parsers.exclamation * parsers.full_link)
8845                       * Cg(Cc("full"), "link_type")
8846                       + #( parsers.exclamation
8847                           * parsers.collapsed_link)
8848                       * Cg(Cc("collapsed"), "link_type")
8849                       + #(parsers.exclamation * parsers.shortcut_link)

```



```

8850         * Cg(Cc("shortcut"), "link_type")
8851     + #(parsers.exclamation * parsers.empty_link)
8852     * Cg(Cc("empty"), "link_type")
8853
8854     parsers.link_opening = #parsers.inline_link
8855     * Cg(Cc("inline"), "link_type")
8856     + #parsers.full_link
8857     * Cg(Cc("full"), "link_type")
8858     + #parsers.collapsed_link
8859     * Cg(Cc("collapsed"), "link_type")
8860     + #parsers.shortcut_link
8861     * Cg(Cc("shortcut"), "link_type")
8862     + #parsers.empty_link
8863     * Cg(Cc("empty_link"), "link_type")
8864     + #parsers.link_text
8865     * Cg(Cc("link_text"), "link_type")
8866
8867     parsers.note_opening = #(parsers.circumflex * parsers.link_text)
8868     * Cg(Cc("note_inline"), "link_type")
8869
8870     parsers.raw_note_opening = #( parsers.lbracket
8871         * parsers.circumflex
8872         * parsers.link_label_body
8873         * parsers.rbracket)
8874     * Cg(Cc("raw_note"), "link_type")
8875
8876     local inline_note_element = Cg(Cc("note"), "element")
8877     * parsers.note_opening
8878     * Cg( parsers.circumflex
8879         * parsers.lbracket, "content")
8880
8881     local image_element = Cg(Cc("image"), "element")
8882     * parsers.image_opening
8883     * Cg( parsers.exclamation
8884         * parsers.lbracket, "content")
8885
8886     local note_element = Cg(Cc("note"), "element")
8887     * parsers.raw_note_opening
8888     * Cg( parsers.lbracket
8889         * parsers.circumflex, "content")
8890
8891     local link_element = Cg(Cc("link"), "element")
8892     * parsers.link_opening
8893     * Cg(parsers.lbracket, "content")
8894
8895     local opening_elements = parsers.fail
8896

```

```

8897 if options.inlineNotes then
8898     opening_elements = opening_elements + inline_note_element
8899 end
8900
8901 opening_elements = opening_elements + image_element
8902
8903 if options.notes then
8904     opening_elements = opening_elements + note_element
8905 end
8906
8907 opening_elements = opening_elements + link_element
8908
8909 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
8910     * Cg(Cc(true), "is_opening")
8911     * Cg(Cc(false), "is_closing")
8912     * opening_elements)
8913
8914 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
8915     * Cg(Cc("link"), "element")
8916     * Cg(Cc(false), "is_opening")
8917     * Cg(Cc(true), "is_closing")
8918     * ( Cg(Cc(true), "is_direct")
8919     * Cg( parsers.rbracket
8920         * #parsers.inline_direct_ref,
8921         "content")
8922     + Cg(Cc(false), "is_direct")
8923     * Cg(parsers.rbracket, "content")))
8924
8925 parsers.link_image_open_or_close = parsers.link_image_opening
8926     + parsers.link_image_closing
8927
8928 if options.html then
8929     parsers.link_emph_precedence = parsers.inticks
8930     + parsers.autolink
8931     + parsers.html_inline_tags
8932 else
8933     parsers.link_emph_precedence = parsers.inticks
8934     + parsers.autolink
8935 end
8936
8937 parsers.link_and_emph_endline = parsers.newline
8938     * ((parsers.check_minimal_indent
8939     * -V("EndlineExceptions")
8940     + parsers.check_optional_indent
8941     * -V("EndlineExceptions")
8942     * -parsers.starter) / "")
8943     * parsers.spacechar^0 / "\n"

```

```

8944
8945 parsers.link_and_emph_content
8946   = Ct( Cg(Cc("content"), "type")
8947     * Cg(Cs(( parsers.link_emph_precedence
8948         + parsers.backslash * parsers.linechar
8949         + parsers.link_and_emph_endline
8950         + (parsers.linechar
8951           - parsers.blankline^2
8952           - parsers.link_image_open_or_close
8953           - parsers.emph_open_or_close))^0), "content"))
8954
8955 parsers.link_and_emph_table
8956   = (parsers.link_image_opening + parsers.emph_open)
8957     * parsers.link_and_emph_content
8958     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
8959       * parsers.link_and_emph_content)^1
8960

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8961 local function collect_link_content(t, opening_index, closing_index)
8962   local content = {}
8963   for i = opening_index, closing_index do
8964     content[#content + 1] = t[i].content
8965   end
8966   return util.ropetostring(content)
8967 end
8968

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

8969 local function find_link_opener(t, bottom_index, latest_index)
8970   for i = latest_index, bottom_index, -1 do
8971     local value = t[i]
8972     if value.type == "delimiter" and
8973        value.is_opening and
8974        ( value.element == "link"
8975          or value.element == "image"
8976          or value.element == "note")
8977        and not value.removed then
8978       if value.is_active then
8979         return i
8980       end
8981       value.removed = true
8982       return nil
8983     end
8984   end
8985 end

```

8986

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
8987 local function find_next_link_closing_index(t, latest_index)
8988     for i = latest_index, #t do
8989         local value = t[i]
8990         if value.is_closing and
8991             value.element == "link" and
8992             not value.removed then
8993             return i
8994         end
8995     end
8996 end
8997
```

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.

```
8998 local function disable_previous_link_openers(t, opening_index)
8999     if t[opening_index].element == "image" then
9000         return
9001     end
9002
9003     for i = opening_index, 1, -1 do
9004         local value = t[i]
9005         if value.is_active and
9006             value.type == "delimiter" and
9007             value.is_opening and
9008             value.element == "link" then
9009             value.is_active = false
9010         end
9011     end
9012 end
9013
```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
9014 local function disable_range(t, opening_index, closing_index)
9015     for i = opening_index, closing_index do
9016         local value = t[i]
9017         if value.is_active then
9018             value.is_active = false
9019             if value.type == "delimiter" then
9020                 value.removed = true
9021             end
9022         end
9023     end
9024 end
```

9025

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
9026 local delete_parsed_content_in_range =
9027   function(t, opening_index, closing_index)
9028     for i = opening_index, closing_index do
9029       t[i].rendered = nil
9030     end
9031   end
9032
```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
9033 local function empty_content_in_range(t, opening_index, closing_index)
9034   for i = opening_index, closing_index do
9035     t[i].content = ''
9036   end
9037 end
9038
```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
9039 local function join_attributes(reference_attributes, own_attributes)
9040   local merged_attributes = {}
9041   for _, attribute in ipairs(reference_attributes or {}) do
9042     table.insert(merged_attributes, attribute)
9043   end
9044   for _, attribute in ipairs(own_attributes or {}) do
9045     table.insert(merged_attributes, attribute)
9046   end
9047   if next(merged_attributes) == nil then
9048     merged_attributes = nil
9049   end
9050   return merged_attributes
9051 end
9052
```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
9053 local render_link_or_image =
9054   function(t, opening_index, closing_index, content_end_index,
9055           reference)
9056     process_emphasis(t, opening_index, content_end_index)
9057     local mapped = collect_emphasis_content(t, opening_index + 1,
9058                                           content_end_index - 1)
9059
9060     local rendered = {}
```

```

9061     if (t[opening_index].element == "link") then
9062         rendered = writer.link(mapped, reference.url,
9063                               reference.title, reference.attributes)
9064     end
9065
9066     if (t[opening_index].element == "image") then
9067         rendered = writer.image(mapped, reference.url, reference.title,
9068                                reference.attributes)
9069     end
9070
9071     if (t[opening_index].element == "note") then
9072         if (t[opening_index].link_type == "note_inline") then
9073             rendered = writer.note(mapped)
9074         end
9075         if (t[opening_index].link_type == "raw_note") then
9076             rendered = writer.note(reference)
9077         end
9078     end
9079
9080     t[opening_index].rendered = rendered
9081     delete_parsed_content_in_range(t, opening_index + 1,
9082                                   closing_index)
9083     empty_content_in_range(t, opening_index, closing_index)
9084     disable_previous_link_openers(t, opening_index)
9085     disable_range(t, opening_index, closing_index)
9086 end
9087

```

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.

```

9088     local resolve_inline_following_content =
9089         function(t, closing_index, match_reference, match_link_attributes)
9090             local content = ""
9091             for i = closing_index + 1, #t do
9092                 content = content .. t[i].content
9093             end
9094
9095             local matching_content = parsers.succeed
9096
9097             if match_reference then
9098                 matching_content = matching_content
9099                     * parsers.inline_direct_ref_inside
9100             end
9101
9102             if match_link_attributes then
9103                 matching_content = matching_content

```

```

9104             * Cg(Ct(parsers.attributes^-1), "attributes")
9105     end
9106
9107     local matched = lpeg.match(Ct( matching_content
9108             * Cg(Cp(), "end_position")), content)
9109
9110     local matched_count = matched.end_position - 1
9111     for i = closing_index + 1, #t do
9112         local value = t[i]
9113
9114         local chars_left = matched_count
9115         matched_count = matched_count - #value.content
9116
9117         if matched_count <= 0 then
9118             value.content = value.content:sub(chars_left + 1)
9119             break
9120         end
9121
9122         value.content = ''
9123         value.is_active = false
9124     end
9125
9126     local attributes = matched.attributes
9127     if attributes == nil or next(attributes) == nil then
9128         attributes = nil
9129     end
9130
9131     return {
9132         url = matched.url or "",
9133         title = matched.title or "",
9134         attributes = attributes
9135     }
9136 end
9137

```

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.

```

9138     local function resolve_inline_link(t, opening_index, closing_index)
9139         local inline_content
9140             = resolve_inline_following_content(t, closing_index, true,
9141                 t.match_link_attributes)
9142         render_link_or_image(t, opening_index, closing_index,
9143             closing_index, inline_content)
9144     end
9145

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
9146 local resolve_note_inline_link =
9147   function(t, opening_index, closing_index)
9148     local inline_content
9149       = resolve_inline_following_content(t, closing_index,
9150                                         false, false)
9151     render_link_or_image(t, opening_index, closing_index,
9152                         closing_index, inline_content)
9153   end
9154
```

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.

```
9155 local function resolve_shortcut_link(t, opening_index, closing_index)
9156   local content
9157     = collect_link_content(t, opening_index + 1, closing_index - 1)
9158   local r = self.lookup_reference(content)
9159
9160   if r then
9161     local inline_content
9162       = resolve_inline_following_content(t, closing_index, false,
9163                                         t.match_link_attributes)
9164     r.attributes
9165       = join_attributes(r.attributes, inline_content.attributes)
9166     render_link_or_image(t, opening_index, closing_index,
9167                         closing_index, r)
9168   end
9169 end
9170
```

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.

```
9171 local function resolve_raw_note_link(t, opening_index, closing_index)
9172   local content
9173     = collect_link_content(t, opening_index + 1, closing_index - 1)
9174   local r = self.lookup_note_reference(content)
9175
9176   if r then
9177     local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9178     render_link_or_image(t, opening_index, closing_index,
9179                         closing_index, parsed_ref)
9180   end
9181 end
9182
```


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.

```
9183 local function resolve_full_link(t, opening_index, closing_index)
9184   local next_link_closing_index
9185     = find_next_link_closing_index(t, closing_index + 4)
9186   local next_link_content
9187     = collect_link_content(t, closing_index + 3,
9188                           next_link_closing_index - 1)
9189   local r = self.lookup_reference(next_link_content)
9190
9191   if r then
9192     local inline_content
9193       = resolve_inline_following_content(t, next_link_closing_index,
9194                                         false,
9195                                         t.match_link_attributes)
9196     r.attributes
9197       = join_attributes(r.attributes, inline_content.attributes)
9198     render_link_or_image(t, opening_index, next_link_closing_index,
9199                        closing_index, r)
9200   end
9201 end
9202
```

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.

```
9203 local function resolve_collapsed_link(t, opening_index, closing_index)
9204   local next_link_closing_index
9205     = find_next_link_closing_index(t, closing_index + 4)
9206   local content
9207     = collect_link_content(t, opening_index + 1, closing_index - 1)
9208   local r = self.lookup_reference(content)
9209
9210   if r then
9211     local inline_content
9212       = resolve_inline_following_content(t, closing_index, false,
9213                                         t.match_link_attributes)
9214     r.attributes
9215       = join_attributes(r.attributes, inline_content.attributes)
9216     render_link_or_image(t, opening_index, next_link_closing_index,
9217                        closing_index, r)
9218   end
9219 end
9220
```

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.

```
9221 local function process_links_and_emphasis(t)
9222   for _,value in ipairs(t) do
9223     value.is_active = true
9224   end
9225
9226   for i,value in ipairs(t) do
9227     if not value.is_closing
9228       or value.type ~= "delimiter"
9229       or not ( value.element == "link"
9230             or value.element == "image"
9231             or value.element == "note")
9232       or value.removed then
9233       goto continue
9234     end
9235
9236     local opener_position = find_link_opener(t, 1, i - 1)
9237     if (opener_position == nil) then
9238       goto continue
9239     end
9240
9241     local opening_delimiter = t[opener_position]
9242     opening_delimiter.removed = true
9243
9244     local link_type = opening_delimiter.link_type
9245
9246     if (link_type == "inline") then
9247       resolve_inline_link(t, opener_position, i)
9248     end
9249     if (link_type == "shortcut") then
9250       resolve_shortcut_link(t, opener_position, i)
9251     end
9252     if (link_type == "full") then
9253       resolve_full_link(t, opener_position, i)
9254     end
9255     if (link_type == "collapsed") then
9256       resolve_collapsed_link(t, opener_position, i)
9257     end
9258     if (link_type == "note_inline") then
9259       resolve_note_inline_link(t, opener_position, i)
9260     end
9261     if (link_type == "raw_note") then
9262       resolve_raw_note_link(t, opener_position, i)
9263     end
9264
9265     ::continue::
```

```

9266     end
9267
9268     t[#t].content = t[#t].content:gsub("%s*$","")
9269
9270     process_emphasis(t, 1, #t)
9271     local final_result = collect_emphasis_content(t, 1, #t)
9272     return final_result
9273 end
9274
9275 function self.defer_link_and_emphasis_processing(delimiter_table)
9276     return writer.defer_call(function()
9277         return process_links_and_emphasis(delimiter_table)
9278     end)
9279 end
9280

```

3.1.6.8 Inline Elements (local)

```

9281 parsers.Str      = ( parsers.normalchar
9282                   * (parsers.normalchar + parsers.at)^0)
9283                   / writer.string
9284
9285 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
9286                   / writer.string
9287
9288 parsers.Ellipsis = P("...") / writer.ellipsis
9289
9290 parsers.Smart    = parsers.Ellipsis
9291
9292 parsers.Code     = parsers.inticks / writer.code
9293
9294 if options.blankBeforeBlockquote then
9295     parsers.bqstart = parsers.fail
9296 else
9297     parsers.bqstart = parsers.blockquote_start
9298 end
9299
9300 if options.blankBeforeHeading then
9301     parsers.headerstart = parsers.fail
9302 else
9303     parsers.headerstart = parsers.atx_heading
9304 end
9305
9306 if options.blankBeforeList then
9307     parsers.interrupting_bullets = parsers.fail
9308     parsers.interrupting_enumerators = parsers.fail
9309 else

```

```

9310     parsers.interrupting_bullets
9311         = parsers.bullet(parsers.dash, true)
9312         + parsers.bullet(parsers.asterisk, true)
9313         + parsers.bullet(parsers.plus, true)
9314
9315     parsers.interrupting_enumerators
9316         = parsers.enumerator(parsers.period, true)
9317         + parsers.enumerator(parsers.rparent, true)
9318     end
9319
9320     if options.html then
9321         parsers.html_interrupting
9322             = parsers.check_trail
9323             * ( parsers.html_incomplete_open_tag
9324                 + parsers.html_incomplete_close_tag
9325                 + parsers.html_incomplete_open_special_tag
9326                 + parsers.html_comment_start
9327                 + parsers.html_cdatasection_start
9328                 + parsers.html_declaration_start
9329                 + parsers.html_instruction_start
9330                 - parsers.html_close_special_tag
9331                 - parsers.html_empty_special_tag)
9332     else
9333         parsers.html_interrupting = parsers.fail
9334     end
9335
9336     parsers.EndlineExceptions
9337         = parsers.blankline -- paragraph break
9338         + parsers.eof      -- end of document
9339         + parsers.bqstart
9340         + parsers.thematic_break_lines
9341         + parsers.interrupting_bullets
9342         + parsers.interrupting_enumerators
9343         + parsers.headerstart
9344         + parsers.html_interrupting
9345
9346     parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9347
9348     parsers.endline = parsers.newline
9349         * (parsers.check_minimal_indent
9350             * -V("EndlineExceptions")
9351             + parsers.check_optional_indent
9352             * -V("EndlineExceptions")
9353             * -parsers.starter) / function(_) return end
9354         * parsers.spacechar^0
9355
9356     parsers.Endline = parsers.endline

```

```

9357             / writer.soft_line_break
9358
9359 parsers.EndlineNoSub = parsers.endline
9360
9361 parsers.NoSoftLineBreakEndline
9362             = parsers.newline
9363             * (parsers.check_minimal_indent
9364             * -V("NoSoftLineBreakEndlineExceptions")
9365             + parsers.check_optional_indent
9366             * -V("NoSoftLineBreakEndlineExceptions")
9367             * -parsers.starter)
9368             * parsers.spacechar^0
9369             / writer.space
9370
9371 parsers.EndlineBreak = parsers.backslash * parsers.Endline
9372                       / writer.hard_line_break
9373
9374 parsers.OptionalIndent
9375             = parsers.spacechar^1 / writer.space
9376
9377 parsers.Space     = parsers.spacechar^2 * parsers.Endline
9378                   / writer.hard_line_break
9379                   + parsers.spacechar^1
9380                   * parsers.Endline^-1
9381                   * parsers.eof / self.expandtabs
9382                   + parsers.spacechar^1 * parsers.Endline
9383                   / writer.soft_line_break
9384                   + parsers.spacechar^1
9385                   * -parsers.newline / self.expandtabs
9386
9387 parsers.NoSoftLineBreakSpace
9388             = parsers.spacechar^2 * parsers.Endline
9389             / writer.hard_line_break
9390             + parsers.spacechar^1
9391             * parsers.Endline^-1
9392             * parsers.eof / self.expandtabs
9393             + parsers.spacechar^1 * parsers.Endline
9394             / writer.soft_line_break
9395             + parsers.spacechar^1
9396             * -parsers.newline / self.expandtabs
9397
9398 parsers.NonbreakingEndline
9399             = parsers.endline
9400             / writer.nbsp
9401
9402 parsers.NonbreakingSpace
9403             = parsers.spacechar^2 * parsers.endline

```

```

9404                                     / writer.nbsp
9405     + parsers.spacechar^1
9406     * parsers.endline^-1 * parsers.eof / ""
9407     + parsers.spacechar^1 * parsers.endline
9408                                     * parsers.optionalspace
9409                                     / writer.nbsp
9410     + parsers.spacechar^1 * parsers.optionalspace
9411                                     / writer.nbsp
9412

```

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.

```

9413 function self.auto_link_url(url, attributes)
9414   return writer.link(writer.escape(url),
9415                     url, nil, attributes)
9416 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.

```

9417 function self.auto_link_email(email, attributes)
9418   return writer.link(writer.escape(email),
9419                     "mailto:".email,
9420                     nil, attributes)
9421 end
9422
9423 parsers.AutoLinkUrl = parsers.auto_link_url
9424                     / self.auto_link_url
9425
9426 parsers.AutoLinkEmail
9427                     = parsers.auto_link_email
9428                     / self.auto_link_email
9429
9430 parsers.AutoLinkRelativeReference
9431                     = parsers.auto_link_relative_reference
9432                     / self.auto_link_url
9433
9434 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9435                     / self.defer_link_and_emphasis_processing
9436
9437 parsers.EscapedChar = parsers.backslash
9438                     * C(parsers.escapable) / writer.string
9439
9440 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9441                     / writer.inline_html_comment
9442                     + Cs(parsers.html_any_empty_inline_tag

```

```

9443         + parsers.html_inline_instruction
9444         + parsers.html_inline_cdatasection
9445         + parsers.html_inline_declaration
9446         + parsers.html_any_open_inline_tag
9447         + parsers.html_any_close_tag)
9448     / writer.inline_html_tag
9449
9450     parsers.HtmlEntity = parsers.html_entities / writer.string

```

3.1.6.9 Block Elements (local)

```

9451     parsers.DisplayHtml = Cs(parsers.check_trail
9452         * ( parsers.html_comment
9453           + parsers.html_special_block
9454           + parsers.html_block
9455           + parsers.html_any_block
9456           + parsers.html_instruction
9457           + parsers.html_cdatasection
9458           + parsers.html_declaration))
9459     / writer.block_html_element
9460
9461     parsers.indented_non_blank_line = parsers.indentedline
9462         - parsers.blankline
9463
9464     parsers.Verbatim
9465     = Cs( parsers.check_code_trail
9466         * (parsers.line - parsers.blankline)
9467         * (( parsers.check_minimal_blank_indent_and_full_code_trail
9468           * parsers.blankline)^0
9469         * ( (parsers.check_minimal_indent / "")
9470           * parsers.check_code_trail
9471           * (parsers.line - parsers.blankline))^1)^0)
9472     / self.expandtabs / writer.verbatim
9473
9474     parsers.Blockquote = parsers.blockquote_body
9475         / writer.blockquote
9476
9477     parsers.ThematicBreak = parsers.thematic_break_lines
9478         / writer.thematic_break
9479
9480     parsers.Reference = parsers.define_reference_parser
9481         / self.register_link
9482
9483     parsers.Paragraph = parsers.freeze_trail
9484         * (Ct((parsers.Inline)^1)
9485         * (parsers.newline + parsers.eof)
9486         * parsers.unfreeze_trail

```

```

9487             / writer.paragraph)
9488
9489 parsers.Plain      = parsers.nonindentospace * Ct(parsers.Inline~1)
9490             / writer.plain

```

3.1.6.10 Lists (local)

```

9491
9492 if options.taskLists then
9493   parsers.tickbox = ( parsers.ticked_box
9494                     + parsers.halfticked_box
9495                     + parsers.unticked_box
9496                     ) / writer.tickbox
9497 else
9498   parsers.tickbox = parsers.fail
9499 end
9500
9501 parsers.list_blank = parsers.conditionally_indented_blankline
9502
9503 parsers.ref_or_block_list_separated
9504   = parsers.sep_group_no_output(parsers.list_blank)
9505   * parsers.minimally_indented_ref
9506   + parsers.block_sep_group(parsers.list_blank)
9507   * parsers.minimally_indented_block
9508
9509 parsers.ref_or_block_non_separated
9510   = parsers.minimally_indented_ref
9511   + (parsers.succeed / writer.interblocksep)
9512   * parsers.minimally_indented_block
9513   - parsers.minimally_indented_blankline
9514
9515 parsers.tight_list_loop_body_pair =
9516   parsers.create_loop_body_pair(
9517     parsers.ref_or_block_non_separated,
9518     parsers.minimally_indented_par_or_plain_no_blank,
9519     (parsers.succeed / writer.interblocksep),
9520     (parsers.succeed / writer.paragraphsep))
9521
9522 parsers.loose_list_loop_body_pair =
9523   parsers.create_loop_body_pair(
9524     parsers.ref_or_block_list_separated,
9525     parsers.minimally_indented_par_or_plain,
9526     parsers.block_sep_group(parsers.list_blank),
9527     parsers.par_sep_group(parsers.list_blank))
9528
9529 parsers.tight_list_content_loop
9530   = V("Block")

```



```

9531     * parsers.tight_list_loop_body_pair.block^0
9532     + (V("Paragraph") + V("Plain"))
9533     * parsers.ref_or_block_non_separated
9534     * parsers.tight_list_loop_body_pair.block^0
9535     + (V("Paragraph") + V("Plain"))
9536     * parsers.tight_list_loop_body_pair.par^0
9537
9538     parsers.loose_list_content_loop
9539     = V("Block")
9540     * parsers.loose_list_loop_body_pair.block^0
9541     + (V("Paragraph") + V("Plain"))
9542     * parsers.ref_or_block_list_separated
9543     * parsers.loose_list_loop_body_pair.block^0
9544     + (V("Paragraph") + V("Plain"))
9545     * parsers.loose_list_loop_body_pair.par^0
9546
9547     parsers.list_item_tightness_condition
9548     = -( parsers.list_blank^0
9549         * parsers.minimally_indented_ref_or_block_or_par)
9550     * remove_indent("li")
9551     + remove_indent("li")
9552     * parsers.fail
9553
9554     parsers.indented_content_tight
9555     = Ct( (parsers.blankline / "")
9556         * #parsers.list_blank
9557         * remove_indent("li")
9558         + ( (V("Reference") + (parsers.blankline / ""))
9559             * parsers.check_minimal_indent
9560             * parsers.tight_list_content_loop
9561             + (V("Reference") + (parsers.blankline / ""))
9562             + (parsers.checkbox^-1 / writer.escape)
9563             * parsers.tight_list_content_loop
9564             )
9565         * parsers.list_item_tightness_condition)
9566
9567     parsers.indented_content_loose
9568     = Ct( (parsers.blankline / "")
9569         * #parsers.list_blank
9570         + ( (V("Reference") + (parsers.blankline / ""))
9571             * parsers.check_minimal_indent
9572             * parsers.loose_list_content_loop
9573             + (V("Reference") + (parsers.blankline / ""))
9574             + (parsers.checkbox^-1 / writer.escape)
9575             * parsers.loose_list_content_loop))
9576
9577     parsers.TightListItem = function(starter)

```

```

9578     return -parsers.ThematicBreak
9579         * parsers.add_indent(starter, "li")
9580         * parsers.indented_content_tight
9581     end
9582
9583     parsers.LooseListItem = function(starter)
9584         return -parsers.ThematicBreak
9585             * parsers.add_indent(starter, "li")
9586             * parsers.indented_content_loose
9587             * remove_indent("li")
9588     end
9589
9590     parsers.BulletListOfType = function(bullet_type)
9591         local bullet = parsers.bullet(bullet_type)
9592         return ( Ct( parsers.TightListItem(bullet)
9593             * ( (parsers.check_minimal_indent / "")
9594                 * parsers.TightListItem(bullet)
9595             )^0
9596             )
9597             * Cc(true)
9598             * -#( (parsers.list_blank^0 / "")
9599                 * parsers.check_minimal_indent
9600                 * (bullet - parsers.ThematicBreak)
9601             )
9602             + Ct( parsers.LooseListItem(bullet)
9603                 * ( (parsers.list_blank^0 / "")
9604                     * (parsers.check_minimal_indent / "")
9605                     * parsers.LooseListItem(bullet)
9606                 )^0
9607             )
9608             * Cc(false)
9609             ) / writer.bulletlist
9610     end
9611
9612     parsers.BulletList = parsers.BulletListOfType(parsers.dash)
9613         + parsers.BulletListOfType(parsers.asterisk)
9614         + parsers.BulletListOfType(parsers.plus)
9615
9616     local function ordered_list(items,tight,starter)
9617         local startnum = starter[2][1]
9618         if options.startNumber then
9619             startnum = tonumber(startnum) or 1 -- fallback for '#'
9620             if startnum ~= nil then
9621                 startnum = math.floor(startnum)
9622             end
9623         else
9624             startnum = nil

```

```

9625     end
9626     return writer.orderedlist(items,tight,startnum)
9627 end
9628
9629 parsers.OrderedListOfType = function(delimiter_type)
9630     local enumerator = parsers.enumerator(delimiter_type)
9631     return Cg(enumerator, "listtype")
9632         * (Ct( parsers.TightListItem(Cb("listtype"))
9633             * ( (parsers.check_minimal_indent / "")
9634                 * parsers.TightListItem(enumerator))^0)
9635         * Cc(true)
9636         * -#((parsers.list_blank^0 / "")
9637             * parsers.check_minimal_indent * enumerator)
9638     + Ct( parsers.LooseListItem(Cb("listtype"))
9639         * ((parsers.list_blank^0 / "")
9640             * (parsers.check_minimal_indent / "")
9641             * parsers.LooseListItem(enumerator))^0)
9642         * Cc(false)
9643     ) * Ct(Cb("listtype")) / ordered_list
9644 end
9645
9646 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
9647     + parsers.OrderedListOfType(parsers.rparent)

```

3.1.6.11 Blank (local)

```

9648 parsers.Blank           = parsers.blankline / ""
9649     + V("Reference")

```

3.1.6.12 Headings (local)

```

9650 function parsers.parse_heading_text(s)
9651     local inlines = self.parser_functions.parse_inlines(s)
9652     local flatten_inlines = self.writer.flatten_inlines
9653     self.writer.flatten_inlines = true
9654     local flat_text = self.parser_functions.parse_inlines(s)
9655     flat_text = util.rope_to_string(flat_text)
9656     self.writer.flatten_inlines = flatten_inlines
9657     return {flat_text, inlines}
9658 end
9659
9660 -- parse atx header
9661 parsers.AtxHeading = parsers.check_trail_no_rem
9662     * Cg(parsers.heading_start, "level")
9663     * ((C( parsers.optionalspace
9664         * parsers.hash^0
9665         * parsers.optionalspace
9666         * parsers.newline)

```

```

9667         + parsers.spacechar^1
9668         * C(parsers.line))
9669         / strip_atx_end
9670         / parsers.parse_heading_text)
9671         * Cb("level")
9672         / writer.heading
9673
9674 parsers.heading_line = parsers.linechar^1
9675                       - parsers.thematic_break_lines
9676
9677 parsers.heading_text = parsers.heading_line
9678                       * ( (V("Endline") / "\n")
9679                           * ( parsers.heading_line
9680                               - parsers.heading_level))^0
9681                       * parsers.newline^-1
9682
9683 parsers.SetextHeading = parsers.freeze_trail
9684                       * parsers.check_trail_no_rem
9685                       * #( parsers.heading_text
9686                           * parsers.check_minimal_indent
9687                           * parsers.check_trail
9688                           * parsers.heading_level)
9689                       * Cs(parsers.heading_text)
9690                       / parsers.parse_heading_text
9691                       * parsers.check_minimal_indent_and_trail
9692                       * parsers.heading_level
9693                       * parsers.newline
9694                       * parsers.unfreeze_trail
9695                       / writer.heading
9696
9697 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain $\text{T}_{\text{E}}\text{X}$ output.

```

9698 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.

```

9699     local walkable_syntax = (function(global_walkable_syntax)
9700         local local_walkable_syntax = {}
9701         for lhs, rule in pairs(global_walkable_syntax) do

```

```

9702     local_walkable_syntax[lhs] = util.table_copy(rule)
9703     end
9704     return local_walkable_syntax
9705 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.

```

9706     local current_extension_name = nil
9707     self.insert_pattern = function(selector, pattern, pattern_name)
9708         assert(pattern_name == nil or type(pattern_name) == "string")
9709         local _, _, lhs, pos, rhs
9710             = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9711         assert(lhs ~= nil,
9712             [[Expected selector in form ]]
9713             .. [[LHS (before|after|instead of) RHS", not "]]
9714             .. selector .. [[]])
9715         assert(walkable_syntax[lhs] ~= nil,
9716             [[Rule ]] .. lhs
9717             .. [[ -> ... does not exist in markdown grammar]])
9718         assert(pos == "before" or pos == "after" or pos == "instead of",
9719             [[Expected positional specifier "before", "after", ]]
9720             .. [[or "instead of", not "]]
9721             .. pos .. [[]])
9722         local rule = walkable_syntax[lhs]
9723         local index = nil
9724         for current_index, current_rhs in ipairs(rule) do
9725             if type(current_rhs) == "string" and current_rhs == rhs then
9726                 index = current_index
9727                 if pos == "after" then
9728                     index = index + 1
9729                 end
9730                 break
9731             end
9732         end
9733         assert(index ~= nil,
9734             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
9735             .. [[ does not exist in markdown grammar]])
9736         local accountable_pattern
9737         if current_extension_name then
9738             accountable_pattern
9739                 = {pattern, current_extension_name, pattern_name}
9740         else
9741             assert(type(pattern) == "string",
9742                 [[reader->insert_pattern() was called outside ]]
9743                 .. [[an extension with ]]
9744                 .. [[a PEG pattern instead of a rule name]])

```

```

9745     accountable_pattern = pattern
9746   end
9747   if pos == "instead of" then
9748     rule[index] = accountable_pattern
9749   else
9750     table.insert(rule, index, accountable_pattern)
9751   end
9752 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.

```

9753   local syntax =
9754     { "Blocks",
9755
9756       Blocks = V("InitializeState")
9757         * V("ExpectedJekyllData")
9758         * 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.

```

9759     * ( V("Block")
9760       * ( V("Blank")^0 * parsers.eof
9761         + ( V("Blank")^2 / writer.paragraphsep
9762           + V("Blank")^0 / writer.interblocksep
9763         )
9764       )
9765     + ( V("Paragraph") + V("Plain") )
9766     * ( V("Blank")^0 * parsers.eof
9767       + ( V("Blank")^2 / writer.paragraphsep
9768         + V("Blank")^0 / writer.interblocksep
9769       )
9770     )
9771     * V("Block")
9772     * ( V("Blank")^0 * parsers.eof
9773       + ( V("Blank")^2 / writer.paragraphsep
9774         + V("Blank")^0 / writer.interblocksep
9775       )
9776     )
9777     + ( V("Paragraph") + V("Plain") )
9778     * ( V("Blank")^0 * parsers.eof
9779       + V("Blank")^0 / writer.paragraphsep
9780     )
9781     )^0,
9782
9783     ExpectedJekyllData = parsers.succeed,
9784
9785     Blank = parsers.Blank,

```

```

9786     Reference           = parsers.Reference,
9787
9788     Blockquote          = parsers.Blockquote,
9789     Verbatim            = parsers.Verbatim,
9790     ThematicBreak       = parsers.ThematicBreak,
9791     BulletList          = parsers.BulletList,
9792     OrderedList         = parsers.OrderedList,
9793     DisplayHtml         = parsers.DisplayHtml,
9794     Heading             = parsers.Heading,
9795     Paragraph           = parsers.Paragraph,
9796     Plain               = parsers.Plain,
9797
9798     EndlineExceptions   = parsers.EndlineExceptions,
9799     NoSoftLineBreakEndlineExceptions
9800                         = parsers.NoSoftLineBreakEndlineExceptions,
9801
9802     Str                 = parsers.Str,
9803     Space               = parsers.Space,
9804     NoSoftLineBreakSpace
9805                         = parsers.NoSoftLineBreakSpace,
9806     OptionalIndent     = parsers.OptionalIndent,
9807     Endline             = parsers.Endline,
9808     EndlineNoSub       = parsers.EndlineNoSub,
9809     NoSoftLineBreakEndline
9810                         = parsers.NoSoftLineBreakEndline,
9811     EndlineBreak       = parsers.EndlineBreak,
9812     LinkAndEmph        = parsers.LinkAndEmph,
9813     Code               = parsers.Code,
9814     AutoLinkUrl        = parsers.AutoLinkUrl,
9815     AutoLinkEmail      = parsers.AutoLinkEmail,
9816     AutoLinkRelativeReference
9817                         = parsers.AutoLinkRelativeReference,
9818     InlineHtml         = parsers.InlineHtml,
9819     HtmlEntity         = parsers.HtmlEntity,
9820     EscapedChar        = parsers.EscapedChar,
9821     Smart              = parsers.Smart,
9822     Symbol             = parsers.Symbol,
9823     SpecialChar        = parsers.fail,
9824     InitializeState    = parsers.succeed,
9825 }

```

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]`.

```

9826     self.update_rule = function(rule_name, get_pattern)

```

```

9827     assert(current_extension_name ~= nil)
9828     assert(syntax[rule_name] ~= nil,
9829         [[Rule ]] .. rule_name
9830         .. [[ -> ... does not exist in markdown grammar]])
9831     local previous_pattern
9832     local extension_name
9833     if walkable_syntax[rule_name] then
9834         local previous_accountable_pattern
9835         = walkable_syntax[rule_name][1]
9836         previous_pattern = previous_accountable_pattern[1]
9837         extension_name
9838         = previous_accountable_pattern[2]
9839         .. ", " .. current_extension_name
9840     else
9841         previous_pattern = nil
9842         extension_name = current_extension_name
9843     end
9844     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

```

```

9845     if type(get_pattern) == "function" then
9846         pattern = get_pattern(previous_pattern)
9847     else
9848         assert(previous_pattern == nil,
9849             [[Rule ]] .. rule_name ..
9850             [[ has already been updated by ]] .. extension_name)
9851         pattern = get_pattern
9852     end
9853     local accountable_pattern = { pattern, extension_name, rule_name }
9854     walkable_syntax[rule_name] = { accountable_pattern }
9855 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.

```

9856     local special_characters = {}
9857     self.add_special_character = function(c)

```



```

9858     table.insert(special_characters, c)
9859     syntax.SpecialChar = S(table.concat(special_characters, ""))
9860 end
9861
9862 self.add_special_character("*")
9863 self.add_special_character("[")
9864 self.add_special_character("]")
9865 self.add_special_character("<")
9866 self.add_special_character("!")
9867 self.add_special_character("\\")

```

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

```

9868 self.initialize_named_group = function(name, value)
9869     local pattern = Ct("")
9870     if value ~= nil then
9871         pattern = pattern / value
9872     end
9873     syntax.InitializeState = syntax.InitializeState
9874         * Cg(pattern, name)
9875 end

```

Add a named group for indentation.

```

9876 self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

9877 for _, extension in ipairs(extensions) do
9878     current_extension_name = extension.name
9879     extension.extend_writer(writer)
9880     extension.extend_reader(self)
9881 end
9882 current_extension_name = nil

```

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

```

9883 if options.debugExtensions then
9884     local sorted_lhs = {}
9885     for lhs, _ in pairs(walkable_syntax) do
9886         table.insert(sorted_lhs, lhs)
9887     end
9888     table.sort(sorted_lhs)
9889
9890     local output_lines = {"{"}
9891     for lhs_index, lhs in ipairs(sorted_lhs) do
9892         local encoded_lhs = util.encode_json_string(lhs)
9893         table.insert(output_lines, [[    ]] .. encoded_lhs .. [[: ]])
9894         local rule = walkable_syntax[lhs]
9895         for rhs_index, rhs in ipairs(rule) do

```

```

9896     local human_readable_rhs
9897     if type(rhs) == "string" then
9898         human_readable_rhs = rhs
9899     else
9900         local pattern_name
9901         if rhs[3] then
9902             pattern_name = rhs[3]
9903         else
9904             pattern_name = "Anonymous Pattern"
9905         end
9906         local extension_name = rhs[2]
9907         human_readable_rhs = pattern_name .. [[ (]]
9908                               .. extension_name .. [[]]]
9909     end
9910     local encoded_rhs
9911         = util.encode_json_string(human_readable_rhs)
9912     local output_line = [[          ]] .. encoded_rhs
9913     if rhs_index < #rule then
9914         output_line = output_line .. ","
9915     end
9916     table.insert(output_lines, output_line)
9917 end
9918 local output_line = "    ]"
9919 if lhs_index < #sorted_lhs then
9920     output_line = output_line .. ","
9921 end
9922 table.insert(output_lines, output_line)
9923 end
9924 table.insert(output_lines, "}")
9925
9926 local output = table.concat(output_lines, "\n")
9927 local output_filename = options.debugExtensionsFileName
9928 local output_file = assert(io.open(output_filename, "w"),
9929     [[Could not open file ]] .. output_filename
9930     .. [[ for writing]])
9931 assert(output_file:write(output))
9932 assert(output_file:close())
9933 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`.

```

9934     for lhs, rule in pairs(walkable_syntax) do
9935         syntax[lhs] = parsers.fail
9936         for _, rhs in ipairs(rule) do
9937             local pattern

```

Although the interface of the `reader->insert_pattern` method does not doc-

ument 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.

```
9938         if type(rhs) == "string" then
9939             pattern = V(rhs)
9940         else
9941             pattern = rhs[1]
9942             if type(pattern) == "string" then
9943                 pattern = V(pattern)
9944             end
9945         end
9946         syntax[lhs] = syntax[lhs] + pattern
9947     end
9948 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.

```
9949     if options.underscores then
9950         self.add_special_character("_")
9951     end
9952
9953     if not options.codeSpans then
9954         syntax.Code = parsers.fail
9955     else
9956         self.add_special_character("`")
9957     end
9958
9959     if not options.html then
9960         syntax.DisplayHtml = parsers.fail
9961         syntax.InlineHtml = parsers.fail
9962         syntax.HtmlEntity = parsers.fail
9963     else
9964         self.add_special_character("&")
9965     end
9966
9967     if options.preserveTabs then
9968         options.stripIndent = false
9969     end
9970
9971     if not options.smartEllipses then
9972         syntax.Smart = parsers.fail
9973     else
9974         self.add_special_character(".")
9975     end
9976
9977     if not options.relativeReferences then
```

```

9978     syntax.AutoLinkRelativeReference = parsers.fail
9979 end
9980
9981 if options.contentLevel == "inline" then
9982     syntax[1] = "Inlines"
9983     syntax.Inlines = V("InitializeState")
9984         * parsers.Inline^0
9985         * ( parsers.spacing^0
9986             * parsers.eof / "" )
9987     syntax.Space = parsers.Space + parsers.blankline / writer.space
9988 end
9989
9990 local blocks_nested_t = util.table_copy(syntax)
9991 blocks_nested_t.ExpectedJekyllData = parsers.succeed
9992 parsers.blocks_nested = Ct(blocks_nested_t)
9993
9994 parsers.blocks = Ct(syntax)
9995
9996 local inlines_t = util.table_copy(syntax)
9997 inlines_t[1] = "Inlines"
9998 inlines_t.Inlines = V("InitializeState")
9999     * parsers.Inline^0
10000     * ( parsers.spacing^0
10001         * parsers.eof / "" )
10002 parsers.inlines = Ct(inlines_t)
10003
10004 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10005 inlines_no_inline_note_t.InlineNote = parsers.fail
10006 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10007
10008 local inlines_no_html_t = util.table_copy(inlines_t)
10009 inlines_no_html_t.DisplayHtml = parsers.fail
10010 inlines_no_html_t.InlineHtml = parsers.fail
10011 inlines_no_html_t.HtmlEntity = parsers.fail
10012 parsers.inlines_no_html = Ct(inlines_no_html_t)
10013
10014 local inlines_nbsp_t = util.table_copy(inlines_t)
10015 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10016 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10017 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10018
10019 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10020 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10021 inlines_no_link_or_emphasis_t.EndlineExceptions
10022     = parsers.EndlineExceptions - parsers.eof
10023 parsers.inlines_no_link_or_emphasis
10024     = Ct(inlines_no_link_or_emphasis_t)

```

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

```
10025     return function(input)
Unicode-normalize the input.
10026     if options.unicodeNormalization then
10027         local form = options.unicodeNormalizationForm
10028         if form == "nfc" then
10029             input = uni_algos.normalize.NFC(input)
10030         elseif form == "nfd" then
10031             input = uni_algos.normalize.NFD(input)
10032         elseif form == "nfkc" then
10033             input = uni_algos.normalize.NFKC(input)
10034         elseif form == "nfkd" then
10035             input = uni_algos.normalize.NFKD(input)
10036         else
10037             return writer.error(
10038                 format("Unknown normalization form %s.", form))
10039         end
10040     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.

```
10041     input = input:gsub("\r\n?", "\n")
10042     if input:sub(-1) ~= "\n" then
10043         input = input .. "\n"
10044     end
```

Clear the table of references.

```
10045     references = {}
10046     local document = self.parser_functions.parse_blocks(input)
10047     local output = util.ropetostring(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
10048     local undosep_start, undosep_end
10049     local potential_secend_start, secend_start
10050     local potential_sep_start, sep_start
10051     while true do
10052         -- find a `writer->undosep`
10053         undosep_start, undosep_end
10054             = output:find(writer.undosep_text, 1, true)
10055         if undosep_start == nil then break end
10056         -- skip any preceding section ends
10057         secend_start = undosep_start
10058         while true do
10059             potential_secend_start = secend_start - #writer.secend_text
```

```

10060         if potential_secend_start < 1
10061             or output:sub(potential_secend_start,
10062                          secend_start - 1) ~= writer.secend_text
10063         then
10064             break
10065         end
10066         secend_start = potential_secend_start
10067     end
10068     -- find an immediately preceding
10069     -- block element / paragraph separator
10070     sep_start = secend_start
10071     potential_sep_start = sep_start - #writer.interblocksep_text
10072     if potential_sep_start >= 1
10073         and output:sub(potential_sep_start,
10074                      sep_start - 1) == writer.interblocksep_text
10075     then
10076         sep_start = potential_sep_start
10077     else
10078         potential_sep_start = sep_start - #writer.paragraphsep_text
10079         if potential_sep_start >= 1
10080             and output:sub(potential_sep_start,
10081                          sep_start - 1) == writer.paragraphsep_text
10082         then
10083             sep_start = potential_sep_start
10084         end
10085     end
10086     -- remove `writer->undosep` and immediately preceding
10087     -- block element / paragraph separator
10088     output = output:sub(1, sep_start - 1)
10089         .. output:sub(secend_start, undosep_start - 1)
10090         .. output:sub(undosep_end + 1)
10091     end
10092     return output
10093 end
10094 end
10095 return self
10096 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.

```
10097 M.extensions = {}
```

3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
10098 M.extensions.bracketed_spans = function()
10099   return {
10100     name = "built-in bracketed_spans syntax extension",
10101     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.

```
10102     function self.span(s, attr)
10103       if self.flatten_inlines then return s end
10104       return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10105             self.attributes(attr),
10106             s,
10107             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10108     end
10109   end, extend_reader = function(self)
10110     local parsers = self.parsers
10111     local writer = self.writer
10112
10113     local span_label = parsers.lbracket
10114                       * (Cs((parsers.alphanumeric^1
10115                             + parsers.inticks
10116                             + parsers.autolink
10117                             + V("InlineHtml")
10118                             + ( parsers.backslash * parsers.backslash)
10119                             + ( parsers.backslash
10120                               * (parsers.lbracket + parsers.rbracket)
10121                               + V("Space") + V("Endline")
10122                               + (parsers.any
10123                                 - ( parsers.newline
10124                                   + parsers.lbracket
10125                                   + parsers.rbracket
10126                                   + parsers.blankline^2))))^1)
10127                       / self.parser_functions.parse_inlines)
10128                       * parsers.rbracket
10129
10130     local Span = span_label
10131                 * Ct(parsers.attributes)
10132                 / writer.span
10133
10134     self.insert_pattern("Inline before LinkAndEmph",
10135                       Span, "Span")
10136   end
10137 }
10138 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.

```
10139 M.extensions.citations = function(citation_nbsps)
10140   return {
10141     name = "built-in citations syntax extension",
10142     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.

```
10143     function self.citations(text_cites, cites)
10144       local buffer = {}
10145       if self.flatten_inlines then
10146         for _,cite in ipairs(cites) do
10147           if cite.prenote then
10148             table.insert(buffer, {cite.prenote, " "})
10149           end
10150           table.insert(buffer, cite.name)
10151           if cite.postnote then
10152             table.insert(buffer, {" ", cite.postnote})
10153           end
10154         end
10155       else
10156         table.insert(buffer,
10157           {"\\markdownRenderer",
10158             text_cites and "TextCite" or "Cite",
10159             "{", #cites, "}"})
10160         for _,cite in ipairs(cites) do
10161           table.insert(buffer,
10162             {cite.suppress_author and "-" or "+", "{",
10163               cite.prenote or "", "}{",
```



```

10164             cite.postnote or "", "}{", cite.name, "}")
10165         end
10166     end
10167     return buffer
10168 end
10169 end, extend_reader = function(self)
10170     local parsers = self.parsers
10171     local writer = self.writer
10172
10173     local citation_chars
10174         = parsers.alphanumeric
10175         + S("#$%&-+<>-/_")
10176
10177     local citation_name
10178         = Cs(parsers.dash^-1) * parsers.at
10179         * Cs(citation_chars
10180             * ((( citation_chars
10181                 + parsers.internal_punctuation
10182                 - parsers.comma - parsers.semicolon)
10183                 * -#(( parsers.internal_punctuation
10184                     - parsers.comma
10185                     - parsers.semicolon)^0
10186                 * -( citation_chars
10187                     + parsers.internal_punctuation
10188                     - parsers.comma
10189                     - parsers.semicolon)))^0
10190             * citation_chars)^-1)
10191
10192     local citation_body_prenote
10193         = Cs((parsers.alphanumeric^1
10194             + parsers.bracketed
10195             + parsers.inticks
10196             + parsers.autolink
10197             + V("InlineHtml")
10198             + V("Space") + V("EndlineNoSub")
10199             + (parsers.anyescaped
10200                 - ( parsers.newline
10201                     + parsers.rbracket
10202                     + parsers.blankline^2))
10203             - ( parsers.spnl
10204                 * parsers.dash^-1
10205                 * parsers.at))^1)
10206
10207     local citation_body_postnote
10208         = Cs((parsers.alphanumeric^1
10209             + parsers.bracketed
10210             + parsers.inticks

```

```

10211         + parsers.autolink
10212         + V("InlineHtml")
10213         + V("Space") + V("EndlineNoSub")
10214         + (parsers.anyescaped
10215           - ( parsers.newline
10216             + parsers.rbracket
10217             + parsers.semicolon
10218             + parsers.blankline^2))
10219         - (parsers.spnl * parsers.rbracket))^1)
10220
10221     local citation_body_chunk
10222         = ( citation_body_prenote
10223           * parsers.spnlc_sep
10224           + Cc("")
10225           * parsers.spnlc
10226         )
10227         * citation_name
10228         * ( parsers.internal_punctuation
10229           - parsers.semicolon)^-1
10230         * ( parsers.spnlc / function(_) return end
10231           * citation_body_postnote
10232           + Cc("")
10233           * parsers.spnlc
10234         )
10235
10236     local citation_body
10237         = citation_body_chunk
10238         * ( parsers.semicolon
10239           * parsers.spnlc
10240           * citation_body_chunk
10241         )^0
10242
10243     local citation_headless_body_postnote
10244         = Cs((parsers.alphanumeric^1
10245           + parsers.bracketed
10246           + parsers.inticks
10247           + parsers.autolink
10248           + V("InlineHtml")
10249           + V("Space") + V("Endline")
10250           + (parsers.anyescaped
10251             - ( parsers.newline
10252               + parsers.rbracket
10253               + parsers.at
10254               + parsers.semicolon + parsers.blankline^2))
10255           - (parsers.spnl * parsers.rbracket))^0)
10256
10257     local citation_headless_body

```

```

10258         = citation_headless_body_postnote
10259         * ( parsers.semicolon
10260         * parsers.spnlc
10261         * citation_body_chunk
10262         )^0
10263
10264     local citations
10265         = function(text_cites, raw_cites)
10266         local function normalize(str)
10267             if str == "" then
10268                 str = nil
10269             else
10270                 str = (citation_nbsps and
10271                     self.parser_functions.parse_inlines_nbsp or
10272                     self.parser_functions.parse_inlines)(str)
10273             end
10274             return str
10275         end
10276
10277         local cites = {}
10278         for i = 1,#raw_cites,4 do
10279             cites[#cites+1] = {
10280                 prenote = normalize(raw_cites[i]),
10281                 suppress_author = raw_cites[i+1] == "-",
10282                 name = writer.identifier(raw_cites[i+2]),
10283                 postnote = normalize(raw_cites[i+3]),
10284             }
10285         end
10286         return writer.citations(text_cites, cites)
10287     end
10288
10289     local TextCitations
10290         = Ct((parsers.spnlc
10291         * Cc("")
10292         * citation_name
10293         * ((parsers.spnlc
10294         * parsers.lbracket
10295         * citation_headless_body
10296         * parsers.rbracket) + Cc("")))^1)
10297     / function(raw_cites)
10298         return citations(true, raw_cites)
10299     end
10300
10301     local ParenthesizedCitations
10302         = Ct((parsers.spnlc
10303         * parsers.lbracket
10304         * citation_body

```

```

10305         * parsers.rbracket)^1)
10306         / function(raw_cites)
10307             return citations(false, raw_cites)
10308         end
10309
10310     local Citations = TextCitations + ParenthesizedCitations
10311
10312     self.insert_pattern("Inline before LinkAndEmph",
10313         Citations, "Citations")
10314
10315     self.add_special_character("@")
10316     self.add_special_character("-")
10317 end
10318 }
10319 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.

```

10320 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.

```

10321 local languages_json = (function()
10322     local base, prev, curr
10323     for _, pathname in ipairs{kpse.lookup(language_map,
10324         {all=true})} do
10325         local file = io.open(pathname, "r")
10326         if not file then goto continue end
10327         local input = assert(file:read("*a"))
10328         assert(file:close())
10329         local json = input:gsub('[^\n]-:', '[%1]=')
10330         curr = load("_ENV = {}; return "..json")()
10331         if type(curr) == "table" then
10332             if base == nil then
10333                 base = curr
10334             else
10335                 setmetatable(prev, { __index = curr })
10336             end
10337             prev = curr
10338         end
10339         ::continue::
10340     end

```

```

10341     return base or {}
10342 end)()
10343
10344 return {
10345     name = "built-in content_blocks syntax extension",
10346     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.

```

10347     function self.contentblock(src,suf,type,tit)
10348         if not self.is_writing then return "" end
10349         src = src.." "..suf
10350         suf = suf:lower()
10351         if type == "onlineimage" then
10352             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10353                 "{" ,self.string(src),"} ",
10354                 "{" ,self.uri(src),"} ",
10355                 "{" ,self.string(tit or ""),"} "}
10356         elseif languages_json[suf] then
10357             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10358                 "{" ,self.string(languages_json[suf]),"} ",
10359                 "{" ,self.string(src),"} ",
10360                 "{" ,self.uri(src),"} ",
10361                 "{" ,self.string(tit or ""),"} "}
10362         else
10363             return {"\\markdownRendererContentBlock{" ,suf,"} ",
10364                 "{" ,self.string(src),"} ",
10365                 "{" ,self.uri(src),"} ",
10366                 "{" ,self.string(tit or ""),"} "}
10367         end
10368     end
10369 end, extend_reader = function(self)
10370     local parsers = self.parsers
10371     local writer = self.writer
10372
10373     local contentblock_tail
10374         = parsers.optionaltitle
10375         * (parsers.newline + parsers.eof)
10376
10377     -- case insensitive online image suffix:
10378     local onlineimagesuffix
10379         = (function(...)
10380             local parser = nil
10381             for _, suffix in ipairs({...}) do
10382                 local pattern=nil

```

```

10383         for i=1,#suffix do
10384             local char=suffix:sub(i,i)
10385             char = S(char:lower()..char:upper())
10386             if pattern == nil then
10387                 pattern = char
10388             else
10389                 pattern = pattern * char
10390             end
10391         end
10392         if parser == nil then
10393             parser = pattern
10394         else
10395             parser = parser + pattern
10396         end
10397     end
10398     return parser
10399 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10400
10401 -- online image url for iA Writer content blocks with
10402 -- mandatory suffix, allowing nested brackets:
10403 local onlineimageurl
10404     = (parsers.less
10405         * Cs((parsers.anyescaped
10406             - parsers.more
10407             - parsers.spacing
10408             - #(parsers.period
10409                 * onlineimagesuffix
10410                 * parsers.more
10411                 * contentblock_tail))^0)
10412         * parsers.period
10413         * Cs(onlineimagesuffix)
10414         * parsers.more
10415         + (Cs((parsers.inparens
10416             + (parsers.anyescaped
10417                 - parsers.spacing
10418                 - parsers.rparent
10419                 - #(parsers.period
10420                     * onlineimagesuffix
10421                     * contentblock_tail))))^0)
10422         * parsers.period
10423         * Cs(onlineimagesuffix))
10424     ) * Cc("onlineimage")
10425
10426 -- filename for iA Writer content blocks with mandatory suffix:
10427 local localfilepath
10428     = parsers.slash
10429     * Cs((parsers.anyescaped

```

```

10430         - parsers.tab
10431         - parsers.newline
10432         - #(parsers.period
10433           * parsers.alphanumeric^1
10434           * contentblock_tail))^1)
10435     * parsers.period
10436     * Cs(parsers.alphanumeric^1)
10437     * Cc("localfile")
10438
10439     local ContentBlock
10440         = parsers.check_trail_no_rem
10441         * (localfilepath + onlineimageurl)
10442         * contentblock_tail
10443         / writer.contentblock
10444
10445     self.insert_pattern("Block before Blockquote",
10446                       ContentBlock, "ContentBlock")
10447   end
10448 }
10449 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.

```

10450 M.extensions.definition_lists = function(tight_lists)
10451   return {
10452     name = "built-in definition_lists syntax extension",
10453     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.

```

10454     local function dlitem(term, defs)
10455       local retVal = {"\\markdownRendererDlItem{",term,"}"}
10456       for _, def in ipairs(defs) do
10457         retVal[#retVal+1]
10458           = {"\\markdownRendererDlDefinitionBegin ",def,
10459             "\\markdownRendererDlDefinitionEnd "}
10460       end
10461       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10462       return retVal
10463     end
10464
10465     function self.definitionlist(items,tight)

```

```

10466     if not self.is_writing then return "" end
10467     local buffer = {}
10468     for _,item in ipairs(items) do
10469         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
10470     end
10471     if tight and tight_lists then
10472         return {"\\markdownRendererDlBeginTight\n", buffer,
10473             "\\n\\markdownRendererDlEndTight"}
10474     else
10475         return {"\\markdownRendererDlBegin\n", buffer,
10476             "\\n\\markdownRendererDlEnd"}
10477     end
10478 end
10479 end, extend_reader = function(self)
10480     local parsers = self.parsers
10481     local writer = self.writer
10482
10483     local defstartchar = S("~:")
10484
10485     local defstart
10486         = parsers.check_trail_length(0) * defstartchar
10487         * #parsers.spacing
10488         * (parsers.tab + parsers.space^-3)
10489         + parsers.check_trail_length(1)
10490         * defstartchar * #parsers.spacing
10491         * (parsers.tab + parsers.space^-2)
10492         + parsers.check_trail_length(2)
10493         * defstartchar * #parsers.spacing
10494         * (parsers.tab + parsers.space^-1)
10495         + parsers.check_trail_length(3)
10496         * defstartchar * #parsers.spacing
10497
10498     local indented_line
10499         = (parsers.check_minimal_indent / "")
10500         * parsers.check_code_trail * parsers.line
10501
10502     local blank
10503         = parsers.check_minimal_blank_indent_and_any_trail
10504         * parsers.optionalspace * parsers.newline
10505
10506     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10507
10508     local indented_blocks = function(bl)
10509         return Cs( bl
10510             * (blank^1 * (parsers.check_minimal_indent / ""))
10511             * parsers.check_code_trail * -parsers.blankline * bl)^0
10512             * (blank^1 + parsers.eof))

```



```

10513     end
10514
10515     local function definition_list_item(term, defs, _)
10516         return { term = self.parser_functions.parse_inlines(term),
10517                 definitions = defs }
10518     end
10519
10520     local DefinitionListItemLoose
10521     = C(parsers.line) * blank^0
10522     * Ct((parsers.check_minimal_indent * (defstart
10523         * indented_blocks(dlchunk)
10524         / self.parser_functions.parse_blocks_nested))^1)
10525     * Cc(false) / definition_list_item
10526
10527     local DefinitionListItemTight
10528     = C(parsers.line)
10529     * Ct((parsers.check_minimal_indent * (defstart * dlchunk
10530         / self.parser_functions.parse_blocks_nested))^1)
10531     * Cc(true) / definition_list_item
10532
10533     local DefinitionList
10534     = ( Ct(DefinitionListItemLoose^1) * Cc(false)
10535         + Ct(DefinitionListItemTight^1)
10536         * (blank^0
10537           * -DefinitionListItemLoose * Cc(true))
10538         ) / writer.definitionlist
10539
10540     self.insert_pattern("Block after Heading",
10541                       DefinitionList, "DefinitionList")
10542     end
10543 }
10544 end

```

3.1.7.5 Fancy Lists

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

```

10545 M.extensions.fancy_lists = function()
10546     return {
10547         name = "built-in fancy_lists syntax extension",
10548         extend_writer = function(self)
10549             local options = self.options
10550

```

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.

```

10551     function self.fancylist(items,tight,startnum,numstyle,numdelim)
10552         if not self.is_writing then return "" end
10553         local buffer = {}
10554         local num = startnum
10555         for _,item in ipairs(items) do
10556             if item ~= "" then
10557                 buffer[#buffer + 1] = self.fancyitem(item,num)
10558             end
10559             if num ~= nil and item ~= "" then
10560                 num = num + 1
10561             end
10562         end
10563         local contents = util.intersperse(buffer,"\n")
10564         if tight and options.tightLists then
10565             return {"\markdownRendererFancy0lBeginTight{",
10566                 numstyle,"}{",numdelim,"}",contents,
10567                 "\n\markdownRendererFancy0lEndTight "}
10568         else
10569             return {"\markdownRendererFancy0lBegin{",
10570                 numstyle,"}{",numdelim,"}",contents,
10571                 "\n\markdownRendererFancy0lEnd "}
10572         end
10573     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.

```

10574     function self.fancyitem(s,num)
10575         if num ~= nil then

```

```

10576         return {"\\markdownRendererFancyOliItemWithNumber{" ,num,"} ",s,
10577                 "\\markdownRendererFancyOliItemEnd "}
10578     else
10579         return {"\\markdownRendererFancyOliItem " ,s,
10580                 "\\markdownRendererFancyOliItemEnd "}
10581     end
10582 end
10583 end, extend_reader = function(self)
10584     local parsers = self.parsers
10585     local options = self.options
10586     local writer = self.writer
10587
10588     local function combine_markers_and_delims(markers, delims)
10589         local markers_table = {}
10590         for _,marker in ipairs(markers) do
10591             local start_marker
10592             local continuation_marker
10593             if type(marker) == "table" then
10594                 start_marker = marker[1]
10595                 continuation_marker = marker[2]
10596             else
10597                 start_marker = marker
10598                 continuation_marker = marker
10599             end
10600             for _,delim in ipairs(delims) do
10601                 table.insert(markers_table,
10602                             {start_marker, continuation_marker, delim})
10603             end
10604         end
10605         return markers_table
10606     end
10607
10608     local function join_table_with_func(func, markers_table)
10609         local pattern = func(table.unpack(markers_table[1]))
10610         for i = 2, #markers_table do
10611             pattern = pattern + func(table.unpack(markers_table[i]))
10612         end
10613         return pattern
10614     end
10615
10616     local lowercase_letter_marker = R("az")
10617     local uppercase_letter_marker = R("AZ")
10618
10619     local roman_marker = function(chars)
10620         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
10621         local l, x, v, i
10622         = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])

```

```

10623     return  m^-3
10624           * (c*m + c*d + d^-1 * c^-3)
10625           * (x*c + x*l + l^-1 * x^-3)
10626           * (i*x + i*v + v^-1 * i^-3)
10627     end
10628
10629     local lowercase_roman_marker
10630           = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10631     local uppercase_roman_marker
10632           = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10633
10634     local lowercase_opening_roman_marker = P("i")
10635     local uppercase_opening_roman_marker = P("I")
10636
10637     local digit_marker = parsers.dig * parsers.dig^-8
10638
10639     local markers = {
10640       {lowercase_opening_roman_marker, lowercase_roman_marker},
10641       {uppercase_opening_roman_marker, uppercase_roman_marker},
10642       lowercase_letter_marker,
10643       uppercase_letter_marker,
10644       lowercase_roman_marker,
10645       uppercase_roman_marker,
10646       digit_marker
10647     }
10648
10649     local delims = {
10650       parsers.period,
10651       parsers.rparent
10652     }
10653
10654     local markers_table = combine_markers_and_delims(markers, delims)
10655
10656     local function enumerator(start_marker, _,
10657                               delimiter_type, interrupting)
10658       local delimiter_range
10659       local allowed_end
10660       if interrupting then
10661         delimiter_range = P("1")
10662         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10663       else
10664         delimiter_range = start_marker
10665         allowed_end = C(parsers.spacechar^1)
10666                     + #(parsers.newline + parsers.eof)
10667       end
10668
10669     return parsers.check_trail

```

```

10670             * Ct(C(delimiter_range) * C(delimiter_type))
10671             * allowed_end
10672         end
10673
10674         local starter = join_table_with_func(enumerator, markers_table)
10675
10676         local TightListItem = function(starter)
10677             return parsers.add_indent(starter, "li")
10678                 * parsers.indented_content_tight
10679         end
10680
10681         local LooseListItem = function(starter)
10682             return parsers.add_indent(starter, "li")
10683                 * parsers.indented_content_loose
10684                 * remove_indent("li")
10685         end
10686
10687         local function roman2number(roman)
10688             local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
10689                 ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
10690             local numeral = 0
10691
10692             local i = 1
10693             local len = string.len(roman)
10694             while i < len do
10695                 local z1, z2 = romans[ string.sub(roman, i, i) ],
10696                     romans[ string.sub(roman, i+1, i+1) ]
10697                 if z1 < z2 then
10698                     numeral = numeral + (z2 - z1)
10699                     i = i + 2
10700                 else
10701                     numeral = numeral + z1
10702                     i = i + 1
10703                 end
10704             end
10705             if i <= len then
10706                 numeral = numeral + romans[ string.sub(roman,i,i) ]
10707             end
10708             return numeral
10709         end
10710
10711         local function sniffstyle(numstr, delimend)
10712             local numdelim
10713             if delimend == ")" then
10714                 numdelim = "OneParen"
10715             elseif delimend == "." then
10716                 numdelim = "Period"

```

```

10717     else
10718         numdelim = "Default"
10719     end
10720
10721     local num
10722     num = numstr:match("^([I])$")
10723     if num then
10724         return roman2number(num), "UpperRoman", numdelim
10725     end
10726     num = numstr:match("^([i])$")
10727     if num then
10728         return roman2number(string.upper(num)), "LowerRoman", numdelim
10729     end
10730     num = numstr:match("^([A-Z])$")
10731     if num then
10732         return string.byte(num) - string.byte("A") + 1,
10733             "UpperAlpha", numdelim
10734     end
10735     num = numstr:match("^([a-z])$")
10736     if num then
10737         return string.byte(num) - string.byte("a") + 1,
10738             "LowerAlpha", numdelim
10739     end
10740     num = numstr:match("^([IVXLCDM]+)")
10741     if num then
10742         return roman2number(num), "UpperRoman", numdelim
10743     end
10744     num = numstr:match("^([ivxlcdm]+)")
10745     if num then
10746         return roman2number(string.upper(num)), "LowerRoman", numdelim
10747     end
10748     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10749 end
10750
10751 local function fancylist(items,tight,start)
10752     local startnum, numstyle, numdelim
10753     = sniffstyle(start[2][1], start[2][2])
10754     return writer.fancylist(items,tight,
10755                             options.startNumber and startnum or 1,
10756                             numstyle or "Decimal",
10757                             numdelim or "Default")
10758 end
10759
10760 local FancyListOfType
10761 = function(start_marker, continuation_marker, delimiter_type)
10762     local enumerator_start
10763     = enumerator(start_marker, continuation_marker,

```

```

10764         delimiter_type)
10765     local enumerator_cont
10766         = enumerator(continuation_marker, continuation_marker,
10767                     delimiter_type)
10768     return Cg(enumerator_start, "listtype")
10769         * (Ct( TightListItem(Cb("listtype"))
10770             * ((parsers.check_minimal_indent / "")
10771               * TightListItem(enumerator_cont))^0)
10772         * Cc(true)
10773         * -#((parsers.conditionally_indented_blankline^0 / "")
10774             * parsers.check_minimal_indent * enumerator_cont)
10775         + Ct( LooseListItem(Cb("listtype"))
10776             * ((parsers.conditionally_indented_blankline^0 / "")
10777               * (parsers.check_minimal_indent / "")
10778               * LooseListItem(enumerator_cont))^0)
10779         * Cc(false)
10780         ) * Ct(Cb("listtype")) / fancylist
10781     end
10782
10783     local FancyList
10784         = join_table_with_func(FancyListOfType, markers_table)
10785
10786     local Endline = parsers.newline
10787         * (parsers.check_minimal_indent
10788           * -parsers.EndlineExceptions
10789           + parsers.check_optional_indent
10790           * -parsers.EndlineExceptions
10791           * -starter)
10792         * parsers.spacechar^0
10793         / writer.soft_line_break
10794
10795     self.update_rule("OrderedList", FancyList)
10796     self.update_rule("Endline", Endline)
10797 end
10798 }
10799 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.

```
10800 M.extensions.fenced_code = function(blank_before_code_fence,
10801                                     allow_attributes,
10802                                     allow_raw_blocks)
10803   return {
10804     name = "built-in fenced_code syntax extension",
10805     extend_writer = function(self)
10806       local options = self.options
10807
```

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.

```
10808     function self.fencedCode(s, i, attr)
10809       if not self.is_writing then return "" end
10810       s = s:gsub("\n$", "")
10811       local buf = {}
10812       if attr ~= nil then
10813         table.insert(buf,
10814           {"\\markdownRendererFencedCodeAttributeContextBegin",
10815             self.attributes(attr)})
10816       end
10817       local name = util.cache_verbatim(options.cacheDir, s)
10818       table.insert(buf,
10819         {"\\markdownRendererInputFencedCode{" ,
10820           name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}"}))
10821       if attr ~= nil then
10822         table.insert(buf,
10823           "\\markdownRendererFencedCodeAttributeContextEnd{")
10824       end
10825       return buf
10826     end
10827
```

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

```
10828     if allow_raw_blocks then
10829       function self.rawBlock(s, attr)
10830         if not self.is_writing then return "" end
10831         s = s:gsub("\n$", "")
10832         local name = util.cache_verbatim(options.cacheDir, s)
10833         return {"\\markdownRendererInputRawBlock{" ,
10834           name,"}{" , self.string(attr),"}"}
10835       end
10836     end
10837   end, extend_reader = function(self)
10838     local parsers = self.parsers
```



```

10839     local writer = self.writer
10840
10841     local function captures_geq_length(_,i,a,b)
10842         return #a >= #b and i
10843     end
10844
10845     local function strip_enclosing_whitespaces(str)
10846         return str:gsub("^%s*(.)%s*$", "%1")
10847     end
10848
10849     local tilde_infostring = Cs(Cs((V("HtmlEntity")
10850         + parsers.anyescaped
10851         - parsers.newline)^0)
10852         / strip_enclosing_whitespaces)
10853
10854     local backtick_infostring
10855         = Cs( Cs((V("HtmlEntity")
10856             + ( -#(parsers.backslash * parsers.backtick)
10857                 * parsers.anyescaped)
10858                 - parsers.newline
10859                 - parsers.backtick)^0)
10860             / strip_enclosing_whitespaces)
10861
10862     local fenceindent
10863
10864     local function has_trail(indent_table)
10865         return indent_table ~= nil and
10866             indent_table.trail ~= nil and
10867             next(indent_table.trail) ~= nil
10868     end
10869
10870     local function has_indents(indent_table)
10871         return indent_table ~= nil and
10872             indent_table.indents ~= nil and
10873             next(indent_table.indents) ~= nil
10874     end
10875
10876     local function get_last_indent_name(indent_table)
10877         if has_indents(indent_table) then
10878             return indent_table.indents[#indent_table.indents].name
10879         end
10880     end
10881
10882     local count_fenced_start_indent =
10883         function(_, _, indent_table, trail)
10884             local last_indent_name = get_last_indent_name(indent_table)
10885             fenceindent = 0

```

```

10886         if last_indent_name ~= "li" then
10887             fenceindent = #trail
10888         end
10889         return true
10890     end
10891
10892     local fencehead = function(char, infostring)
10893         return Cmt( Cb("indent_info")
10894             * parsers.check_trail, count_fenced_start_indent)
10895             * Cg(char^3, "fencelength")
10896             * parsers.optionalspace
10897             * infostring
10898             * (parsers.newline + parsers.eof)
10899     end
10900
10901     local fencetail = function(char)
10902         return parsers.check_trail_no_rem
10903             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10904             * parsers.optionalspace * (parsers.newline + parsers.eof)
10905             + parsers.eof
10906     end
10907
10908     local process_fenced_line =
10909         function(s, i, -- luacheck: ignore s i
10910             indent_table, line_content, is_blank)
10911         local remainder = ""
10912         if has_trail(indent_table) then
10913             remainder = indent_table.trail.internal_remainder
10914         end
10915
10916         if is_blank
10917             and get_last_indent_name(indent_table) == "li" then
10918             remainder = ""
10919         end
10920
10921         local str = remainder .. line_content
10922         local index = 1
10923         local remaining = fenceindent
10924
10925         while true do
10926             local c = str:sub(index, index)
10927             if c == " " and remaining > 0 then
10928                 remaining = remaining - 1
10929                 index = index + 1
10930             elseif c == "\t" and remaining > 3 then
10931                 remaining = remaining - 4
10932                 index = index + 1

```

```

10933         else
10934             break
10935         end
10936     end
10937
10938     return true, str:sub(index)
10939 end
10940
10941 local fencedline = function(char)
10942     return Cmt( Cb("indent_info")
10943         * C(parsers.line - fencetail(char))
10944         * Cc(false), process_fenced_line)
10945 end
10946
10947 local blankfencedline
10948     = Cmt( Cb("indent_info")
10949         * C(parsers.blankline)
10950         * Cc(true), process_fenced_line)
10951
10952 local TildeFencedCode
10953     = fencehead(parsers.tilde, tilde_infostring)
10954     * Cs(( parsers.check_minimal_blank_indent / ""
10955         * blankfencedline
10956         + ( parsers.check_minimal_indent / ""
10957         * fencedline(parsers.tilde))^0)
10958     * ( (parsers.check_minimal_indent / ""
10959         * fencetail(parsers.tilde) + parsers.succeed)
10960
10961 local BacktickFencedCode
10962     = fencehead(parsers.backtick, backtick_infostring)
10963     * Cs(( (parsers.check_minimal_blank_indent / ""
10964         * blankfencedline
10965         + (parsers.check_minimal_indent / ""
10966         * fencedline(parsers.backtick))^0)
10967     * ( (parsers.check_minimal_indent / ""
10968         * fencetail(parsers.backtick) + parsers.succeed)
10969
10970 local infostring_with_attributes
10971     = Ct(C((parsers.linechar
10972         - ( parsers.optionalspace
10973         * parsers.attributes))^0)
10974         * parsers.optionalspace
10975         * Ct(parsers.attributes))
10976
10977 local FencedCode
10978     = ((TildeFencedCode + BacktickFencedCode)
10979     / function(infostring, code)

```

```

10980         local expanded_code = self.expandtabs(code)
10981
10982         if allow_raw_blocks then
10983             local raw_attr = lpeg.match(parsers.raw_attribute,
10984                                       infostring)
10985             if raw_attr then
10986                 return writer.rawBlock(expanded_code, raw_attr)
10987             end
10988         end
10989
10990         local attr = nil
10991         if allow_attributes then
10992             local match = lpeg.match(infostring_with_attributes,
10993                                     infostring)
10994             if match then
10995                 infostring, attr = table.unpack(match)
10996             end
10997         end
10998         return writer.fencedCode(expanded_code, infostring, attr)
10999     end)
11000
11001     self.insert_pattern("Block after Verbatim",
11002                       FencedCode, "FencedCode")
11003
11004     local fencestart
11005     if blank_before_code_fence then
11006         fencestart = parsers.fail
11007     else
11008         fencestart = fencehead(parsers.backtick, backtick_infostring)
11009                       + fencehead(parsers.tilde, tilde_infostring)
11010     end
11011
11012     self.update_rule("EndlineExceptions", function(previous_pattern)
11013         if previous_pattern == nil then
11014             previous_pattern = parsers.EndlineExceptions
11015         end
11016         return previous_pattern + fencestart
11017     end)
11018
11019     self.add_special_character("`")
11020     self.add_special_character("~")
11021 end
11022 }
11023 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.

```
11024 M.extensions.fenced_divs = function(blank_before_div_fence)
11025   return {
11026     name = "built-in fenced_divs syntax extension",
11027     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.

```
11028     function self.div_begin(attributes)
11029       local start_output
11030       = {"\\markdownRendererFencedDivAttributeContextBegin\n",
11031         self.attributes(attributes)}
11032       local end_output
11033       = {"\\markdownRendererFencedDivAttributeContextEnd{}}
11034       return self.push_attributes(
11035         "div", attributes, start_output, end_output)
11036     end
```

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

```
11037     function self.div_end()
11038       return self.pop_attributes("div")
11039     end
11040   end, extend_reader = function(self)
11041     local parsers = self.parsers
11042     local writer = self.writer
```

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

```
11043     local fenced_div_infostring
11044       = C((parsers.linechar
11045         - ( parsers.spacechar^1
11046           * parsers.colon^1))^1)
11047
11048     local fenced_div_begin = parsers.nonindentspace
11049       * parsers.colon^3
11050       * parsers.optionalspace
11051       * fenced_div_infostring
11052       * ( parsers.spacechar^1
11053         * parsers.colon^1)^0
11054       * parsers.optionalspace
11055       * (parsers.newline + parsers.eof)
11056
11057     local fenced_div_end = parsers.nonindentspace
11058       * parsers.colon^3
```

```

11059             * parsers.optionalspace
11060             * (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.

```

11061     self.initialize_named_group("fenced_div_level", "0")
11062     self.initialize_named_group("fenced_div_num_opening_indents")
11063
11064     local function increment_div_level()
11065         local push_indent_table =
11066             function(s, i, indent_table, -- luacheck: ignore s i
11067                 fenced_div_num_opening_indents, fenced_div_level)
11068             fenced_div_level = tonumber(fenced_div_level) + 1
11069             local num_opening_indents = 0
11070             if indent_table.indents ~= nil then
11071                 num_opening_indents = #indent_table.indents
11072             end
11073             fenced_div_num_opening_indents[fenced_div_level]
11074                 = num_opening_indents
11075             return true, fenced_div_num_opening_indents
11076         end
11077
11078     local increment_level =
11079         function(s, i, fenced_div_level) -- luacheck: ignore s i
11080             fenced_div_level = tonumber(fenced_div_level) + 1
11081             return true, tostring(fenced_div_level)
11082         end
11083
11084     return Cg( Cmt( Cb("indent_info")
11085         * Cb("fenced_div_num_opening_indents")
11086         * Cb("fenced_div_level"), push_indent_table)
11087         , "fenced_div_num_opening_indents")
11088         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11089         , "fenced_div_level")
11090     end
11091
11092     local function decrement_div_level()
11093         local pop_indent_table =
11094             function(s, i, -- luacheck: ignore s i
11095                 fenced_div_indent_table, fenced_div_level)
11096             fenced_div_level = tonumber(fenced_div_level)
11097             fenced_div_indent_table[fenced_div_level] = nil
11098             return true, tostring(fenced_div_level - 1)

```

```

11099         end
11100
11101         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11102             * Cb("fenced_div_level"), pop_indent_table)
11103             , "fenced_div_level")
11104     end
11105
11106
11107     local non_fenced_div_block
11108         = parsers.check_minimal_indent * V("Block")
11109         - parsers.check_minimal_indent_and_trail * fenced_div_end
11110
11111     local non_fenced_div_paragraph
11112         = parsers.check_minimal_indent * V("Paragraph")
11113         - parsers.check_minimal_indent_and_trail * fenced_div_end
11114
11115     local blank = parsers.minimally_indented_blank
11116
11117     local block_separated = parsers.block_sep_group(blank)
11118         * non_fenced_div_block
11119
11120     local loop_body_pair
11121         = parsers.create_loop_body_pair(block_separated,
11122             non_fenced_div_paragraph,
11123             parsers.block_sep_group(blank),
11124             parsers.par_sep_group(blank))
11125
11126     local content_loop = ( non_fenced_div_block
11127         * loop_body_pair.block^0
11128         + non_fenced_div_paragraph
11129         * block_separated
11130         * loop_body_pair.block^0
11131         + non_fenced_div_paragraph
11132         * loop_body_pair.par^0)
11133         * blank^0
11134
11135     local FencedDiv = fenced_div_begin
11136         / function (infostring)
11137             local attr
11138                 = lpeg.match(Ct(parsers.attributes),
11139                     infostring)
11140             if attr == nil then
11141                 attr = {"." .. infostring}
11142             end
11143             return attr
11144         end
11145         / writer.div_begin

```

```

11146         * increment_div_level()
11147         * parsers.skipblanklines
11148         * Ct(content_loop)
11149         * parsers.minimally_indented_blank^0
11150         * parsers.check_minimal_indent_and_trail
11151         * fenced_div_end
11152         * decrement_div_level()
11153         * (Cc("") / writer.div_end)
11154
11155     self.insert_pattern("Block after Verbatim",
11156                       FencedDiv, "FencedDiv")
11157
11158     self.add_special_character(":")
11159

```

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.

```

11160     local function is_inside_div()
11161         local check_div_level =
11162             function(s, i, fenced_div_level) -- luacheck: ignore s i
11163                 fenced_div_level = tonumber(fenced_div_level)
11164                 return fenced_div_level > 0
11165             end
11166
11167     return Cmt(Cb("fenced_div_level"), check_div_level)
11168 end
11169
11170 local function check_indent()
11171     local compare_indent =
11172         function(s, i, indent_table, -- luacheck: ignore s i
11173                fenced_div_num_opening_indents, fenced_div_level)
11174             fenced_div_level = tonumber(fenced_div_level)
11175             local num_current_indents
11176                 = ( indent_table.current_line_indents ~= nil and
11177                   #indent_table.current_line_indents) or 0
11178             local num_opening_indents
11179                 = fenced_div_num_opening_indents[fenced_div_level]
11180             return num_current_indents == num_opening_indents
11181         end
11182
11183     return Cmt( Cb("indent_info")
11184               * Cb("fenced_div_num_opening_indents")
11185               * Cb("fenced_div_level"), compare_indent)
11186 end
11187
11188 local fencestart = is_inside_div()

```



```

11189             * fenced_div_end
11190             * check_indent()
11191
11192     if not blank_before_div_fence then
11193         self.update_rule("EndlineExceptions", function(previous_pattern)
11194             if previous_pattern == nil then
11195                 previous_pattern = parsers.EndlineExceptions
11196             end
11197             return previous_pattern + fencestart
11198         end)
11199     end
11200 end
11201 }
11202 end

```

3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11203 M.extensions.header_attributes = function()
11204     return {
11205         name = "built-in header_attributes syntax extension",
11206         extend_writer = function()
11207             end, extend_reader = function(self)
11208                 local parsers = self.parsers
11209                 local writer = self.writer
11210
11211                 local function strip_atx_end(s)
11212                     return s:gsub("%s+##%s*$", "")
11213                 end
11214
11215                 local AtxHeading = Cg(parsers.heading_start, "level")
11216                     * parsers.optionalspace
11217                     * (C(((parsers.linechar
11218                         - (parsers.attributes
11219                             * parsers.optionalspace
11220                             * parsers.newline))
11221                         * (parsers.linechar
11222                             - parsers.lbrace)^0)^1)
11223                         / strip_atx_end
11224                         / parsers.parse_heading_text)
11225                 * Cg(Ct(parsers.newline
11226                     + (parsers.attributes
11227                         * parsers.optionalspace
11228                         * parsers.newline)), "attributes")
11229                 * Cb("level")
11230                 * Cb("attributes")

```

```

11231             / writer.heading
11232
11233     local function strip_trailing_spaces(s)
11234         return s:gsub("%s*$","")
11235     end
11236
11237     local heading_line = (parsers.linechar
11238                         - (parsers.attributes
11239                           * parsers.optionalspace
11240                           * parsers.newline))^1
11241                         - parsers.thematic_break_lines
11242
11243     local heading_text
11244         = heading_line
11245         * ( (V("Endline") / "\n")
11246           * (heading_line - parsers.heading_level))^0
11247         * parsers.newline^-1
11248
11249     local SettextHeading
11250         = parsers.freeze_trail * parsers.check_trail_no_rem
11251         * #(heading_text
11252           * (parsers.attributes
11253             * parsers.optionalspace
11254             * parsers.newline)^-1
11255           * parsers.check_minimal_indent
11256           * parsers.check_trail
11257           * parsers.heading_level)
11258         * Cs(heading_text) / strip_trailing_spaces
11259         / parsers.parse_heading_text
11260         * Cg(Ct((parsers.attributes
11261               * parsers.optionalspace
11262               * parsers.newline)^-1), "attributes")
11263         * parsers.check_minimal_indent_and_trail * parsers.heading_level
11264         * Cb("attributes")
11265         * parsers.newline
11266         * parsers.unfreeze_trail
11267         / writer.heading
11268
11269     local Heading = AtxHeading + SettextHeading
11270     self.update_rule("Heading", Heading)
11271 end
11272 }
11273 end

```

3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc in-line code attribute syntax extension.

```
11274 M.extensions.inline_code_attributes = function()
11275   return {
11276     name = "built-in inline_code_attributes syntax extension",
11277     extend_writer = function()
11278     end, extend_reader = function(self)
11279       local writer = self.writer
11280
11281       local CodeWithAttributes = parsers.inticks
11282         * Ct(parsers.attributes)
11283         / writer.code
11284
11285       self.insert_pattern("Inline before Code",
11286         CodeWithAttributes,
11287         "CodeWithAttributes")
11288     end
11289   }
11290 end
```

3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
11291 M.extensions.line_blocks = function()
11292   return {
11293     name = "built-in line_blocks syntax extension",
11294     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.

```
11295       function self.lineblock(lines)
11296         if not self.is_writing then return "" end
11297         local buffer = {}
11298         for i = 1, #lines - 1 do
11299           buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11300         end
11301         buffer[#buffer + 1] = lines[#lines]
11302
11303         return {"\\markdownRendererLineBlockBegin\n"
11304           ,buffer,
11305           "\n\\markdownRendererLineBlockEnd "}
11306       end
11307     end, extend_reader = function(self)
11308       local parsers = self.parsers
11309       local writer = self.writer
11310
```

```

11311     local LineBlock
11312         = Ct((Cs(( (parsers.pipe * parsers.space) / ""
11313                 * ((parsers.space)/entities.char_entity("nbsp"))^0
11314                 * parsers.linechar^0 * (parsers.newline/""))
11315                 * (-parsers.pipe
11316                   * (parsers.space^1/" ")
11317                   * parsers.linechar^1
11318                   * (parsers.newline/"")
11319                   )^0
11320                 * (parsers.blankline/"")^0)
11321           / self.parser_functions.parse_inlines)^1)
11322     / writer.lineblock
11323
11324     self.insert_pattern("Block after Blockquote",
11325                       LineBlock, "LineBlock")
11326 end
11327 }
11328 end

```

3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11329 M.extensions.mark = function()
11330     return {
11331         name = "built-in mark syntax extension",
11332         extend_writer = function(self)

```

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

```

11333         function self.mark(s)
11334             if self.flatten_inlines then return s end
11335             return {"\\markdownRendererMark{" , s, "}"}
11336         end
11337     end, extend_reader = function(self)
11338         local parsers = self.parsers
11339         local writer = self.writer
11340
11341         local doubleequals = P("==")
11342
11343         local Mark
11344             = parsers.between(V("Inline"), doubleequals, doubleequals)
11345             / function (inlines) return writer.mark(inlines) end
11346
11347         self.add_special_character("=")
11348         self.insert_pattern("Inline before LinkAndEmph",
11349                             Mark, "Mark")
11350     end
11351 }

```

11352 end

3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11353 M.extensions.link_attributes = function()
11354   return {
11355     name = "built-in link_attributes syntax extension",
11356     extend_writer = function()
11357     end, extend_reader = function(self)
11358       local parsers = self.parsers
11359       local options = self.options
11360
```

The following patterns define link reference definitions with attributes.

```
11361     local define_reference_parser
11362       = (parsers.check_trail / ""
11363         * parsers.link_label
11364         * parsers.colon
11365         * parsers.spnlc * parsers.url
11366         * ( parsers.spnlc_sep * parsers.title
11367           * (parsers.spnlc * Ct(parsers.attributes))
11368           * parsers.only_blank
11369           + parsers.spnlc_sep * parsers.title * parsers.only_blank
11370           + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11371           * parsers.only_blank
11372           + Cc("") * parsers.only_blank)
11373
11374     local ReferenceWithAttributes = define_reference_parser
11375                                   / self.register_link
11376
11377     self.update_rule("Reference", ReferenceWithAttributes)
11378
```

The following patterns define direct and indirect links with attributes.

```
11379
11380     local LinkWithAttributesAndEmph
11381       = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11382         "match_link_attributes"))
11383       / self.defer_link_and_emphasis_processing
11384
11385     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11386
```

The following patterns define autolinks with attributes.

```
11387     local AutoLinkUrlWithAttributes
11388       = parsers.auto_link_url
```

```

11389             * Ct(parsers.attributes)
11390             / self.auto_link_url
11391
11392     self.insert_pattern("Inline before AutoLinkUrl",
11393                       AutoLinkUrlWithAttributes,
11394                       "AutoLinkUrlWithAttributes")
11395
11396     local AutoLinkEmailWithAttributes
11397           = parsers.auto_link_email
11398           * Ct(parsers.attributes)
11399           / self.auto_link_email
11400
11401     self.insert_pattern("Inline before AutoLinkEmail",
11402                       AutoLinkEmailWithAttributes,
11403                       "AutoLinkEmailWithAttributes")
11404
11405     if options.relativeReferences then
11406
11407         local AutoLinkRelativeReferenceWithAttributes
11408               = parsers.auto_link_relative_reference
11409               * Ct(parsers.attributes)
11410               / self.auto_link_url
11411
11412         self.insert_pattern(
11413           "Inline before AutoLinkRelativeReference",
11414           AutoLinkRelativeReferenceWithAttributes,
11415           "AutoLinkRelativeReferenceWithAttributes")
11416
11417     end
11418
11419 end
11420 }
11421 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.

```

11422 M.extensions.notes = function(notes, inline_notes)
11423   assert(notes or inline_notes)
11424   return {
11425     name = "built-in notes syntax extension",
11426     extend_writer = function(self)

```

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

```
11427     function self.note(s)
11428         if self.flatten_inlines then return "" end
11429         return {"\\markdownRendererNote{" ,s,"}"}
11430     end
11431 end, extend_reader = function(self)
11432     local parsers = self.parsers
11433     local writer = self.writer
11434
11435     local rawnotes = parsers.rawnotes
11436
11437     if inline_notes then
11438         local InlineNote
11439             = parsers.circumflex
11440             * ( parsers.link_label
11441               / self.parser_functions.parse_inlines_no_inline_note)
11442             / writer.note
11443
11444         self.insert_pattern("Inline after LinkAndEmph",
11445                             InlineNote, "InlineNote")
11446     end
11447     if notes then
11448         local function strip_first_char(s)
11449             return s:sub(2)
11450         end
11451
11452         local RawNoteRef
11453             = #(parsers.lbracket * parsers.circumflex)
11454             * parsers.link_label / strip_first_char
11455
11456         -- like indirect_link
11457         local function lookup_note(ref)
11458             return writer.defer_call(function()
11459                 local found = rawnotes[self.normalize_tag(ref)]
11460                 if found then
11461                     return writer.note(
11462                         self.parser_functions.parse_blocks_nested(found))
11463                 else
11464                     return {"[",
11465                             self.parser_functions.parse_inlines("^" .. ref), "]" }
11466                 end
11467             end)
11468         end
11469
11470         local function register_note(ref,rawnote)
11471             local normalized_tag = self.normalize_tag(ref)
```

```

11472         if rawnotes[normalized_tag] == nil then
11473             rawnotes[normalized_tag] = rawnote
11474         end
11475         return ""
11476     end
11477
11478     local NoteRef = RawNoteRef / lookup_note
11479
11480     local optionally_indented_line
11481         = parsers.check_optional_indent_and_any_trail * parsers.line
11482
11483     local blank
11484         = parsers.check_optional_blank_indent_and_any_trail
11485         * parsers.optionalspace * parsers.newline
11486
11487     local chunk
11488         = Cs(parsers.line
11489             * (optionally_indented_line - blank)^0)
11490
11491     local indented_blocks = function(bl)
11492         return Cs( bl
11493             * ( blank^1 * (parsers.check_optional_indent / "")
11494               * parsers.check_code_trail
11495               * -parsers.blankline * bl)^0)
11496     end
11497
11498     local NoteBlock
11499         = parsers.check_trail_no_rem
11500         * RawNoteRef * parsers.colon
11501         * parsers.spnlc * indented_blocks(chunk)
11502         / register_note
11503
11504     local Reference = NoteBlock + parsers.Reference
11505
11506     self.update_rule("Reference", Reference)
11507     self.insert_pattern("Inline before LinkAndEmph",
11508         NoteRef, "NoteRef")
11509     end
11510
11511     self.add_special_character("^")
11512 end
11513 }
11514 end

```

3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syn-

tax 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.

```

11515 M.extensions.pipe_tables = function(table_captions, table_attributes)
11516
11517   local function make_pipe_table_rectangular(rows)
11518     local num_columns = #rows[2]
11519     local rectangular_rows = {}
11520     for i = 1, #rows do
11521       local row = rows[i]
11522       local rectangular_row = {}
11523       for j = 1, num_columns do
11524         rectangular_row[j] = row[j] or ""
11525       end
11526       table.insert(rectangular_rows, rectangular_row)
11527     end
11528     return rectangular_rows
11529   end
11530
11531   local function pipe_table_row(allow_empty_first_column
11532                                 , nonempty_column
11533                                 , column_separator
11534                                 , column)
11535     local row_beginning
11536     if allow_empty_first_column then
11537       row_beginning = -- empty first column
11538                       #(parsers.spacechar^4
11539                         * column_separator)
11540                       * parsers.optionalspace
11541                       * column
11542                       * parsers.optionalspace
11543                       -- non-empty first column
11544                       + parsers.nonindentSPACE
11545                       * nonempty_column^-1
11546                       * parsers.optionalspace
11547     else
11548       row_beginning = parsers.nonindentSPACE
11549                       * nonempty_column^-1
11550                       * parsers.optionalspace
11551     end
11552
11553   return Ct(row_beginning
11554             * (-- single column with no leading pipes
11555               #(column_separator

```

```

11556         * parsers.optionalspace
11557         * parsers.newline)
11558     * column_separator
11559     * parsers.optionalspace
11560     -- single column with leading pipes or
11561     -- more than a single column
11562     + (column_separator
11563       * parsers.optionalspace
11564       * column
11565       * parsers.optionalspace)^1
11566     * (column_separator
11567       * parsers.optionalspace)^-1))
11568 end
11569
11570 return {
11571     name = "built-in pipe_tables syntax extension",
11572     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.

```

11573     function self.table(rows, caption, attributes)
11574         if not self.is_writing then return "" end
11575         local buffer = {}
11576         if attributes ~= nil then
11577             table.insert(buffer,
11578                 "\\markdownRendererTableAttributeContextBegin\n")
11579             table.insert(buffer, self.attributes(attributes))
11580         end
11581         table.insert(buffer,
11582             {"\\markdownRendererTable{" ,
11583             caption or "", "}{" , #rows - 1, "}{" ,
11584             #rows[1], "}")
11585         local temp = rows[2] -- put alignments on the first row
11586         rows[2] = rows[1]
11587         rows[1] = temp
11588         for i, row in ipairs(rows) do
11589             table.insert(buffer, "{")
11590             for _, column in ipairs(row) do
11591                 if i > 1 then -- do not use braces for alignments
11592                     table.insert(buffer, "{")
11593                 end
11594                 table.insert(buffer, column)
11595                 if i > 1 then
11596                     table.insert(buffer, "}")
11597                 end
11598             end
11599         end

```

```

11599         table.insert(buffer, "}")
11600     end
11601     if attributes ~= nil then
11602         table.insert(buffer,
11603             "\\markdownRendererTableAttributeContextEnd{")
11604     end
11605     return buffer
11606 end
11607 end, extend_reader = function(self)
11608     local parsers = self.parsers
11609     local writer = self.writer
11610
11611     local table_hline_separator = parsers.pipe + parsers.plus
11612
11613     local table_hline_column = (parsers.dash
11614         - #(parsers.dash
11615             * (parsers.spacechar
11616                 + table_hline_separator
11617                 + parsers.newline)))^1
11618     * (parsers.colon * Cc("r")
11619         + parsers.dash * Cc("d"))
11620     + parsers.colon
11621     * (parsers.dash
11622         - #(parsers.dash
11623             * (parsers.spacechar
11624                 + table_hline_separator
11625                 + parsers.newline)))^1
11626     * (parsers.colon * Cc("c")
11627         + parsers.dash * Cc("l"))
11628
11629     local table_hline = pipe_table_row(false
11630         , table_hline_column
11631         , table_hline_separator
11632         , table_hline_column)
11633
11634     local table_caption_beginning
11635     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
11636         * parsers.optionalspace * parsers.newline)^0
11637     * parsers.check_minimal_indent_and_trail
11638     * (P("Table")^-1 * parsers.colon)
11639     * parsers.optionalspace
11640
11641     local function strip_trailing_spaces(s)
11642         return s:gsub("%s*$", "")
11643     end
11644
11645     local table_row

```

```

11646     = pipe_table_row(true
11647         , (C((parsers.linechar - parsers.pipe)^1)
11648           / strip_trailing_spaces
11649           / self.parser_functions.parse_inlines)
11650         , parsers.pipe
11651         , (C((parsers.linechar - parsers.pipe)^0)
11652           / strip_trailing_spaces
11653           / self.parser_functions.parse_inlines))
11654
11655     local table_caption
11656     if table_captions then
11657         table_caption = #table_caption_beginning
11658             * table_caption_beginning
11659         if table_attributes then
11660             table_caption = table_caption
11661                 * (C(((( parsers.linechar
11662                   - (parsers.attributes
11663                     * parsers.optionalspace
11664                     * parsers.newline
11665                     * -( parsers.optionalspace
11666                       * parsers.linechar))))
11667                   + ( parsers.newline
11668                     * #( parsers.optionalspace
11669                       * parsers.linechar)
11670                     * C(parsers.optionalspace)
11671                       / writer.space))
11672                   * (parsers.linechar
11673                     - parsers.lbrace)^0)^1)
11674                   / self.parser_functions.parse_inlines)
11675                 * (parsers.newline
11676                   + ( Ct(parsers.attributes)
11677                     * parsers.optionalspace
11678                     * parsers.newline))
11679         else
11680             table_caption = table_caption
11681                 * C(( parsers.linechar
11682                   + ( parsers.newline
11683                     * #( parsers.optionalspace
11684                       * parsers.linechar)
11685                     * C(parsers.optionalspace)
11686                       / writer.space))^1)
11687                   / self.parser_functions.parse_inlines
11688                 * parsers.newline
11689         end
11690     else
11691         table_caption = parsers.fail
11692     end

```

```

11693
11694     local PipeTable
11695         = Ct( table_row * parsers.newline
11696             * (parsers.check_minimal_indent_and_trail / {})
11697             * table_hline * parsers.newline
11698             * ( (parsers.check_minimal_indent / {})
11699               * table_row * parsers.newline)^0)
11700         / make_pipe_table_rectangular
11701         * table_caption^-1
11702         / writer.table
11703
11704     self.insert_pattern("Block after Blockquote",
11705                       PipeTable, "PipeTable")
11706 end
11707 }
11708 end

```

3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

11709 M.extensions.raw_inline = function()
11710     return {
11711         name = "built-in raw_inline syntax extension",
11712         extend_writer = function(self)
11713             local options = self.options
11714

```

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

```

11715         function self.rawInline(s, attr)
11716             if not self.is_writing then return "" end
11717             if self.flatten_inlines then return s end
11718             local name = util.cache_verbatim(options.cacheDir, s)
11719             return {"\\markdownRendererInputRawInline{" ,
11720                 name,"}{" , self.string(attr),"}"}
11721         end
11722     end, extend_reader = function(self)
11723         local writer = self.writer
11724
11725         local RawInline = parsers.inticks
11726             * parsers.raw_attribute
11727             / writer.rawInline
11728
11729         self.insert_pattern("Inline before Code",
11730                           RawInline, "RawInline")
11731     end
11732 }

```

```
11733 end
```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
11734 M.extensions.strike_through = function()
11735   return {
11736     name = "built-in strike_through syntax extension",
11737     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.

```
11738       function self.strike_through(s)
11739         if self.flatten_inlines then return s end
11740         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
11741       end
11742     end, extend_reader = function(self)
11743       local parsers = self.parsers
11744       local writer = self.writer
11745
11746       local StrikeThrough = (
11747         parsers.between(parsers.Inline, parsers.doubletildes,
11748           parsers.doubletildes)
11749       ) / writer.strike_through
11750
11751       self.insert_pattern("Inline after LinkAndEmph",
11752         StrikeThrough, "StrikeThrough")
11753
11754       self.add_special_character("~")
11755     end
11756   }
11757 end
```

3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
11758 M.extensions.subscripts = function()
11759   return {
11760     name = "built-in subscripts syntax extension",
11761     extend_writer = function(self)
```

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

```
11762       function self.subscript(s)
11763         if self.flatten_inlines then return s end
11764         return {"\\markdownRendererSubscript{" ,s,"}"}

```

```

11765     end
11766 end, extend_reader = function(self)
11767     local parsers = self.parsers
11768     local writer = self.writer
11769
11770     local Subscript = (
11771         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
11772     ) / writer.subscript
11773
11774     self.insert_pattern("Inline after LinkAndEmph",
11775                         Subscript, "Subscript")
11776
11777     self.add_special_character("~")
11778 end
11779 }
11780 end

```

3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

11781 M.extensions.superscripts = function()
11782     return {
11783         name = "built-in superscripts syntax extension",
11784         extend_writer = function(self)

```

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

```

11785         function self.superscript(s)
11786             if self.flatten_inlines then return s end
11787             return {"\\markdownRendererSuperscript{" ,s,"}"}
11788         end
11789     end, extend_reader = function(self)
11790         local parsers = self.parsers
11791         local writer = self.writer
11792
11793         local Superscript = (
11794             parsers.between(parsers.Str, parsers.circumflex,
11795                             parsers.circumflex)
11796         ) / writer.superscript
11797
11798         self.insert_pattern("Inline after LinkAndEmph",
11799                             Superscript, "Superscript")
11800
11801         self.add_special_character("^")
11802     end
11803 }
11804 end

```

3.1.7.19 T_EX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
11805 M.extensions.tex_math = function(tex_math_dollars,  
11806                                     tex_math_single_backslash,  
11807                                     tex_math_double_backslash)  
11808   return {  
11809     name = "built-in tex_math syntax extension",  
11810     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.

```
11811     function self.display_math(s)  
11812       if self.flatten_inlines then return s end  
11813       return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}  
11814     end
```

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

```
11815     function self.inline_math(s)  
11816       if self.flatten_inlines then return s end  
11817       return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}  
11818     end  
11819   end, extend_reader = function(self)  
11820     local parsers = self.parsers  
11821     local writer = self.writer  
11822  
11823     local function between(p, starter, ender)  
11824       return (starter * Cs(p * (p - ender)^0) * ender)  
11825     end  
11826  
11827     local function strip_preceding_whitespaces(str)  
11828       return str:gsub("^%s*(.-)$", "%1")  
11829     end  
11830  
11831     local allowed_before_closing  
11832       = B( parsers.backslash * parsers.any  
11833         + parsers.any * (parsers.any - parsers.backslash))  
11834  
11835     local allowed_before_closing_no_space  
11836       = B( parsers.backslash * parsers.any  
11837         + parsers.any * (parsers.nonspacechar - parsers.backslash))  
11838
```

The following patterns implement the Pandoc dollar math syntax extension.

```
11839     local dollar_math_content  
11840       = (parsers.newline * (parsers.check_optional_indent / ""))  
11841       + parsers.backslash^-1
```



```

11842     * parsers.linechar)
11843     - parsers.blankline^2
11844     - parsers.dollar
11845
11846     local inline_math_opening_dollars = parsers.dollar
11847                                     * #(parsers.nonspacechar)
11848
11849     local inline_math_closing_dollars
11850     = allowed_before_closing_no_space
11851     * parsers.dollar
11852     * -#(parsers.digit)
11853
11854     local inline_math_dollars = between(Cs( dollar_math_content),
11855                                       inline_math_opening_dollars,
11856                                       inline_math_closing_dollars)
11857
11858     local display_math_opening_dollars = parsers.dollar
11859                                       * parsers.dollar
11860
11861     local display_math_closing_dollars = parsers.dollar
11862                                       * parsers.dollar
11863
11864     local display_math_dollars = between(Cs( dollar_math_content),
11865                                       display_math_opening_dollars,
11866                                       display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

11867     local backslash_math_content
11868     = (parsers.newline * (parsers.check_optional_indent / ""))
11869     + parsers.linechar)
11870     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

11871     local inline_math_opening_double = parsers.backslash
11872                                     * parsers.backslash
11873                                     * parsers.lparent
11874
11875     local inline_math_closing_double = allowed_before_closing
11876     * parsers.spacechar^0
11877     * parsers.backslash
11878     * parsers.backslash
11879     * parsers.rparent
11880
11881     local inline_math_double = between(Cs( backslash_math_content),
11882                                       inline_math_opening_double,
11883                                       inline_math_closing_double)

```

```

11884         / strip_preceding_whitespaces
11885
11886     local display_math_opening_double = parsers.backslash
11887         * parsers.backslash
11888         * parsers.lbracket
11889
11890     local display_math_closing_double = allowed_before_closing
11891         * parsers.spacechar^0
11892         * parsers.backslash
11893         * parsers.backslash
11894         * parsers.rbracket
11895
11896     local display_math_double = between(Cs( backslash_math_content),
11897         display_math_opening_double,
11898         display_math_closing_double)
11899         / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

11900     local inline_math_opening_single = parsers.backslash
11901         * parsers.lparent
11902
11903     local inline_math_closing_single = allowed_before_closing
11904         * parsers.spacechar^0
11905         * parsers.backslash
11906         * parsers.rparent
11907
11908     local inline_math_single = between(Cs( backslash_math_content),
11909         inline_math_opening_single,
11910         inline_math_closing_single)
11911         / strip_preceding_whitespaces
11912
11913     local display_math_opening_single = parsers.backslash
11914         * parsers.lbracket
11915
11916     local display_math_closing_single = allowed_before_closing
11917         * parsers.spacechar^0
11918         * parsers.backslash
11919         * parsers.rbracket
11920
11921     local display_math_single = between(Cs( backslash_math_content),
11922         display_math_opening_single,
11923         display_math_closing_single)
11924         / strip_preceding_whitespaces
11925
11926     local display_math = parsers.fail
11927
11928     local inline_math = parsers.fail
11929

```

```

11930     if tex_math_dollars then
11931         display_math = display_math + display_math_dollars
11932         inline_math = inline_math + inline_math_dollars
11933     end
11934
11935     if tex_math_double_backslash then
11936         display_math = display_math + display_math_double
11937         inline_math = inline_math + inline_math_double
11938     end
11939
11940     if tex_math_single_backslash then
11941         display_math = display_math + display_math_single
11942         inline_math = inline_math + inline_math_single
11943     end
11944
11945     local TexMath = display_math / writer.display_math
11946                   + inline_math / writer.inline_math
11947
11948     self.insert_pattern("Inline after LinkAndEmph",
11949                       TexMath, "TexMath")
11950
11951     if tex_math_dollars then
11952         self.add_special_character("$")
11953     end
11954
11955     if tex_math_single_backslash or tex_math_double_backslash then
11956         self.add_special_character("\\")
11957         self.add_special_character("[")
11958         self.add_special_character("]")
11959         self.add_special_character("(")
11960         self.add_special_character("(")
11961     end
11962 end
11963 }
11964 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.

```

11965 M.extensions.jekyll_data = function(expect_jekyll_data,
11966                                     ensure_jekyll_data)
11967     return {

```

```

11968     name = "built-in jekyll_data syntax extension",
11969     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.

```

11970     function self.jekyllData(d, t, p)
11971         if not self.is_writing then return "" end
11972
11973         local buf = {}
11974
11975         local keys = {}
11976         for k, _ in pairs(d) do
11977             table.insert(keys, k)
11978         end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

11979         table.sort(keys, function(first, second)
11980             if type(first) ~= type(second) then
11981                 return type(first) < type(second)
11982             else
11983                 return first < second
11984             end
11985         end)
11986
11987         if not p then
11988             table.insert(buf, "\\markdownRendererJekyllDataBegin")
11989         end
11990
11991         local is_sequence = false
11992         if #d > 0 and #d == #keys then
11993             for i=1, #d do
11994                 if d[i] == nil then
11995                     goto not_a_sequence
11996                 end
11997             end
11998             is_sequence = true
11999         end
12000         ::not_a_sequence::
12001
12002         if is_sequence then
12003             table.insert(buf,
12004                 "\\markdownRendererJekyllDataSequenceBegin{")

```

```

12005         table.insert(buf, self.identified(p or "null"))
12006         table.insert(buf, "}{")
12007         table.insert(buf, #keys)
12008         table.insert(buf, "}")
12009     else
12010         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12011         table.insert(buf, self.identified(p or "null"))
12012         table.insert(buf, "}{")
12013         table.insert(buf, #keys)
12014         table.insert(buf, "}")
12015     end
12016
12017     for _, k in ipairs(keys) do
12018         local v = d[k]
12019         local typ = type(v)
12020         k = tostring(k or "null")
12021         if typ == "table" and next(v) ~= nil then
12022             table.insert(
12023                 buf,
12024                 self.jekyllData(v, t, k)
12025             )
12026         else
12027             k = self.identified(k)
12028             v = tostring(v)
12029             if typ == "boolean" then
12030                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12031                 table.insert(buf, k)
12032                 table.insert(buf, "}{")
12033                 table.insert(buf, v)
12034                 table.insert(buf, "}")
12035             elseif typ == "number" then
12036                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12037                 table.insert(buf, k)
12038                 table.insert(buf, "}{")
12039                 table.insert(buf, v)
12040                 table.insert(buf, "}")
12041             elseif typ == "string" then
12042                 table.insert(buf,
12043                     "\\markdownRendererJekyllDataProgrammaticString{")
12044                 table.insert(buf, k)
12045                 table.insert(buf, "}{")
12046                 table.insert(buf, self.identified(v))
12047                 table.insert(buf, "}")
12048                 table.insert(buf,
12049                     "\\markdownRendererJekyllDataTypographicString{")
12050                 table.insert(buf, k)
12051                 table.insert(buf, "}{")

```

```

12052         table.insert(buf, t(v))
12053         table.insert(buf, "}")
12054     elseif typ == "table" then
12055         table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12056         table.insert(buf, k)
12057         table.insert(buf, "}")
12058     else
12059         local error = self.error(format(
12060             "Unexpected type %s for value of "
12061             .. "YAML key %s.", typ, k))
12062         table.insert(buf, error)
12063     end
12064 end
12065 end
12066
12067 if is_sequence then
12068     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12069 else
12070     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12071 end
12072
12073 if not p then
12074     table.insert(buf, "\\markdownRendererJekyllDataEnd")
12075 end
12076
12077 return buf
12078 end
12079 end, extend_reader = function(self)
12080     local parsers = self.parsers
12081     local writer = self.writer
12082
12083     local JekyllData
12084     = Cmt( C((parsers.line - P("---") - P("..."))^0)
12085         , function(s, i, text) -- luacheck: ignore s i
12086             local data
12087             local ran_ok, _ = pcall(function()
12088                 -- TODO: Use `require("tinyyaml")` in TeX Live 2023
12089                 local tinyyaml = require("markdown-tinyyaml")
12090                 data = tinyyaml.parse(text, {timestamps=false})
12091             end)
12092             if ran_ok and data ~= nil then
12093                 return true, writer.jekyllData(data, function(s)
12094                     return self.parser_functions.parse_blocks_nested(s)
12095                 end, nil)
12096             else
12097                 return false
12098             end
12099         end)

```

```

12099         end
12100     )
12101
12102     local UnexpectedJekyllData
12103     = P("----")
12104     * parsers.blankline / 0
12105     -- if followed by blank, it's thematic break
12106     * #(-parsers.blankline)
12107     * JekyllData
12108     * (P("----") + P("..."))
12109
12110     local ExpectedJekyllData
12111     = ( P("----")
12112     * parsers.blankline / 0
12113     -- if followed by blank, it's thematic break
12114     * #(-parsers.blankline)
12115     )^-1
12116     * JekyllData
12117     * (P("----") + P("..."))^-1
12118
12119     if ensure_jekyll_data then
12120         ExpectedJekyllData = ExpectedJekyllData
12121         * parsers.eof
12122     else
12123         ExpectedJekyllData = ( ExpectedJekyllData
12124         * (V("Blank")^0 / writer.interblocksep)
12125         )^-1
12126     end
12127
12128     self.insert_pattern("Block before Blockquote",
12129         UnexpectedJekyllData, "UnexpectedJekyllData")
12130     if expect_jekyll_data then
12131         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12132     end
12133 end
12134 }
12135 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 own 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`.

```
12136 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```

12137 options = options or {}
12138 setmetatable(options, { __index = function (_, key)
12139     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.

```

12140 local parser_convert = nil
12141 return function(input)
12142     local function convert(input)
12143         if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

12144             local parser = require("markdown-parser")
12145             if metadata.version ~= parser.metadata.version then
12146                 warn("markdown.lua " .. metadata.version .. " used with " ..
12147                     "markdown-parser.lua " .. parser.metadata.version .. ".")
12148             end
12149             parser_convert = parser.new(options)
12150         end
12151         return parser_convert(input)
12152     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).

```

12153     local output
12154     if options.eagerCache or options.finalizeCache then
12155         local salt = util.salt(options)
12156         local name = util.cache(options.cacheDir, input, salt, convert,
12157                                 ".md.tex")
12158         output = [[\input{}}] .. name .. [{}\relax]]

```

Otherwise, return the result of the conversion directly.

```

12159     else
12160         output = convert(input)
12161     end

```

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

```

12162     if options.finalizeCache then
12163         local file, mode
12164         if options.frozenCacheCounter > 0 then
12165             mode = "a"

```



```

12166     else
12167         mode = "w"
12168     end
12169     file = assert(io.open(options.frozenCacheFileName, mode),
12170         [[Could not open file "]] .. options.frozenCacheFileName
12171         .. [[ " for writing]])
12172     assert(file:write(
12173         [[\expandafter\global\expandafter\def\csname ]]
12174         .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12175         .. [[\endcsname{}]] .. output .. [[]]] .. "\n"))
12176     assert(file:close())
12177     end
12178     return output
12179 end
12180 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain \TeX output. See Section 2.1.1.

```

12181 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

12182     options = options or {}
12183     setmetatable(options, { __index = function (_, key)
12184         return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12185     if options.singletonCache and singletonCache.convert then
12186         for k, v in pairs(defaultOptions) do
12187             if type(v) == "table" then
12188                 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12189                     if singletonCache.options[k][i] ~= options[k][i] then
12190                         goto miss
12191                     end
12192                 end

```

The `cacheDir` option is disregarded.

```

12193         elseif k ~= "cacheDir"
12194             and singletonCache.options[k] ~= options[k] then
12195             goto miss
12196         end
12197     end
12198     return singletonCache.convert
12199 end
12200 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

12201     local extensions = {}

```

```

12202
12203 if options.bracketedSpans then
12204     local bracketed_spans_extension = M.extensions.bracketed_spans()
12205     table.insert(extensions, bracketed_spans_extension)
12206 end
12207
12208 if options.contentBlocks then
12209     local content_blocks_extension = M.extensions.content_blocks(
12210         options.contentBlocksLanguageMap)
12211     table.insert(extensions, content_blocks_extension)
12212 end
12213
12214 if options.definitionLists then
12215     local definition_lists_extension = M.extensions.definition_lists(
12216         options.tightLists)
12217     table.insert(extensions, definition_lists_extension)
12218 end
12219
12220 if options.fencedCode then
12221     local fenced_code_extension = M.extensions.fenced_code(
12222         options.blankBeforeCodeFence,
12223         options.fencedCodeAttributes,
12224         options.rawAttribute)
12225     table.insert(extensions, fenced_code_extension)
12226 end
12227
12228 if options.fencedDivs then
12229     local fenced_div_extension = M.extensions.fenced_divs(
12230         options.blankBeforeDivFence)
12231     table.insert(extensions, fenced_div_extension)
12232 end
12233
12234 if options.headerAttributes then
12235     local header_attributes_extension = M.extensions.header_attributes()
12236     table.insert(extensions, header_attributes_extension)
12237 end
12238
12239 if options.inlineCodeAttributes then
12240     local inline_code_attributes_extension =
12241         M.extensions.inline_code_attributes()
12242     table.insert(extensions, inline_code_attributes_extension)
12243 end
12244
12245 if options.jekyllData then
12246     local jekyll_data_extension = M.extensions.jekyll_data(
12247         options.expectJekyllData, options.ensureJekyllData)
12248     table.insert(extensions, jekyll_data_extension)

```

```

12249 end
12250
12251 if options.linkAttributes then
12252     local link_attributes_extension =
12253         M.extensions.link_attributes()
12254     table.insert(extensions, link_attributes_extension)
12255 end
12256
12257 if options.lineBlocks then
12258     local line_block_extension = M.extensions.line_blocks()
12259     table.insert(extensions, line_block_extension)
12260 end
12261
12262 if options.mark then
12263     local mark_extension = M.extensions.mark()
12264     table.insert(extensions, mark_extension)
12265 end
12266
12267 if options.pipeTables then
12268     local pipe_tables_extension = M.extensions.pipe_tables(
12269         options.tableCaptions, options.tableAttributes)
12270     table.insert(extensions, pipe_tables_extension)
12271 end
12272
12273 if options.rawAttribute then
12274     local raw_inline_extension = M.extensions.raw_inline()
12275     table.insert(extensions, raw_inline_extension)
12276 end
12277
12278 if options.strikeThrough then
12279     local strike_through_extension = M.extensions.strike_through()
12280     table.insert(extensions, strike_through_extension)
12281 end
12282
12283 if options.subscripts then
12284     local subscript_extension = M.extensions.subscripts()
12285     table.insert(extensions, subscript_extension)
12286 end
12287
12288 if options.superscripts then
12289     local superscript_extension = M.extensions.superscripts()
12290     table.insert(extensions, superscript_extension)
12291 end
12292
12293 if options.texMathDollars or
12294     options.texMathSingleBackslash or
12295     options.texMathDoubleBackslash then

```

```

12296     local tex_math_extension = M.extensions.tex_math(
12297         options.texMathDollars,
12298         options.texMathSingleBackslash,
12299         options.texMathDoubleBackslash)
12300     table.insert(extensions, tex_math_extension)
12301 end
12302
12303 if options.notes or options.inlineNotes then
12304     local notes_extension = M.extensions.notes(
12305         options.notes, options.inlineNotes)
12306     table.insert(extensions, notes_extension)
12307 end
12308
12309 if options.citations then
12310     local citations_extension
12311         = M.extensions.citations(options.citationNbsps)
12312     table.insert(extensions, citations_extension)
12313 end
12314
12315 if options.fancyLists then
12316     local fancy_lists_extension = M.extensions.fancy_lists()
12317     table.insert(extensions, fancy_lists_extension)
12318 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12319 for _, user_extension_filename in ipairs(options.extensions) do
12320     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

12321     local pathname = assert(kpse.find_file(filename),
12322         [[Could not locate user-defined syntax extension "]]
12323         .. filename)
12324     local input_file = assert(io.open(pathname, "r"),
12325         [[Could not open user-defined syntax extension "]]
12326         .. pathname .. [{" for reading}]]
12327     local input = assert(input_file:read("*a"))
12328     assert(input_file:close())
12329     local user_extension, err = load([[
12330         local sandbox = {}
12331         setmetatable(sandbox, {__index = _G})
12332         _ENV = sandbox
12333     ]] .. input)()
12334     assert(user_extension,
12335         [[Failed to compile user-defined syntax extension "]]
12336         .. pathname .. [{": }]] .. (err or [{"}]))

```

Then, validate the user-defined syntax extension.

```

12337     assert(user_extension.api_version ~= nil,

```

```

12338     [[User-defined syntax extension "]] .. pathname
12339     .. [[ " does not specify mandatory field "api_version" ]])
12340     assert(type(user_extension.api_version) == "number",
12341           [[User-defined syntax extension "]] .. pathname
12342           .. [[ " specifies field "api_version" of type "]]
12343           .. type(user_extension.api_version)
12344           .. [[ " but "number" was expected ]])
12345     assert(user_extension.api_version > 0
12346           and user_extension.api_version
12347           <= metadata.user_extension_api_version,
12348           [[User-defined syntax extension "]] .. pathname
12349           .. [[ " uses syntax extension API version "]]
12350           .. user_extension.api_version .. [[ but markdown.lua ]]
12351           .. metadata.version .. [[ uses API version ]]
12352           .. metadata.user_extension_api_version
12353           .. [[, which is incompatible]])
12354
12355     assert(user_extension.grammar_version ~= nil,
12356           [[User-defined syntax extension "]] .. pathname
12357           .. [[ " does not specify mandatory field "grammar_version" ]])
12358     assert(type(user_extension.grammar_version) == "number",
12359           [[User-defined syntax extension "]] .. pathname
12360           .. [[ " specifies field "grammar_version" of type "]]
12361           .. type(user_extension.grammar_version)
12362           .. [[ " but "number" was expected ]])
12363     assert(user_extension.grammar_version == metadata.grammar_version,
12364           [[User-defined syntax extension "]] .. pathname
12365           .. [[ " uses grammar version "]]
12366           .. user_extension.grammar_version
12367           .. [[ but markdown.lua ]] .. metadata.version
12368           .. [[ uses grammar version ]] .. metadata.grammar_version
12369           .. [[, which is incompatible]])
12370
12371     assert(user_extension.finalize_grammar ~= nil,
12372           [[User-defined syntax extension "]] .. pathname
12373           .. [[ " does not specify mandatory "finalize_grammar" field]])
12374     assert(type(user_extension.finalize_grammar) == "function",
12375           [[User-defined syntax extension "]] .. pathname
12376           .. [[ " specifies field "finalize_grammar" of type "]]
12377           .. type(user_extension.finalize_grammar)
12378           .. [[ " 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.)

```

12379     local extension = {
12380       name = [[user-defined "]] .. pathname .. [[ " syntax extension]],
12381       extend_reader = user_extension.finalize_grammar,

```

```

12382     extend_writer = function() end,
12383   }
12384   return extension
12385 end)(user_extension_filename)
12386 table.insert(extensions, user_extension)
12387 end

```

Produce a conversion function from markdown to plain \TeX .

```

12388 local writer = M.writer.new(options)
12389 local reader = M.reader.new(writer, options)
12390 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`.

```

12391 collectgarbage("collect")

```

Update the singleton cache.

```

12392 if options.singletonCache then
12393   local singletonCacheOptions = {}
12394   for k, v in pairs(options) do
12395     singletonCacheOptions[k] = v
12396   end
12397   setmetatable(singletonCacheOptions,
12398     { __index = function (_, key)
12399       return defaultOptions[key] end })
12400   singletonCache.options = singletonCacheOptions
12401   singletonCache.convert = convert
12402 end

```

Return the conversion function from markdown to plain \TeX .

```

12403 return convert
12404 end
12405 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.

```

12406
12407 local input
12408 if input_filename then
12409   local input_file = assert(io.open(input_filename, "r"),
12410     [[Could not open file ]] .. input_filename .. [[ for reading]])
12411   input = assert(input_file:read("*a"))
12412   assert(input_file:close())
12413 else
12414   input = assert(io.read("*a"))
12415 end

```

12416

First, ensure that the `options.cacheDir` directory exists.

```
12417 local lfs = require("lfs")
12418 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12419     assert(lfs.mkdir(options["cacheDir"]))
12420 end
```

If Kpathsea has not been loaded before or if Lua \TeX has not yet been initialized, configure Kpathsea on top of loading it.

```
12421 local kpse
12422 (function()
12423     local should_initialize = package.loaded.kpse == nil
12424                             or tex.initialize ~= nil
12425     kpse = require("kpse")
12426     if should_initialize then
12427         kpse.set_program_name("luatex")
12428     end
12429 end)()
12430 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.

```
12431 if metadata.version ~= md.metadata.version then
12432     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12433         "markdown.lua " .. md.metadata.version .. ".")
12434 end
12435 local convert = md.new(options)
12436 local output = convert(input)
12437
12438 if output_filename then
12439     local output_file = assert(io.open(output_filename, "w"),
12440     [[Could not open file ]] .. output_filename .. [[ for writing]])
12441     assert(output_file:write(output))
12442     assert(output_file:close())
12443 else
12444     assert(io.write(output))
12445 end
```

Remove the `options.cacheDir` directory if it is empty.

```
12446 if options.cacheDir then
12447     lfs.rmdir(options.cacheDir)
12448 end
```

3.2 Plain \TeX Implementation

The plain \TeX implementation provides macros for the interfacing between \TeX and Lua and for the buffering of input text. These macros are then used to implement

the macros for the conversion from markdown to plain T_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```
12449 \ExplSyntaxOn
12450 \cs_if_free:NT
12451   \markdownInfo
12452   {
12453     \cs_new:Npn
12454       \markdownInfo #1
12455       {
12456         \msg_info:nne
12457           { markdown }
12458           { generic-message }
12459           { #1 }
12460       }
12461   }
12462 \cs_if_free:NT
12463   \markdownWarning
12464   {
12465     \cs_new:Npn
12466       \markdownWarning #1
12467       {
12468         \msg_warning:nne
12469           { markdown }
12470           { generic-message }
12471           { #1 }
12472       }
12473   }
12474 \cs_if_free:NT
12475   \markdownError
12476   {
12477     \cs_new:Npn
12478       \markdownError #1 #2
12479       {
12480         \msg_error:nnee
12481           { markdown }
12482           { generic-message-with-help-text }
12483           { #1 }
12484           { #2 }
12485       }
12486   }
12487 \msg_new:nnn
12488   { markdown }
12489   { generic-message }
12490   { #1 }
```



```

12491 \msg_new:nnnn
12492   { markdown }
12493   { generic-message-with-help-text }
12494   { #1 }
12495   { #2 }
12496 \cs_generate_variant:Nn
12497   \msg_info:nnn
12498   { nne }
12499 \cs_generate_variant:Nn
12500   \msg_warning:nnn
12501   { nne }
12502 \cs_generate_variant:Nn
12503   \msg_error:nnnn
12504   { nnee }
12505 \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_EX themes provided with the Markdown package.

```

12506 \ExplSyntaxOn
12507 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
12508 \cs_new:Nn
12509   \@@_plain_tex_load_theme:nn
12510   {
12511     \prop_get:NnNTF
12512       \g_@@_plain_tex_loaded_themes_linenos_prop
12513       { #1 }
12514       \l_tmpa_tl
12515       {
12516         \msg_warning:nnnV
12517           { markdown }
12518           { repeatedly-loaded-plain-tex-theme }
12519           { #1 }
12520         \l_tmpa_tl
12521       }
12522     {
12523       \msg_info:nnn
12524         { markdown }
12525         { loading-plain-tex-theme }
12526         { #1 }
12527       \prop_gput:Nnx
12528         \g_@@_plain_tex_loaded_themes_linenos_prop
12529         { #1 }
12530         { \tex_the:D \tex_inputlineno:D }
12531       \file_input:n

```

```

12532         { markdown theme #2 }
12533     }
12534 }
12535 \msg_new:nnn
12536 { markdown }
12537 { loading-plain-tex-theme }
12538 { Loading~plain~TeX~Markdown~theme~#1 }
12539 \msg_new:nnn
12540 { markdown }
12541 { repeatedly-loaded-plain-tex-theme }
12542 {
12543     Plain~TeX~Markdown~theme~#1~was~previously~
12544     loaded~on~line~#2,~not~loading~it~again
12545 }
12546 \cs_generate_variant:Nn
12547 \prop_gput:Nnn
12548 { Nnx }
12549 \cs_gset_eq:NN
12550 \@@_load_theme:nn
12551 \@@_plain_tex_load_theme:nn
12552 \cs_generate_variant:Nn
12553 \@@_load_theme:nn
12554 { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain \TeX theme from within themes for higher-level \TeX formats such as \LaTeX and \ConTeXt .

```

12555 \cs_new:Npn
12556 \markdownLoadPlainTeXTheme
12557 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

12558 \tl_set:NV
12559 \l_tmpa_tl
12560 \g_@@_current_theme_tl
12561 \tl_reverse:N
12562 \l_tmpa_tl
12563 \tl_set:Ne
12564 \l_tmpb_tl
12565 {
12566     \tl_tail:V
12567     \l_tmpa_tl
12568 }
12569 \tl_reverse:N
12570 \l_tmpb_tl

```

Next, we munge the theme name.

```

12571 \str_set:NV
12572 \l_tmpa_str
12573 \l_tmpb_tl
12574 \str_replace_all:Nnn
12575 \l_tmpa_str
12576 { / }
12577 { _ }

```

Finally, we load the plain TeX theme.

```

12578 \@@_plain_tex_load_theme:VV
12579 \l_tmpb_tl
12580 \l_tmpa_str
12581 }
12582 \cs_generate_variant:Nn
12583 \tl_set:Nn
12584 { Ne }
12585 \cs_generate_variant:Nn
12586 \@@_plain_tex_load_theme:nn
12587 { VV }
12588 \ExplSyntaxOff

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

12589 \markdownSetup {
12590   rendererPrototypes = {
12591     tilde = {~},
12592   },
12593 }

```

The [witiko/markdown/defaults](#) plain TeX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

12594 \def\markdownRendererInterblockSeparatorPrototype{\par}%
12595 \def\markdownRendererParagraphSeparatorPrototype{%
12596   \markdownRendererInterblockSeparator}%
12597 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
12598 \def\markdownRendererSoftLineBreakPrototype{ }%
12599 \let\markdownRendererEllipsisPrototype\dots
12600 \def\markdownRendererNbspPrototype{~}%
12601 \def\markdownRendererLeftBracePrototype{\char`\{}%
12602 \def\markdownRendererRightBracePrototype{\char`\}}%
12603 \def\markdownRendererDollarSignPrototype{\char`\$}%
12604 \def\markdownRendererPercentSignPrototype{\char`\}%}
12605 \def\markdownRendererAmpersandPrototype{\&}%
12606 \def\markdownRendererUnderscorePrototype{\char`\_}%

```



```

12654 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
12655 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
12656 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
12657 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=Opt}%
12658 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
12659 \def\markdownRendererInputVerbatimPrototype#1{%
12660   \par{\tt\input#1\relax{}}\par}%
12661 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
12662   \markdownRendererInputVerbatim{#1}}%
12663 \def\markdownRendererHeadingOnePrototype#1{#1}%
12664 \def\markdownRendererHeadingTwoPrototype#1{#1}%
12665 \def\markdownRendererHeadingThreePrototype#1{#1}%
12666 \def\markdownRendererHeadingFourPrototype#1{#1}%
12667 \def\markdownRendererHeadingFivePrototype#1{#1}%
12668 \def\markdownRendererHeadingSixPrototype#1{#1}%
12669 \def\markdownRendererThematicBreakPrototype{}%
12670 \def\markdownRendererNotePrototype#1{#1}%
12671 \def\markdownRendererCitePrototype#1{}%
12672 \def\markdownRendererTextCitePrototype#1{}%
12673 \def\markdownRendererTickedBoxPrototype{[X]}%
12674 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
12675 \def\markdownRendererUntickedBoxPrototype{[ ]}%
12676 \def\markdownRendererStrikeThroughPrototype#1{#1}%
12677 \def\markdownRendererSuperscriptPrototype#1{#1}%
12678 \def\markdownRendererSubscriptPrototype#1{#1}%
12679 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
12680 \def\markdownRendererInlineMathPrototype#1{$#1$}%
12681 \ExplSyntaxOn
12682 \cs_gset:Npn
12683   \markdownRendererHeaderAttributeContextBeginPrototype
12684   {
12685     \group_begin:
12686     \color_group_begin:
12687   }
12688 \cs_gset:Npn
12689   \markdownRendererHeaderAttributeContextEndPrototype
12690   {
12691     \color_group_end:
12692     \group_end:
12693   }
12694 \cs_gset_eq:NN
12695   \markdownRendererBracketedSpanAttributeContextBeginPrototype
12696   \markdownRendererHeaderAttributeContextBeginPrototype
12697 \cs_gset_eq:NN
12698   \markdownRendererBracketedSpanAttributeContextEndPrototype
12699   \markdownRendererHeaderAttributeContextEndPrototype
12700 \cs_gset_eq:NN

```

```

12701 \markdownRendererFencedDivAttributeContextBeginPrototype
12702 \markdownRendererHeaderAttributeContextBeginPrototype
12703 \cs_gset_eq:NN
12704 \markdownRendererFencedDivAttributeContextEndPrototype
12705 \markdownRendererHeaderAttributeContextEndPrototype
12706 \cs_gset_eq:NN
12707 \markdownRendererFencedCodeAttributeContextBeginPrototype
12708 \markdownRendererHeaderAttributeContextBeginPrototype
12709 \cs_gset_eq:NN
12710 \markdownRendererFencedCodeAttributeContextEndPrototype
12711 \markdownRendererHeaderAttributeContextEndPrototype
12712 \cs_gset:Npn
12713 \markdownRendererReplacementCharacterPrototype
12714 { \codepoint_str_generate:n { fffd } }
12715 \ExplSyntaxOff
12716 \def\markdownRendererSectionBeginPrototype{ }%
12717 \def\markdownRendererSectionEndPrototype{ }%
12718 \ExplSyntaxOn
12719 \cs_gset:Npn
12720 \markdownRendererWarningPrototype
12721 #1#2#3#4
12722 {
12723   \tl_set:Nn
12724     \l_tmpa_tl
12725     { #2 }
12726   \tl_if_empty:nF
12727     { #4 }
12728     {
12729       \tl_put_right:Nn
12730         \l_tmpa_tl
12731         { \iow_newline: #4 }
12732     }
12733   \exp_args:NV
12734     \markdownWarning
12735     \l_tmpa_tl
12736 }
12737 \ExplSyntaxOff
12738 \def\markdownRendererErrorPrototype#1#2#3#4{%
12739   \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.

```

12740 \ExplSyntaxOn
12741 \cs_new:Nn

```

```

12742 \@@_plain_tex_default_input_raw_inline:nn
12743 {
12744   \str_case:nn
12745     { #2 }
12746     {
12747       { md } { \markdownInput{#1} }
12748       { tex } { \markdownEscape{#1} \unskip }
12749     }
12750 }
12751 \cs_new:Nn
12752 \@@_plain_tex_default_input_raw_block:nn
12753 {
12754   \str_case:nn
12755     { #2 }
12756     {
12757       { md } { \markdownInput{#1} }
12758       { tex } { \markdownEscape{#1} }
12759     }
12760 }
12761 \cs_gset:Npn
12762 \markdownRendererInputRawInlinePrototype#1#2
12763 {
12764   \@@_plain_tex_default_input_raw_inline:nn
12765     { #1 }
12766     { #2 }
12767 }
12768 \cs_gset:Npn
12769 \markdownRendererInputRawBlockPrototype#1#2
12770 {
12771   \@@_plain_tex_default_input_raw_block:nn
12772     { #1 }
12773     { #2 }
12774 }
12775 \ExplSyntaxOff

```

3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position p :

`\c_@@_jekyll_data_sequence_t1` The currently traversed branch of the YAML document contains a sequence at depth p .

`\c_@@_jekyll_data_mapping_t1` 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 .

```
12776 \ExplSyntaxOn
12777 \seq_new:N \g_@@_jekyll_data_datatypes_seq
12778 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
12779 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
12780 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }
```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```
12781 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
12782 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
12783 {
12784   \seq_if_empty:NF
12785     \g_@@_jekyll_data_datatypes_seq
12786     {
12787       \seq_get_right:NN
12788         \g_@@_jekyll_data_datatypes_seq
12789         \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.

```
12790   \str_if_eq:NNTF
12791     \l_tmpa_tl
12792     \c_@@_jekyll_data_sequence_tl
12793     {
12794       \seq_put_right:Nn
12795         \g_@@_jekyll_data_wildcard_absolute_address_seq
12796         { * }
12797     }
12798     {
12799       \seq_put_right:Nn
12800         \g_@@_jekyll_data_wildcard_absolute_address_seq
12801         { #1 }
12802     }
12803   }
12804 }
```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual

keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```
12805 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
12806 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
12807 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
12808   {
12809     \seq_pop_left:NN #1 \l_tmpa_tl
12810     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
12811     \seq_put_left:NV #1 \l_tmpa_tl
12812   }
12813 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
12814   {
12815     \markdown_jekyll_data_concatenate_address:NN
12816     \g_@@_jekyll_data_wildcard_absolute_address_seq
12817     \g_@@_jekyll_data_wildcard_absolute_address_tl
12818     \seq_get_right:NN
12819     \g_@@_jekyll_data_wildcard_absolute_address_seq
12820     \g_@@_jekyll_data_wildcard_relative_address_tl
12821   }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```
12822 \cs_new:Nn \markdown_jekyll_data_push:nN
12823   {
12824     \markdown_jekyll_data_push_address_segment:n
12825     { #1 }
12826     \seq_put_right:NV
```

```

12827     \g_@@_jekyll_data_datatypes_seq
12828     #2
12829     \markdown_jekyll_data_update_address_tls:
12830   }
12831   \cs_new:Nn \markdown_jekyll_data_pop:
12832     {
12833     \seq_pop_right:NN
12834     \g_@@_jekyll_data_wildcard_absolute_address_seq
12835     \l_tmpa_tl
12836     \seq_pop_right:NN
12837     \g_@@_jekyll_data_datatypes_seq
12838     \l_tmpa_tl
12839     \markdown_jekyll_data_update_address_tls:
12840   }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

12841   \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
12842     {
12843     \keys_set_known:nn
12844     { markdown/jekyllData }
12845     { { #1 } = { #2 } }
12846   }
12847   \cs_generate_variant:Nn
12848     \markdown_jekyll_data_set_keyval:nn
12849     { Vn }
12850   \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
12851     {
12852     \markdown_jekyll_data_push:nN
12853     { #1 }
12854     \c_@@_jekyll_data_scalar_tl
12855     \markdown_jekyll_data_set_keyval:Vn
12856     \g_@@_jekyll_data_wildcard_absolute_address_tl
12857     { #2 }
12858     \markdown_jekyll_data_set_keyval:Vn
12859     \g_@@_jekyll_data_wildcard_relative_address_tl
12860     { #2 }
12861     \markdown_jekyll_data_pop:
12862   }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

12863   \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
12864     \markdown_jekyll_data_push:nN
12865     { #1 }
12866     \c_@@_jekyll_data_sequence_tl
12867   }

```

```

12868 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
12869   \markdown_jekyll_data_push:nN
12870     { #1 }
12871     \c_@@_jekyll_data_mapping_tl
12872 }
12873 \def\markdownRendererJekyllDataSequenceEndPrototype{
12874   \markdown_jekyll_data_pop:
12875 }
12876 \def\markdownRendererJekyllDataMappingEndPrototype{
12877   \markdown_jekyll_data_pop:
12878 }
12879 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
12880   \markdown_jekyll_data_set_keyvals:nn
12881     { #1 }
12882     { #2 }
12883 }
12884 \def\markdownRendererJekyllDataEmptyPrototype#1{}
12885 \def\markdownRendererJekyllDataNumberPrototype#1#2{
12886   \markdown_jekyll_data_set_keyvals:nn
12887     { #1 }
12888     { #2 }
12889 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

12890 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
12891 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
12892   \markdown_jekyll_data_set_keyvals:nn
12893     { #1 }
12894     { #2 }
12895 }
12896 \ExplSyntaxOff

```

If plain $\text{T}_{\text{E}}\text{X}$ is the top layer, we load the [witiko/markdown/defaults](#) plain $\text{T}_{\text{E}}\text{X}$ theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

12897 \ExplSyntaxOn
12898 \str_if_eq:VVT
12899   \c_@@_top_layer_tl
12900   \c_@@_option_layer_plain_tex_tl
12901   {
12902     \ExplSyntaxOff
12903     \@@_if_option:nF
12904       { noDefaults }
12905       {
12906         \@@_setup:n
12907           {theme = witiko/markdown/defaults}
12908       }

```

```

12909 \ExplSyntaxOn
12910 }
12911 \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).

```

12912 \ExplSyntaxOn
12913 \tl_new:N \g_@@_formatted_lua_options_tl
12914 \cs_new:Nn \@@_format_lua_options:
12915 {
12916   \tl_gclear:N
12917     \g_@@_formatted_lua_options_tl
12918   \seq_map_function:NN
12919     \g_@@_lua_options_seq
12920     \@@_format_lua_option:n
12921 }
12922 \cs_new:Nn \@@_format_lua_option:n
12923 {
12924   \@@_typecheck_option:n
12925     { #1 }
12926   \@@_get_option_type:nN
12927     { #1 }
12928   \l_tmpa_tl
12929   \bool_case_true:nF
12930     {
12931       {
12932         \str_if_eq_p:VV
12933           \l_tmpa_tl
12934             \c_@@_option_type_boolean_tl ||
12935         \str_if_eq_p:VV
12936           \l_tmpa_tl
12937             \c_@@_option_type_number_tl ||
12938         \str_if_eq_p:VV
12939           \l_tmpa_tl
12940             \c_@@_option_type_counter_tl
12941       }
12942     {
12943       \@@_get_option_value:nN
12944         { #1 }
12945       \l_tmpa_tl
12946       \tl_gput_right:Nx
12947         \g_@@_formatted_lua_options_tl
12948         { #1~==~ \l_tmpa_tl ,~ }

```

```

12949     }
12950   {
12951     \str_if_eq_p:VV
12952     \l_tmpa_tl
12953     \c_@@_option_type_clist_tl
12954   }
12955   {
12956     \@@_get_option_value:nN
12957     { #1 }
12958     \l_tmpa_tl
12959     \tl_gput_right:Nx
12960     \g_@@_formatted_lua_options_tl
12961     { #1~::~\c_left_brace_str }
12962     \clist_map_inline:Vn
12963     \l_tmpa_tl
12964     {
12965       \@@_lua_escape:xN
12966       { ##1 }
12967       \l_tmpb_tl
12968       \tl_gput_right:Nn
12969       \g_@@_formatted_lua_options_tl
12970       { " }
12971       \tl_gput_right:NV
12972       \g_@@_formatted_lua_options_tl
12973       \l_tmpb_tl
12974       \tl_gput_right:Nn
12975       \g_@@_formatted_lua_options_tl
12976       { " ,~ }
12977     }
12978     \tl_gput_right:Nx
12979     \g_@@_formatted_lua_options_tl
12980     { \c_right_brace_str ,~ }
12981   }
12982 }
12983 {
12984   \@@_get_option_value:nN
12985   { #1 }
12986   \l_tmpa_tl
12987   \@@_lua_escape:xN
12988   { \l_tmpa_tl }
12989   \l_tmpb_tl
12990   \tl_gput_right:Nn
12991   \g_@@_formatted_lua_options_tl
12992   { #1~::~ " }
12993   \tl_gput_right:NV
12994   \g_@@_formatted_lua_options_tl
12995   \l_tmpb_tl

```

```

12996     \tl_gput_right:Nn
12997     \g_@@_formatted_lua_options_tl
12998     { " ,~ }
12999   }
13000 }
13001 \cs_generate_variant:Nn
13002 \clist_map_inline:nn
13003 { Vn }
13004 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
13005 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
13006 \sys_if_engine luatex:TF
13007 {
13008   \cs_new:Nn
13009     \@@_lua_escape:nN
13010     {
13011       \tl_set:Nx
13012         #2
13013         {
13014           \lua_escape:n
13015             { #1 }
13016         }
13017     }
13018 }
13019 {
13020   \regex_const:Nn
13021     \c_@@_lua_escape_regex
13022     { [\\"' ] }
13023   \cs_new:Nn
13024     \@@_lua_escape:nN
13025     {
13026       \tl_set:Nn
13027         #2
13028         { #1 }
13029       \regex_replace_all:NnN
13030         \c_@@_lua_escape_regex
13031         { \u { c_backslash_str } \0 }
13032         #2
13033     }
13034 }
13035 \cs_generate_variant:Nn
13036 \@@_lua_escape:nN
13037 { 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.

```
13038 \tl_new:N
```

```

13039 \markdownInputFilename
13040 \cs_new:Npn
13041 \markdownPrepareInputFilename
13042 #1
13043 {
13044   \@@_lua_escape:xN
13045   { #1 }
13046   \markdownInputFilename
13047   \tl_gset:Nx
13048   \markdownInputFilename
13049   { " \markdownInputFilename " }
13050 }

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T_EX. It exposes the `convert` function for the use by any further Lua code.

```

13051 \cs_new:Npn
13052 \markdownPrepare
13053 {

```

First, ensure that the `cacheDir` directory exists.

```

13054   local~lfs = require("lfs")
13055   local~options = \markdownLuaOptions
13056   if~not~lfs.isdir(options.cacheDir) then~
13057     assert(lfs.mkdir(options.cacheDir))
13058   end~

```

Next, load the `markdown` module and create a converter function using the plain T_EX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

13059   local~md = require("markdown")
13060   local~convert = md.new(options)
13061 }

```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain T_EX. It opens the input file, converts it, and prints the conversion result.

```

13062 \cs_new:Npn
13063 \markdownConvert
13064 {
13065   local~filename = \markdownInputFilename
13066   local~file = assert(io.open(filename, "r"),
13067     [[Could~not~open~file~]] .. filename .. [[~for~reading]])
13068   local~input = assert(file:read("*a"))
13069   assert(file:close())
13070   print(convert(input))
13071 }
13072 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain TeX.

```
13073 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
13074   if options.cacheDir then
13075     lfs.rmdir(options.cacheDir)
13076   end
13077 }%
```

3.2.5 Buffering Block-Level Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```
13078 \csname newread\endcsname\markdownInputFileStream
13079 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
13080 \begingroup
13081   \catcode\^^I=12%
13082   \gdef\markdownReadAndConvertTab{^^I}%
13083 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the L^AT_EX 2_ε `\filecontents` macro to plain TeX.

```
13084 \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.
13085   \catcode\^^M=13%
13086   \catcode\^^I=13%
13087   \catcode|=0%
13088   \catcode\|=12%
13089   |catcode@=14%
13090   |catcode|=12@
13091   |gdef|markdownReadAndConvert#1#2{@
13092     |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
13093     |markdownIfOption{frozenCache}{-}{@
13094       |immediate|openout|markdownOutputFileStream@
13095       |markdownOptionInputTempFileName|relax@
```



```

13096     |markdownInfo{@
13097         Buffering block-level markdown input into the temporary @
13098         input file "|markdownOptionInputTempFileName" and scanning @
13099         for the closing token sequence "#1"}@
13100     }@

```

Locally change the category of the special plain T_EX 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.

```

13101     |def|do##1{|catcode`##1=12}|dospecials@
13102     |catcode`| =12@
13103     |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.

```

13104     |def|markdownReadAndConvertStripPercentSign##1{@
13105         |markdownIfOption{stripPercentSigns}{@
13106             |if##1%@
13107                 |expandafter|expandafter|expandafter@
13108                 |markdownReadAndConvertProcessLine@
13109             |else@
13110                 |expandafter|expandafter|expandafter@
13111                 |markdownReadAndConvertProcessLine@
13112                 |expandafter|expandafter|expandafter##1@
13113             |fi@
13114         }{@
13115             |expandafter@
13116             |markdownReadAndConvertProcessLine@
13117             |expandafter##1@
13118         }@
13119     }@

```

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.

```

13120     |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.

```

13121         |ifx|relax##3|relax@
13122         |markdownIfOption{frozenCache}{-}{@
13123             |immediate|write|markdownOutputFileStream{##1}@
13124         }@
13125         |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T_EX, `\input` the result of the conversion, and expand the ending control sequence.

```

13126     |def^^M{@
13127         |markdownInfo{The ending token sequence was found}@
13128         |markdownIfOption{frozenCache}{-}{@
13129             |immediate|closeout|markdownOutputFileStream@
13130         }@
13131     |endgroup@
13132     |markdownInput{@
13133         |markdownOptionOutputDir@
13134         /|markdownOptionInputTempFileName@
13135     }@
13136     #2}@
13137     |fi@

```

Repeat with the next line.

```

13138     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

13139     |catcode`|^I=13@
13140     |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.

```

13141     |catcode`|^M=13@
13142     |def^^M##1^^M{@
13143         |def^^M###1^^M{@
13144             |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
13145         ^^M}@
13146     ^^M}@

```

Reset the character categories back to the former state.

```

13147 |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.

```

13148 \ExplSyntaxOn
13149 \cs_new:Npn
13150   \markdownLuaExecute
13151   #1
13152   {
13153     \int_compare:nNnT
13154       { \g_luabridge_method_int }
13155       =

```

```

13156     { \c_luabridge_method_shell_int }
13157     {
13158         \sys_if_shell_unrestricted:F
13159         {
13160             \sys_if_shell:TF
13161             {
13162                 \msg_error:nn
13163                 { markdown }
13164                 { restricted-shell-access }
13165             }
13166             {
13167                 \msg_error:nn
13168                 { markdown }
13169                 { disabled-shell-access }
13170             }
13171         }
13172     }
13173     \str_gset:NV
13174     \g_luabridge_output_dirname_str
13175     \markdownOptionOutputDir
13176     \luabridge_now:e
13177     { #1 }
13178 }
13179 \cs_generate_variant:Nn
13180 \msg_new:nnnn
13181 { nnnV }
13182 \tl_set:Nn
13183 \l_tmpa_tl
13184 {
13185     You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
13186     --enable-write18~flag,~or~write~shell_escape=t~in~the~
13187     texmf.cnf~file.
13188 }
13189 \msg_new:nnnV
13190 { markdown }
13191 { restricted-shell-access }
13192 { Shell~escape~is~restricted }
13193 \l_tmpa_tl
13194 \msg_new:nnnV
13195 { markdown }
13196 { disabled-shell-access }
13197 { Shell~escape~is~disabled }
13198 \l_tmpa_tl
13199 \ExplSyntaxOff

```

3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```
13200 \ExplSyntaxOn
13201 \tl_new:N
13202   \g_@@_after_markinline_tl
13203 \tl_gset:Nn
13204   \g_@@_after_markinline_tl
13205   { \unskip }
13206 \cs_new:Npn
13207   \markinline
13208   {
```

Locally change the category of the special plain \TeX characters to *other* in order to prevent unwanted interpretation of the input markdown text as \TeX code.

```
13209   \group_begin:
13210   \cctab_select:N
13211     \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
13212   \@@_if_option:nF
13213     { frozenCache }
13214     {
13215       \immediate
13216       \openout
13217         \markdownOutputFileStream
13218         \markdownOptionInputTempFileName
13219       \relax
13220       \msg_info:nne
13221         { markdown }
13222         { buffering-markinline }
13223         { \markdownOptionInputTempFileName }
13224     }
```

Peek ahead and extract the inline markdown text.

```
13225   \peek_regex_replace_once:nnF
13226     { { (.*) } }
13227     {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
13228     \c { @@_if_option:nF }
13229     \cB { frozenCache \cE }
13230     \cB {
13231       \c { immediate }
13232       \c { write }
13233       \c { markdownOutputFileStream }
```

```

13234         \cB { \1 \cE }
13235         \c { immediate }
13236         \c { closeout }
13237         \c { markdownOutputFileStream }
13238         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

13239         \c { group_end: }
13240         \c { group_begin: }
13241         \c { @@_setup:n }
13242         \cB { contentLevel = inline \cE }
13243         \c { markdownInput }
13244         \cB {
13245             \c { markdownOptionOutputDir } /
13246             \c { markdownOptionInputTempFileName }
13247         \cE }
13248         \c { group_end: }
13249         \c { tl_use:N }
13250         \c { g_@@_after_markinline_tl }
13251     }
13252     {
13253         \msg_error:nn
13254         { markdown }
13255         { markinline-peek-failure }
13256     \group_end:
13257     \tl_use:N
13258     \g_@@_after_markinline_tl
13259     }
13260 }
13261 \msg_new:nnn
13262 { markdown }
13263 { buffering-markinline }
13264 { Buffering~inline~markdown~input~into~
13265   the~temporary~input~file~"#1". }
13266 \msg_new:nnnn
13267 { markdown }
13268 { markinline-peek-failure }
13269 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
13270 { The~macro~should~be~followed~by~inline~
13271   markdown~text~in~curly~braces }
13272 \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_EX.

```

13273 \ExplSyntaxOn
13274 \cs_new:Npn
13275   \markdownInput
13276   #1
13277   {
13278     \@@_if_option:nTF
13279     { frozenCache }
13280     {
13281       \markdownInputRaw
13282       { #1 }
13283     }
13284   }

```

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 \LaTeX , this also includes the directories specified in `\input@path`.

```

13285     \tl_set:Nx
13286     \l_tmpa_tl
13287     { #1 }
13288     \file_get_full_name:VNTF
13289     \l_tmpa_tl
13290     \l_tmpb_tl
13291     {
13292       \exp_args:NV
13293       \markdownInputRaw
13294       \l_tmpb_tl
13295     }
13296     {
13297       \msg_error:nnV
13298       { markdown }
13299       { markdown-file-does-not-exist }
13300       \l_tmpa_tl
13301     }
13302   }
13303 }
13304 \msg_new:nnn
13305 { markdown }
13306 { markdown-file-does-not-exist }
13307 {
13308   Markdown~file~#1~does~not~exist
13309 }
13310 \ExplSyntaxOff
13311 \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.

```

13312 \catcode`|=0%
13313 \catcode`\|=12%
13314 \catcode`|&=6%
13315 |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.

```

13316 |begingroup
13317 |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.

```

13318 |catcode`|#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

13319 |markdownIfOption{frozenCache}{%
13320 |ifnum|markdownOptionFrozenCacheCounter=0|relax
13321 |markdownInfo{Reading frozen cache from
13322 "|markdownOptionFrozenCacheFileName"%
13323 |input|markdownOptionFrozenCacheFileName|relax
13324 |fi
13325 |markdownInfo{Including markdown document number
13326 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
13327 |csname markdownFrozenCache%
13328 |the|markdownOptionFrozenCacheCounter|endcsname
13329 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
13330 }{%
13331 |markdownInfo{Including markdown document "&1"%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L^AT_EX_Mk to track changes to the markdown document.

```

13332 |openin|markdownInputFileStream&1
13333 |closein|markdownInputFileStream
13334 |markdownPrepareLuaOptions
13335 |markdownPrepareInputFilename{&1}%
13336 |markdownLuaExecute{%
13337 |markdownPrepare
13338 |markdownConvert
13339 |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

13340 |markdownIfOption{finalizeCache}{%
13341 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
13342 }%

```

```

13343     |endgroup
13344   }%
13345 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of `TeX` to execute a `TeX` document in the middle of a markdown document fragment.

```

13346 \gdef\markdownEscape#1{%
13347   \catcode`\%=14\relax
13348   \catcode`\#=6\relax
13349   \input #1\relax
13350   \catcode`\%=12\relax
13351   \catcode`\#=12\relax
13352 }%

```

3.3 `LATEX` Implementation

The `LATEX` implementation makes use of the fact that, apart from some subtle differences, `LATEX` implements the majority of the plain `TeX` format [12, Section 9]. As a consequence, we can directly reuse the existing plain `TeX` implementation.

```

13353 \def\markdownVersionSpace{ }%
13354 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
13355   \markdownVersion\markdownVersionSpace markdown renderer]%

```

3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain `TeX` implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the `LATEX` interface (see Section 2.3.2).

```

13356 \ExplSyntaxOn
13357 \cs_gset_eq:NN
13358   \markinlinePlainTeX
13359   \markinline
13360 \cs_gset:Npn
13361   \markinline
13362   {
13363     \peek_regex_replace_once:nn
13364     { ( \[ (.*) \] ) ? }
13365     {

```

Apply the options locally.

```

13366       \c { group_begin: }
13367       \c { @@_setup:n }
13368       \cB { \2 \cE }

```



```

13369     \c { t1_put_right:Nn }
13370     \c { g_@@_after_markinline_t1 }
13371     \cB { \c { group_end: } \cE }
13372     \c { markinlinePlainTeX }
13373   }
13374 }
13375 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain \TeX implementation of the `\markdownInput` macro. The `\markdownInput` macro is then redefined to accept an optional argument with options recognized by the \LaTeX interface (see Section 2.3.2).

```

13376 \let\markdownInputPlainTeX\markdownInput
13377 \renewcommand\markdownInput[2] []{%
13378   \begingroup
13379     \markdownSetup{#1}%
13380     \markdownInputPlainTeX{#2}%
13381   \endgroup}%

```

The `markdown`, and `markdown*` \LaTeX environments are implemented using the `\markdownReadAndConvert` macro.

```

13382 \ExplSyntaxOn
13383 \renewenvironment
13384 { markdown }
13385 {

```

In our implementation of the `markdown` \LaTeX environment, we want to distinguish between the following two cases:

<code>\begin{markdown}</code> [smartEllipses]	<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.

```
13386 \group_begin:
13387 \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).

```
13388 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
13389 \peek_regex_replace_once:nnF
13390 { \ *[\r*([~]]*)\][^~\r]* }
13391 {
```

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.

```
13392 \c { group_end: }
13393 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
13394 \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}
```

```
13395 \c { exp_args:NV }
13396 \c { markdownReadAndConvert@ }
13397 \c { @currenvir }
13398 }
13399 {
13400 \group_end:
13401 \exp_args:NV
13402 \markdownReadAndConvert@
13403 \@currenvir
13404 }
13405 }
```

```

13406 { \markdownEnd }
13407 \renewenvironment
13408 { markdown* }
13409 [ 1 ]
13410 {
13411   \msg_warning:nnn
13412     { markdown }
13413     { latex-markdown-star-deprecated }
13414     { #1 }
13415   \@@_setup:n
13416     { #1 }
13417   \markdownReadAndConvert@
13418     { markdown* }
13419 }
13420 { \markdownEnd }
13421 \msg_new:nnn
13422 { markdown }
13423 { latex-markdown-star-deprecated }
13424 {
13425   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
13426   be~removed~in~the~next~major~version~of~the~Markdown~package.
13427 }
13428 \cs_generate_variant:Nn
13429 \@@_setup:n
13430 { V }
13431 \ExplSyntaxOff
13432 \begingroup

```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```

13433 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
13434 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
13435 |gdef|markdownReadAndConvert@#1<%
13436   |markdownReadAndConvert<\end{#1}>%
13437   <|end<#1>>>%
13438 |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.

```

13439 \ExplSyntaxOn
13440 \cs_gset:Nn

```

```

13441 \@@_load_theme:nn
13442 {

```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

13443 \ifmarkdownLaTeXLoaded
13444 \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load L^AT_EX themes after the preamble. Otherwise, try loading a plain T_EX theme instead.

```

13445 \file_if_exist:nTF
13446 { markdown theme #2.sty }
13447 {
13448 \msg_error:nnn
13449 { markdown }
13450 { latex-theme-after-preamble }
13451 { #1 }
13452 }
13453 {
13454 \@@_plain_tex_load_theme:nn
13455 { #1 }
13456 { #2 }
13457 }
13458 \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L^AT_EX theme if it exists or load a plain T_EX theme otherwise.

```

13459 \file_if_exist:nTF
13460 { markdown theme #2.sty }
13461 {
13462 \msg_info:nnn
13463 { markdown }
13464 { loading-latex-theme }
13465 { #1 }
13466 \RequirePackage
13467 { markdown theme #2 }
13468 }
13469 {
13470 \@@_plain_tex_load_theme:nn
13471 { #1 }
13472 { #2 }
13473 }
13474 \fi
13475 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

13476     \msg_info:nnn
13477         { markdown }
13478         { theme-loading-postponed }
13479         { #1 }
13480     \AtEndOfPackage
13481     {
13482         \@@_set_theme:n
13483         { #1 }
13484     }
13485     \fi
13486 }
13487 \msg_new:nnn
13488 { markdown }
13489 { theme-loading-postponed }
13490 {
13491     Postponing~loading~Markdown~theme~#1~until~
13492     Markdown~package~has~finished~loading
13493 }
13494 \msg_new:nnn
13495 { markdown }
13496 { loading-latex-theme }
13497 { Loading~LaTeX~Markdown~theme~#1 }
13498 \cs_generate_variant:Nn
13499 \msg_new:nnnn
13500 { nnVV }
13501 \tl_set:Nn
13502 \l_tmpa_tl
13503 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
13504 \tl_put_right:NV
13505 \l_tmpa_tl
13506 \c_backslash_str
13507 \tl_put_right:Nn
13508 \l_tmpa_tl
13509 { begin{document} }
13510 \tl_set:Nn
13511 \l_tmpb_tl
13512 { Load~Markdown~theme~#1~before~ }
13513 \tl_put_right:NV
13514 \l_tmpb_tl
13515 \c_backslash_str
13516 \tl_put_right:Nn
13517 \l_tmpb_tl
13518 { begin{document} }
13519 \msg_new:nnVV
13520 { markdown }
13521 { latex-theme-after-preamble }
13522 \l_tmpa_tl

```

```
13523 \l_tmpb_tl
13524 \ExplSyntaxOff
```

The [witiko/dot](#) theme enables the `fencedCode` Lua option:

```
13525 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
13526 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
13527 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
13528 \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T_EX option is disabled and the code block has not been previously typeset:

```
13529 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
13530 \def\next##1 ##2\relax{%
13531 \ifthenelse{\equal{##1}{dot}}{%
13532 \markdownIfOption{frozenCache}{}}{%
13533 \immediate\write18{%
13534 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
13535 then
13536 dot -Tpdf -o #1.pdf #1;
13537 cp #1 #1.pdf.source;
13538 fi}}%
}
```

We include the typeset image using the image token renderer:

```
13539 \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%
```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```
13540 }{%
13541 \markdown@witiko@dot@oldRendererInputFencedCodePrototype
13542 {#1}{#2}{#3}%
13543 }%
13544 }%
13545 \next#2 \relax}%
```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```
13546 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
13547 \markdownRendererImagePrototype
```

We load the `catchfile` and `grffile` packages, see also Section 1.1.3:

```
13548 \RequirePackage{catchfile,grffile}
```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```
13549 \newcount\markdown@witiko@graphicx@http@counter
13550 \markdown@witiko@graphicx@http@counter=0
13551 \newcommand\markdown@witiko@graphicx@http@filename{%
13552   \markdownOptionCacheDir/witiko_graphicx_http%
13553   .\the\markdown@witiko@graphicx@http@counter}%
```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```
13554 \newcommand\markdown@witiko@graphicx@http@download[2]{%
13555   wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}
```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```
13556 \begingroup
13557 \catcode`\%=12
13558 \catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
13559 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
13560   \begingroup
13561   \edef\filename{\markdown@witiko@graphicx@http@filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```
13562   \markdownIfOption{frozenCache}{}{^^A
13563     \immediate\write18{^^A
13564       mkdir -p "\markdownOptionCacheDir";
13565       if printf '%s' "#3" | grep -q -E '^https?:';
13566       then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
13567       OUTPUT_PREFIX="\markdownOptionCacheDir";
13568       OUTPUT_BODY="$ (printf '%s' '#3' | md5sum | cut -d' ' -f1)";
13569       OUTPUT_SUFFIX="$ (printf '%s' '#3' | sed 's/.*[.]/')";
13570       OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
13571       if ! [ -e "$OUTPUT" ];
13572       then
13573         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
```

```

13574         printf '%s' "$OUTPUT" > "\filename";
13575     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:

```

13576     else
13577         printf '%s' '#3' > "\filename";
13578     fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

13579     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
13580     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
13581     {#1}{#2}{\filename}{#4}^^A
13582 \endgroup
13583 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax^^A
13584 \endgroup

```

The `witiko/markdown/defaults` L^AT_EX theme provides default definitions for token renderer prototypes. First, the L^AT_EX theme loads the plain T_EX theme with the default definitions for plain T_EX:

```

13585 \markdownLoadPlainTeXTheme

```

Next, the L^AT_EX theme overrides some of the plain T_EX definitions. See Section 3.3.4 for the actual definitions.

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

13586 \DeclareOption*{%
13587     \expandafter\markdownSetup\expandafter{\CurrentOption}}%
13588 \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.

```

13589 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the `paralist` package.

```

13590 \@ifclassloaded{beamer}{}{%
13591     \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
13592     \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
13593 }

```


If we loaded the paralist package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

13594 \ExplSyntaxOn
13595 \@ifpackageloaded{paralist}{
13596   \tl_new:N
13597     \l_@@_latex_fancy_list_item_label_number_style_tl
13598   \tl_new:N
13599     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13600   \cs_new:Nn
13601     \@@_latex_fancy_list_item_label_number:nn
13602     {
13603       \str_case:nn
13604         { #1 }
13605         {
13606           { Decimal } { #2 }
13607           { LowerRoman } { \int_to_roman:n { #2 } }
13608           { UpperRoman } { \int_to_Roman:n { #2 } }
13609           { LowerAlpha } { \int_to_alph:n { #2 } }
13610           { UpperAlpha } { \int_to_Alph:n { #2 } }
13611         }
13612     }
13613   \cs_new:Nn
13614     \@@_latex_fancy_list_item_label_delimiter:n
13615     {
13616       \str_case:nn
13617         { #1 }
13618         {
13619           { Default } { . }
13620           { OneParen } { ) }
13621           { Period } { . }
13622         }
13623     }
13624   \cs_new:Nn
13625     \@@_latex_fancy_list_item_label:nnn
13626     {
13627       \@@_latex_fancy_list_item_label_number:nn
13628         { #1 }
13629         { #3 }
13630       \@@_latex_fancy_list_item_label_delimiter:n
13631         { #2 }
13632     }
13633   \cs_new:Nn
13634     \@@_latex_paralist_style:nn
13635     {
13636       \str_case:nn
13637         { #1 }

```

```

13638     {
13639     { Decimal } { 1 }
13640     { LowerRoman } { i }
13641     { UpperRoman } { I }
13642     { LowerAlpha } { a }
13643     { UpperAlpha } { A }
13644     }
13645     \@@_latex_fancy_list_item_label_delimiter:n
13646     { #2 }
13647   }
13648   \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

13649     ulBeginTight = {%
13650     \group_begin:
13651     \pltopsep=\topsep
13652     \plpartopsep=\partopsep
13653     \begin{compactitem}
13654   },
13655     ulEndTight = {
13656     \end{compactitem}
13657     \group_end:
13658   },
13659     fancyOlBegin = {
13660     \group_begin:
13661     \tl_set:Nn
13662     \l_@@_latex_fancy_list_item_label_number_style_tl
13663     { #1 }
13664     \tl_set:Nn
13665     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13666     { #2 }
13667     \@@_if_option:nTF
13668     { startNumber }
13669     {
13670     \tl_set:Nn
13671     \l_tmpa_tl
13672     { \begin{enumerate} }
13673   }
13674     {
13675     \tl_set:Nn
13676     \l_tmpa_tl
13677     { \begin{enumerate}[ ] }
13678     \tl_put_right:Nx
13679     \l_tmpa_tl
13680     { \@@_latex_paralist_style:nn { #1 } { #2 } }
13681     \tl_put_right:Nn

```

```

13682         \l_tmpa_tl
13683         { ] }
13684     }
13685     \tl_use:N
13686         \l_tmpa_tl
13687 },
13688 fancyOlEnd = {
13689     \end{enumerate}
13690     \group_end:
13691 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

13692     olBeginTight = {%
13693         \group_begin:
13694         \plpartopsep=\partopsep
13695         \pltopsep=\topsep
13696         \begin{compactenum}
13697     },
13698     olEndTight = {
13699         \end{compactenum}
13700         \group_end:
13701     },
13702     fancyOlBeginTight = {
13703         \group_begin:
13704         \tl_set:Nn
13705             \l_@@_latex_fancy_list_item_label_number_style_tl
13706             { #1 }
13707         \tl_set:Nn
13708             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13709             { #2 }
13710         \@@_if_option:nTF
13711             { startNumber }
13712             {
13713                 \tl_set:Nn
13714                     \l_tmpa_tl
13715                     { \begin{compactenum} }
13716             }
13717             {
13718                 \tl_set:Nn
13719                     \l_tmpa_tl
13720                     { \begin{compactenum}[ ]
13721                 \tl_put_right:Nx
13722                     \l_tmpa_tl
13723                     { \@@_latex_paralist_style:nn { #1 } { #2 } }
13724                 \tl_put_right:Nn
13725                     \l_tmpa_tl

```

```

13726         { ] }
13727     }
13728     \tl_put_left:Nn
13729     \l_tmpa_tl
13730     {
13731         \plpartopsep=\partopsep
13732         \pltopsep=\topsep
13733     }
13734     \tl_use:N
13735     \l_tmpa_tl
13736 },
13737 fancyO1EndTight = {
13738     \end{compactenum}
13739     \group_end:
13740 },
13741 fancyO1ItemWithNumber = {
13742     \item
13743     [
13744         \@@_latex_fancy_list_item_label:VVn
13745         \l_@@_latex_fancy_list_item_label_number_style_tl
13746         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
13747         { #1 }
13748     ]
13749 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

13750     dlBeginTight = {
13751         \group_begin:
13752         \plpartopsep=\partopsep
13753         \pltopsep=\topsep
13754         \begin{compactdesc}
13755     },
13756     dlEndTight = {
13757         \end{compactdesc}
13758         \group_end:
13759     }}}
13760 \cs_generate_variant:Nn
13761 \@@_latex_fancy_list_item_label:nnn
13762 { VVn }
13763 }{
13764 \markdownSetup{rendererPrototypes={
13765     ulBeginTight = {\markdownRendererUlBegin},
13766     ulEndTight = {\markdownRendererUlEnd},
13767     fancyO1Begin = {\markdownRendererO1Begin},
13768     fancyO1End = {\markdownRendererO1End},
13769     olBeginTight = {\markdownRendererO1Begin},

```

```

13770     olEndTight = {\markdownRendererOlEnd},
13771     fancyOlBeginTight = {\markdownRendererOlBegin},
13772     fancyOlEndTight = {\markdownRendererOlEnd},
13773     dlBeginTight = {\markdownRendererDlBegin},
13774     dlEndTight = {\markdownRendererDlEnd}}}
13775 }
13776 \ExplSyntaxOff
13777 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

13778 \@ifpackageloaded{unicode-math}{
13779   \markdownSetup{rendererPrototypes={
13780     untickedBox = {\$mdlgwhtsquare$},
13781   }}
13782 }{
13783   \RequirePackage{amssymb}
13784   \markdownSetup{rendererPrototypes={
13785     untickedBox = {\$square$},
13786   }}
13787 }
13788 \RequirePackage{csvsimple}
13789 \RequirePackage{fancyvrb}
13790 \RequirePackage{graphicx}
13791 \markdownSetup{rendererPrototypes={
13792   hardLineBreak = {\},
13793   leftBrace = {\textbraceleft},
13794   rightBrace = {\textbraceright},
13795   dollarSign = {\textdollar},
13796   underscore = {\textunderscore},
13797   circumflex = {\textasciicircum},
13798   backslash = {\textbackslash},
13799   tilde = {\textasciitilde},
13800   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by \TeX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,³⁴ 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.

```

13801   codeSpan = {%
13802     \ifmmode
13803       \text{#1}%

```

³⁴This 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.

```

13804     \else
13805         \texttt{#1}%
13806     \fi
13807 }}}}
13808 \ExplSyntaxOn
13809 \markdownSetup{
13810     rendererPrototypes = {
13811         contentBlock = {
13812             \str_case:nnF
13813             { #1 }
13814             {
13815                 { csv }
13816                 {
13817                     \begin{table}
13818                     \begin{center}
13819                         \csvautotabular{#3}
13820                     \end{center}
13821                     \tl_if_empty:nF
13822                     { #4 }
13823                     { \caption{#4} }
13824                     \end{table}
13825                 }
13826                 { tex } { \markdownEscape{#3} }
13827             }
13828             { \markdownInput{#3} }
13829         },
13830     },
13831 }
13832 \ExplSyntaxOff
13833 \markdownSetup{rendererPrototypes={
13834     image = {%
13835         \begin{figure}%
13836         \begin{center}%
13837             \includegraphics[alt={#1}]{#3}%
13838         \end{center}%
13839         \ifx\empty#4\empty\else
13840             \caption{#4}%
13841         \fi
13842     \end{figure}},
13843     ulBegin = {\begin{itemize}},
13844     ulEnd = {\end{itemize}},
13845     olBegin = {\begin{enumerate}},
13846     olItem = {\item{}},
13847     olItemWithNumber = {\item[#1.]},
13848     olEnd = {\end{enumerate}},
13849     dlBegin = {\begin{description}},
13850     dlItem = {\item[#1]},

```

```

13851 dlEnd = {\end{description}},
13852 emphasis = {\emph{#1}},
13853 tickedBox = {\boxtimes$},
13854 halfTickedBox = {\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

13855 \ExplSyntaxOn
13856 \seq_new:N
13857 \l_@@_header_identifiers_seq
13858 \markdownSetup
13859 {
13860   rendererPrototypes = {
13861     headerAttributeContextBegin = {
13862       \markdownSetup
13863       {
13864         rendererPrototypes = {
13865           attributeIdentifier = {
13866             \seq_put_right:Nn
13867             \l_@@_header_identifiers_seq
13868             { ##1 }
13869           },
13870         },
13871       }
13872     },
13873     headerAttributeContextEnd = {
13874       \seq_map_inline:Nn
13875       \l_@@_header_identifiers_seq
13876       { \label { ##1 } }
13877       \seq_clear:N
13878       \l_@@_header_identifiers_seq
13879     },
13880   },
13881 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

13882 \bool_new:N
13883 \l_@@_header_unnumbered_bool
13884 \markdownSetup
13885 {
13886   rendererPrototypes = {
13887     headerAttributeContextBegin += {
13888       \markdownSetup
13889       {
13890         rendererPrototypes = {
13891           attributeClassName = {
13892             \bool_if:nT
13893             {

```

```

13894         \str_if_eq_p:nn
13895         { ##1 }
13896         { unnumbered } &&
13897         ! \l_@@_header_unnumbered_bool
13898     }
13899     {
13900     \group_begin:
13901     \bool_set_true:N
13902     \l_@@_header_unnumbered_bool
13903     \c@secnumdepth = 0
13904     \markdownSetup
13905     {
13906         rendererPrototypes = {
13907             sectionBegin = {
13908                 \group_begin:
13909                 },
13910             sectionEnd = {
13911                 \group_end:
13912                 },
13913             },
13914         }
13915     },
13916 },
13917 },
13918 }
13919 },
13920 },
13921 }
13922 \ExplSyntaxOff
13923 \markdownSetup{rendererPrototypes={
13924     superscript = {\textsuperscript{#1}},
13925     subscript = {\textsubscript{#1}},
13926     blockQuoteBegin = {\begin{quotation}},
13927     blockQuoteEnd = {\end{quotation}},
13928     inputVerbatim = {\VerbatimInput{#1}},
13929     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
13930     note = {\footnote{#1}}}}

```

3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13931 \RequirePackage{ltxcmds}
13932 \ExplSyntaxOn
13933 \cs_gset:Npn
13934 \markdownRendererInputFencedCodePrototype#1#2#3
13935 {
13936     \tl_if_empty:nTF

```



```

13937     { #2 }
13938     { \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.

```

13939     {
13940         \regex_extract_once:nnN
13941         { \w* }
13942         { #2 }
13943         \l_tmpa_seq
13944         \seq_pop_left:NN
13945         \l_tmpa_seq
13946         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

13947     \ltx@ifpackageloaded
13948     { minted }
13949     {
13950         \catcode`\#=6\relax
13951         \exp_args:NV
13952         \inputminted
13953         \l_tmpa_tl
13954         { #1 }
13955         \catcode`\#=12\relax
13956     }
13957     {
13958         \ltx@ifpackageloaded
13959         { minted2 }
13960         {
13961             \catcode`\#=6\relax
13962             \exp_args:NV
13963             \inputminted
13964             \l_tmpa_tl
13965             { #1 }
13966             \catcode`\#=12\relax
13967         }
13968     }

```

When the listings package is loaded, use it for syntax highlighting.

```

13969         \ltx@ifpackageloaded
13970         { listings }
13971         { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

13972         { \markdownRendererInputFencedCode{#1}{-}{-} }
13973     }
13974 }

```

```

13975     }
13976   }
13977 \ExplSyntaxOff

Support the nesting of strong emphasis.

13978 \ExplSyntaxOn
13979 \def\markdownLATEXStrongEmphasis#1{%
13980   \str_if_in:NnTF
13981     \f@series
13982     { b }
13983     { \textnormal{#1} }
13984     { \textbf{#1} }
13985 }
13986 \ExplSyntaxOff
13987 \markdownSetup{rendererPrototypes={strongEmphasis={%
13988   \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L^AT_EX document classes that do not provide chapters.

```

13989 \@ifundefined{chapter}{%
13990   \markdownSetup{rendererPrototypes = {
13991     headingOne = {\section{#1}},
13992     headingTwo = {\subsection{#1}},
13993     headingThree = {\subsubsection{#1}},
13994     headingFour = {\paragraph{#1}},
13995     headingFive = {\subparagraph{#1}}}}
13996 }{%
13997   \markdownSetup{rendererPrototypes = {
13998     headingOne = {\chapter{#1}},
13999     headingTwo = {\section{#1}},
14000     headingThree = {\subsection{#1}},
14001     headingFour = {\subsubsection{#1}},
14002     headingFive = {\paragraph{#1}},
14003     headingSix = {\subparagraph{#1}}}}
14004 }%

```

3.3.4.2 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

14005 \markdownSetup{
14006   rendererPrototypes = {
14007     ulItem = {%
14008       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
14009     },
14010   },
14011 }
14012 \def\markdownLaTeXUListItem{%
14013   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox

```

```

14014     \item[\markdownLaTeXCheckbox]%
14015     \expandafter\@gobble
14016   \else
14017     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
14018       \item[\markdownLaTeXCheckbox]%
14019       \expandafter\expandafter\expandafter\@gobble
14020     \else
14021       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
14022         \item[\markdownLaTeXCheckbox]%
14023         \expandafter\expandafter\expandafter\expandafter
14024         \expandafter\expandafter\expandafter\@gobble
14025       \else
14026         \item{}%
14027       \fi
14028     \fi
14029   \fi
14030 }

```

3.3.4.3 HTML elements

If the `html` option is enabled and we are using `TeX4ht35`, we will pass HTML elements to the output HTML document unchanged.

```

14031 \@ifundefined{HCode}{}{
14032   \markdownSetup{
14033     rendererPrototypes = {
14034       inlineHtmlTag = {%
14035         \ifvmode
14036           \IgnorePar
14037           \EndP
14038         \fi
14039         \HCode{#1}%
14040       },
14041       inputBlockHtmlElement = {%
14042         \ifvmode
14043           \IgnorePar
14044           \fi
14045           \EndP
14046           \special{t4ht*#1}%
14047           \par
14048           \ShowPar
14049       },
14050     },
14051   }
14052 }

```

³⁵See <https://tug.org/tex4ht/>.

3.3.4.4 Citations

Here is a basic implementation for citations that uses the L^AT_EX `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL^AT_EX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
14053 \newcount\markdownLaTeXCitationsCounter
14054
14055 % Basic implementation
14056 \RequirePackage{gobble}
14057 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
14058   \advance\markdownLaTeXCitationsCounter by 1\relax
14059   \ifx\relax#4\relax
14060     \ifx\relax#5\relax
14061       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14062         \relax
14063         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
14064         \expandafter\expandafter\expandafter
14065         \expandafter\expandafter\expandafter\expandafter
14066         \@gobblethree
14067       \fi
14068     \else% Before a postnote (#5), dump the accumulator
14069       \ifx\relax#1\relax\else
14070         \cite{#1}%
14071       \fi
14072       \cite[#5]{#6}%
14073       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14074         \relax
14075       \else
14076         \expandafter\expandafter\expandafter
14077         \expandafter\expandafter\expandafter\expandafter
14078         \expandafter\expandafter\expandafter
14079         \expandafter\expandafter\expandafter\expandafter
14080         \markdownLaTeXBasicCitations
14081       \fi
14082       \expandafter\expandafter\expandafter
14083       \expandafter\expandafter\expandafter\expandafter{%
14084       \expandafter\expandafter\expandafter
14085       \expandafter\expandafter\expandafter\expandafter}%
14086       \expandafter\expandafter\expandafter
14087       \expandafter\expandafter\expandafter\expandafter{%
14088       \expandafter\expandafter\expandafter
14089       \expandafter\expandafter\expandafter\expandafter}%
14090       \expandafter\expandafter\expandafter
14091       \@gobblethree
14092     \fi
14093   \else% Before a prenote (#4), dump the accumulator
```

```

14094 \ifx\relax#1\relax\else
14095 \cite{#1}%
14096 \fi
14097 \ifnum\markdownLaTeXCitationsCounter>1\relax
14098 \space % Insert a space before the prenote in later citations
14099 \fi
14100 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
14101 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14102 \relax
14103 \else
14104 \expandafter\expandafter\expandafter
14105 \expandafter\expandafter\expandafter\expandafter
14106 \markdownLaTeXBasicCitations
14107 \fi
14108 \expandafter\expandafter\expandafter{%
14109 \expandafter\expandafter\expandafter}%
14110 \expandafter\expandafter\expandafter{%
14111 \expandafter\expandafter\expandafter}%
14112 \expandafter
14113 \@gobblethree
14114 \fi\markdownLaTeXBasicCitations{#1#2#6},}
14115 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
14116
14117 % Natbib implementation
14118 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
14119 \advance\markdownLaTeXCitationsCounter by 1\relax
14120 \ifx\relax#3\relax
14121 \ifx\relax#4\relax
14122 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14123 \relax
14124 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
14125 \expandafter\expandafter\expandafter
14126 \expandafter\expandafter\expandafter\expandafter
14127 \@gobbletwo
14128 \fi
14129 \else% Before a postnote (#4), dump the accumulator
14130 \ifx\relax#1\relax\else
14131 \citep{#1}%
14132 \fi
14133 \citep[] [#4]{#5}%
14134 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14135 \relax
14136 \else
14137 \expandafter\expandafter\expandafter
14138 \expandafter\expandafter\expandafter\expandafter
14139 \expandafter\expandafter\expandafter
14140 \expandafter\expandafter\expandafter\expandafter

```

```

14141     \markdownLaTeXNatbibCitations
14142     \fi
14143     \expandafter\expandafter\expandafter
14144     \expandafter\expandafter\expandafter\expandafter{%
14145     \expandafter\expandafter\expandafter
14146     \expandafter\expandafter\expandafter\expandafter}%
14147     \expandafter\expandafter\expandafter
14148     \@gobbletwo
14149     \fi
14150 \else% Before a prenote (#3), dump the accumulator
14151     \ifx\relax#1\relax\relax\else
14152         \citep{#1}%
14153     \fi
14154     \citep[#3][#4]{#5}%
14155     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14156     \relax
14157     \else
14158         \expandafter\expandafter\expandafter
14159         \expandafter\expandafter\expandafter\expandafter
14160         \markdownLaTeXNatbibCitations
14161     \fi
14162     \expandafter\expandafter\expandafter{%
14163     \expandafter\expandafter\expandafter}%
14164     \expandafter
14165     \@gobbletwo
14166     \fi\markdownLaTeXNatbibCitations{#1,#5}}
14167 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
14168     \advance\markdownLaTeXCitationsCounter by 1\relax
14169     \ifx\relax#3\relax
14170         \ifx\relax#4\relax
14171             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14172             \relax
14173             \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
14174             \expandafter\expandafter\expandafter
14175             \expandafter\expandafter\expandafter\expandafter
14176             \@gobbletwo
14177         \fi
14178     \else% After a prenote or a postnote, dump the accumulator
14179         \ifx\relax#1\relax\else
14180             \citet{#1}%
14181         \fi
14182         , \citet[#3][#4]{#5}%
14183         \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
14184         \relax
14185         ,
14186     \else
14187         \ifnum

```

```

14188     \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
14189     \relax
14190     ,
14191     \fi
14192     \fi
14193     \expandafter\expandafter\expandafter
14194     \expandafter\expandafter\expandafter\expandafter
14195     \markdownLaTeXNatbibTextCitations
14196     \expandafter\expandafter\expandafter
14197     \expandafter\expandafter\expandafter\expandafter{%
14198     \expandafter\expandafter\expandafter
14199     \expandafter\expandafter\expandafter\expandafter}%
14200     \expandafter\expandafter\expandafter
14201     \@gobbletwo
14202     \fi
14203 \else% After a prenote or a postnote, dump the accumulator
14204     \ifx\relax#1\relax\relax\else
14205         \citet{#1}%
14206     \fi
14207     , \citet[#3][#4]{#5}%
14208     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
14209     \relax
14210     ,
14211     \else
14212         \ifnum
14213             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
14214             \relax
14215             ,
14216             \fi
14217             \fi
14218             \expandafter\expandafter\expandafter
14219             \markdownLaTeXNatbibTextCitations
14220             \expandafter\expandafter\expandafter{%
14221             \expandafter\expandafter\expandafter}%
14222             \expandafter
14223             \@gobbletwo
14224             \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
14225
14226 % BibLaTeX implementation
14227 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
14228     \advance\markdownLaTeXCitationsCounter by 1\relax
14229     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14230     \relax
14231     \autocites#1[#3][#4]{#5}%
14232     \expandafter\@gobbletwo
14233     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
14234 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%

```

```

14235 \advance\markdownLaTeXCitationsCounter by 1\relax
14236 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14237 \relax
14238 \textcites#1[#3][#4]{#5}%
14239 \expandafter\@gobbletwo
14240 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
14241
14242 \markdownSetup{rendererPrototypes = {
14243 cite = {%
14244 \markdownLaTeXCitationsCounter=1%
14245 \def\markdownLaTeXCitationsTotal{#1}%
14246 \@ifundefined{autocites}{%
14247 \@ifundefined{citep}{%
14248 \expandafter\expandafter\expandafter
14249 \markdownLaTeXBasicCitations
14250 \expandafter\expandafter\expandafter{%
14251 \expandafter\expandafter\expandafter}%
14252 \expandafter\expandafter\expandafter{%
14253 \expandafter\expandafter\expandafter}%
14254 }{%
14255 \expandafter\expandafter\expandafter
14256 \markdownLaTeXNatbibCitations
14257 \expandafter\expandafter\expandafter{%
14258 \expandafter\expandafter\expandafter}%
14259 }%
14260 }{%
14261 \expandafter\expandafter\expandafter
14262 \markdownLaTeXBibLaTeXCitations
14263 \expandafter{\expandafter}%
14264 }},
14265 textCite = {%
14266 \markdownLaTeXCitationsCounter=1%
14267 \def\markdownLaTeXCitationsTotal{#1}%
14268 \@ifundefined{autocites}{%
14269 \@ifundefined{citep}{%
14270 \expandafter\expandafter\expandafter
14271 \markdownLaTeXBasicTextCitations
14272 \expandafter\expandafter\expandafter{%
14273 \expandafter\expandafter\expandafter}%
14274 \expandafter\expandafter\expandafter{%
14275 \expandafter\expandafter\expandafter}%
14276 }{%
14277 \expandafter\expandafter\expandafter
14278 \markdownLaTeXNatbibTextCitations
14279 \expandafter\expandafter\expandafter{%
14280 \expandafter\expandafter\expandafter}%
14281 }%

```



```

14282     }{%
14283     \expandafter\expandafter\expandafter
14284     \markdownLaTeXBibLaTeXTextCitations
14285     \expandafter{\expandafter}%
14286     }}}

```

3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```

14287 \RequirePackage{url}
14288 \RequirePackage{expl3}
14289 \ExplSyntaxOn
14290 \def\markdownRendererLinkPrototype#1#2#3#4{
14291   \tl_set:Nn \l_tmpa_tl { #1 }
14292   \tl_set:Nn \l_tmpb_tl { #2 }
14293   \bool_set:Nn
14294     \l_tmpa_bool
14295     {
14296       \tl_if_eq_p:NN
14297         \l_tmpa_tl
14298         \l_tmpb_tl
14299     }
14300   \tl_set:Nn \l_tmpa_tl { #4 }
14301   \bool_set:Nn
14302     \l_tmpb_bool
14303     {
14304       \tl_if_empty_p:N
14305         \l_tmpa_tl
14306     }

```

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.

```

14307   \bool_if:nTF
14308     {
14309       \l_tmpa_bool && \l_tmpb_bool
14310     }
14311     {
14312       \markdownLaTeXRendererAutolink { #2 } { #3 }
14313     }{
14314       \markdownLaTeXRendererDirectOrIndirectLink
14315         { #1 } { #2 } { #3 } { #4 }
14316     }
14317 }
14318 \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.

```

14319 \tl_set:Nn
14320   \l_tmpa_tl
14321   { #2 }
14322 \tl_trim_spaces:N
14323   \l_tmpa_tl
14324 \tl_set:Nx
14325   \l_tmpb_tl
14326   {
14327     \tl_range:Nnn
14328     \l_tmpa_tl
14329     { 1 }
14330     { 1 }
14331   }
14332 \str_if_eq:NNTF
14333   \l_tmpb_tl
14334   \c_hash_str
14335   {
14336     \tl_set:Nx
14337     \l_tmpb_tl
14338     {
14339       \tl_range:Nnn
14340       \l_tmpa_tl
14341       { 2 }
14342       { -1 }
14343     }
14344     \exp_args:NV
14345     \ref
14346     \l_tmpb_tl
14347   }{
14348     \url { #2 }
14349   }
14350 }
14351 \ExplSyntaxOff
14352 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
14353   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}

```

3.3.4.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

14354 \newcount\markdownLaTeXRowCount
14355 \newcount\markdownLaTeXRowTotal
14356 \newcount\markdownLaTeXColumnCounter
14357 \newcount\markdownLaTeXColumnTotal
14358 \newtoks\markdownLaTeXTable
14359 \newtoks\markdownLaTeXTableAlignment
14360 \newtoks\markdownLaTeXTableEnd

```

```

14361 \AtBeginDocument{%
14362   \@ifpackageloaded{booktabs}{%
14363     \def\markdownLaTeXTopRule{\toprule}%
14364     \def\markdownLaTeXMidRule{\midrule}%
14365     \def\markdownLaTeXBottomRule{\bottomrule}%
14366   }{%
14367     \def\markdownLaTeXTopRule{\hline}%
14368     \def\markdownLaTeXMidRule{\hline}%
14369     \def\markdownLaTeXBottomRule{\hline}%
14370   }%
14371 }
14372 \markdownSetup{rendererPrototypes={
14373   table = {%
14374     \markdownLaTeXTable={}%
14375     \markdownLaTeXTableAlignment={}%
14376     \markdownLaTeXTableEnd={%
14377       \markdownLaTeXBottomRule
14378       \end{tabular}}%
14379     \ifx\empty#1\empty\else
14380       \addto@hook\markdownLaTeXTable{%
14381         \begin{table}
14382         \centering}%
14383       \addto@hook\markdownLaTeXTableEnd{%
14384         \caption{#1}
14385         \end{table}}%
14386     \fi
14387     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
14388     \markdownLaTeXRowCount=0%
14389     \markdownLaTeXRowTotal=#2%
14390     \markdownLaTeXColumnTotal=#3%
14391     \markdownLaTeXRenderTableRow
14392   }
14393 }}
14394 \def\markdownLaTeXRenderTableRow#1{%
14395   \markdownLaTeXColumnCounter=0%
14396   \ifnum\markdownLaTeXRowCount=0\relax
14397     \markdownLaTeXReadAlignments#1%
14398     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
14399       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
14400         \the\markdownLaTeXTableAlignment}}%
14401     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
14402   \else
14403     \markdownLaTeXRenderTableCell#1%
14404   \fi
14405   \ifnum\markdownLaTeXRowCount=1\relax
14406     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
14407   \fi

```

```

14408 \advance\markdownLaTeXRowCount by 1\relax
14409 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
14410 \the\markdownLaTeXTable
14411 \the\markdownLaTeXTableEnd
14412 \expandafter\@gobble
14413 \fi\markdownLaTeXRenderTableRow}
14414 \def\markdownLaTeXReadAlignments#1{%
14415 \advance\markdownLaTeXColumnCounter by 1\relax
14416 \if#1d%
14417 \addto@hook\markdownLaTeXTableAlignment{1}%
14418 \else
14419 \addto@hook\markdownLaTeXTableAlignment{#1}%
14420 \fi
14421 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
14422 \expandafter\@gobble
14423 \fi\markdownLaTeXReadAlignments}
14424 \def\markdownLaTeXRenderTableCell#1{%
14425 \advance\markdownLaTeXColumnCounter by 1\relax
14426 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
14427 \addto@hook\markdownLaTeXTable{#1&}%
14428 \else
14429 \addto@hook\markdownLaTeXTable{#1\\}%
14430 \expandafter\@gobble
14431 \fi\markdownLaTeXRenderTableCell}

```

3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

14432
14433 \markdownIfOption{lineBlocks}{%
14434 \RequirePackage{verse}
14435 \markdownSetup{rendererPrototypes={
14436 lineBlockBegin = {%
14437 \begingroup
14438 \def\markdownRendererHardLineBreak{\\}%
14439 \begin{verse}%
14440 },
14441 lineBlockEnd = {%
14442 \end{verse}%
14443 \endgroup
14444 },
14445 }}
14446 }{}
14447

```

3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

14448 \ExplSyntaxOn
14449 \keys_define:nn
14450   { markdown/jekyllData }
14451   {
14452     author .code:n = { \author{#1} },
14453     date   .code:n = { \date{#1}   },
14454     title  .code:n = { \title{#1}  },
14455   }

```

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.

```

14456 \markdownSetup{
14457   rendererPrototypes = {
14458     jekyllDataEnd = {
14459       \AddToHook{begindocument/end}{\maketitle}
14460     },
14461   },
14462 }
14463 \ExplSyntaxOff

```

3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

14464 \markdownIfOption{strikeThrough}{%
14465   \RequirePackage{soulutf8}%
14466   \markdownSetup{
14467     rendererPrototypes = {
14468       strikeThrough = {%
14469         \st{#1}%
14470       },
14471     }
14472   }
14473 }{}

```

3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```

14474 \markdownIfOption{mark}{%
14475   \RequirePackage{soulutf8}%
14476   \markdownSetup{

```

```

14477     rendererPrototypes = {
14478         mark = {%
14479             \h1{#1}%
14480         },
14481     }
14482 }
14483 }{}

```

3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form `<key>=<value>` set the corresponding keys of the `graphicx` package to the corresponding values.

```

14484 \ExplSyntaxOn
14485 \@@_if_option:nT
14486 { linkAttributes }
14487 {
14488     \RequirePackage{graphicx}
14489     \markdownSetup{
14490         rendererPrototypes = {
14491             imageAttributeContextBegin = {
14492                 \group_begin:
14493                 \markdownSetup{
14494                     rendererPrototypes = {
14495                         attributeKeyValue = {
14496                             \setkeys
14497                             { Gin }
14498                             { { ##1 } = { ##2 } }
14499                         },
14500                     },
14501                 }
14502             },
14503             imageAttributeContextEnd = {
14504                 \group_end:
14505             },
14506         },
14507     }
14508 }
14509 \ExplSyntaxOff

```

3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

14510 \ExplSyntaxOn
14511 \cs_gset:Npn

```

```

14512 \markdownRendererInputRawInlinePrototype#1#2
14513 {
14514   \str_case:nnF
14515     { #2 }
14516     {
14517       { latex }
14518       {
14519         \@_plain_tex_default_input_raw_inline:nn
14520         { #1 }
14521         { tex }
14522       }
14523     }
14524     {
14525       \@_plain_tex_default_input_raw_inline:nn
14526       { #1 }
14527       { #2 }
14528     }
14529   }
14530 \cs_gset:Npn
14531 \markdownRendererInputRawBlockPrototype#1#2
14532 {
14533   \str_case:nnF
14534     { #2 }
14535     {
14536       { latex }
14537       {
14538         \@_plain_tex_default_input_raw_block:nn
14539         { #1 }
14540         { tex }
14541       }
14542     }
14543     {
14544       \@_plain_tex_default_input_raw_block:nn
14545       { #1 }
14546       { #2 }
14547     }
14548   }
14549 \ExplSyntaxOff
14550 \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.

```

14551 \newcommand\markdownMakeOther{%
14552   \count0=128\relax
14553   \loop
14554     \catcode\count0=11\relax
14555     \advance\count0 by 1\relax
14556   \ifnum\count0<256\repeat}%

```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain T_EX implementation. As a consequence, we can directly reuse the existing plain T_EX implementation after supplying the missing plain T_EX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L^AT_EX package.

```

14557 \def\markdownMakeOther{%
14558   \count0=128\relax
14559   \loop
14560     \catcode\count0=11\relax
14561     \advance\count0 by 1\relax
14562   \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_EXt.

```

14563   \catcode`|=12}%

```

3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is defined to accept an optional argument with options recognized by the ConT_EXt interface (see Section 2.4.2).

```

14564 \long\def\inputmarkdown{%
14565   \dosingleempty
14566   \doinputmarkdown}%
14567 \long\def\doinputmarkdown[#1]#2{%
14568   \begingroup
14569     \iffirstargument
14570       \setupmarkdown[#1]%
14571     \fi
14572   \markdownInput{#2}%
14573   \endgroup}%

```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's T_EX, trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)T_EX, but ConT_EXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConT_EXt MkIV and therefore to insert hard line breaks into markdown text.

```

14574 \startluacode
14575   document.markdown_buffering = false
14576   local function preserve_trailing_spaces(line)
14577     if document.markdown_buffering then
14578       line = line:gsub("[ \\t][ \\t]$", "\\t\\t")
14579     end
14580     return line
14581   end
14582   resolvers.installinputlinehandler(preserve_trailing_spaces)
14583 \stopluacode
14584 \beginingroup
14585   \catcode`|=0%
14586   \catcode`\\=12%
14587   |gdef|startmarkdown{%
14588     |ctxlua{document.markdown_buffering = true}%
14589     |markdownReadAndConvert{\\stopmarkdown}%
14590                                   {|stopmarkdown}}%
14591   |gdef|stopmarkdown{%
14592     |ctxlua{document.markdown_buffering = false}%
14593     |markdownEnd}%
14594 |endgroup

```

3.4.2 Themes

This section overrides the plain T_EX implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConT_EXt themes provided with the Markdown package.

```

14595 \ExplSyntaxOn
14596 \cs_gset:Nn
14597   \@@_load_theme:nn
14598   {

```

Determine whether a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain T_EX theme instead.

```

14599   \file_if_exist:nTF
14600     { t - markdown theme #2.tex }
14601     {
14602       \msg_info:nnn
14603         { markdown }
14604         { loading-context-theme }

```

```

14605         { #1 }
14606     \usemodule
14607         [ t ]
14608         [ markdown theme #2 ]
14609     }
14610     {
14611         \@_plain_tex_load_theme:nn
14612         { #1 }
14613         { #2 }
14614     }
14615 }
14616 \msg_new:nnn
14617 { markdown }
14618 { loading-context-theme }
14619 { Loading~ConTeXt~Markdown~theme~#1 }
14620 \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:

```
14621 \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.

```

14622 \markdownIfOption{plain}{\iffalse}{\iftrue}
14623 \def\markdownRendererHardLineBreakPrototype{\blank}%
14624 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
14625 \def\markdownRendererRightBracePrototype{\textbraceright}%
14626 \def\markdownRendererDollarSignPrototype{\textdollar}%
14627 \def\markdownRendererPercentSignPrototype{\percent}%
14628 \def\markdownRendererUnderscorePrototype{\textunderscore}%
14629 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
14630 \def\markdownRendererBackslashPrototype{\textbackslash}%
14631 \def\markdownRendererTildePrototype{\textasciitilde}%
14632 \def\markdownRendererPipePrototype{\char`|}%
14633 \def\markdownRendererLinkPrototype#1#2#3#4{%
14634     \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
14635     \fi\tt<\hyphenatedurl{#3}>}}%
14636 \usemodule[database]
14637 \defineseparatedlist
14638 [MarkdownConTeXtCSV]
14639 [separator={,},

```

```

14640     before=\bTABLE,after=\eTABLE,
14641     first=\bTR,last=\eTR,
14642     left=\bTD,right=\eTD]
14643 \def\markdownConTeXtCSV{csv}
14644 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
14645     \def\markdownConTeXtCSV@arg{#1}%
14646     \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
14647         \placetable [] [tab:#1]{#4}{%
14648             \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
14649     \else
14650         \markdownInput{#3}%
14651     \fi}%
14652 \def\markdownRendererImagePrototype#1#2#3#4{%
14653     \placefigure [] []{#4}{\externalfigure[#3]}%
14654 \def\markdownRendererUlBeginPrototype{\startitemize}%
14655 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
14656 \def\markdownRendererUlItemPrototype{\item}%
14657 \def\markdownRendererUlEndPrototype{\stopitemize}%
14658 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
14659 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
14660 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
14661 \def\markdownRendererOlItemPrototype{\item}%
14662 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
14663 \def\markdownRendererOlEndPrototype{\stopitemize}%
14664 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
14665 \definedescription
14666     [MarkdownConTeXtDlItemPrototype]
14667     [location=hanging,
14668     margin=standard,
14669     headstyle=bold]%
14670 \definestartstop
14671     [MarkdownConTeXtDlPrototype]
14672     [before=\blank,
14673     after=\blank]%
14674 \definestartstop
14675     [MarkdownConTeXtDlTightPrototype]
14676     [before=\blank\startpacked,
14677     after=\stoppacked\blank]%
14678 \def\markdownRendererDlBeginPrototype{%
14679     \startMarkdownConTeXtDlPrototype}%
14680 \def\markdownRendererDlBeginTightPrototype{%
14681     \startMarkdownConTeXtDlTightPrototype}%
14682 \def\markdownRendererDlItemPrototype#1{%
14683     \startMarkdownConTeXtDlItemPrototype{#1}}%
14684 \def\markdownRendererDlItemEndPrototype{%
14685     \stopMarkdownConTeXtDlItemPrototype}%
14686 \def\markdownRendererDlEndPrototype{%

```

```

14687 \stopMarkdownConTeXtDlPrototype}%
14688 \def\markdownRendererDlEndTightPrototype{%
14689 \stopMarkdownConTeXtDlTightPrototype}%
14690 \def\markdownRendererEmphasisPrototype#1{\em#1}%
14691 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
14692 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
14693 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
14694 \def\markdownRendererLineBlockBeginPrototype{%
14695 \begingroup
14696 \def\markdownRendererHardLineBreak{
14697 }%
14698 \startlines
14699 }%
14700 \def\markdownRendererLineBlockEndPrototype{%
14701 \stoplines
14702 \endgroup
14703 }%
14704 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

14705 \ExplSyntaxOn
14706 \cs_gset:Npn
14707 \markdownRendererInputFencedCodePrototype#1#2#3
14708 {
14709 \tl_if_empty:nTF
14710 { #2 }
14711 { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
\startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
Hello world!
\end{document}
~~~

```

```

\stopmarkdown
\stoptext

```

```

14712     {
14713         \regex_extract_once:nnN
14714         { \w* }
14715         { #2 }
14716         \l_tmpa_seq
14717         \seq_pop_left:NN
14718         \l_tmpa_seq
14719         \l_tmpa_tl
14720         \typefile[\l_tmpa_tl] []{#1}
14721     }
14722 }
14723 \ExplSyntaxOff
14724 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
14725 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
14726 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
14727 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
14728 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
14729 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
14730 \def\markdownRendererThematicBreakPrototype{%
14731     \blackrule[height=1pt, width=\hsize]}%
14732 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
14733 \def\markdownRendererTickedBoxPrototype{\boxtimes$}
14734 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
14735 \def\markdownRendererUntickedBoxPrototype{\square$}
14736 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
14737 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
14738 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
14739 \def\markdownRendererDisplayMathPrototype#1{%
14740     \startformula#1\stopformula}%

```

3.4.3.2 Tables

There is a basic implementation of tables.

```

14741 \newcount\markdownConTeXtRowCounter
14742 \newcount\markdownConTeXtRowTotal
14743 \newcount\markdownConTeXtColumnCounter
14744 \newcount\markdownConTeXtColumnTotal
14745 \newtoks\markdownConTeXtTable
14746 \newtoks\markdownConTeXtTableFloat
14747 \def\markdownRendererTablePrototype#1#2#3{%
14748     \markdownConTeXtTable={}%
14749     \ifx\empty#1\empty
14750         \markdownConTeXtTableFloat={%
14751             \the\markdownConTeXtTable}%

```

```

14752 \else
14753   \markdownConTeXtTableFloat={%
14754     \placetable{#1}{\the\markdownConTeXtTable}}%
14755   \fi
14756   \beginngroup
14757   \setupTABLE[r][each][topframe=off, bottomframe=off,
14758     leftframe=off, rightframe=off]
14759   \setupTABLE[c][each][topframe=off, bottomframe=off,
14760     leftframe=off, rightframe=off]
14761   \setupTABLE[r][1][topframe=on, bottomframe=on]
14762   \setupTABLE[r][#1][bottomframe=on]
14763   \markdownConTeXtRowCounter=0%
14764   \markdownConTeXtRowTotal=#2%
14765   \markdownConTeXtColumnTotal=#3%
14766   \markdownConTeXtRenderTableRow}
14767 \def\markdownConTeXtRenderTableRow#1{%
14768   \markdownConTeXtColumnCounter=0%
14769   \ifnum\markdownConTeXtRowCounter=0\relax
14770     \markdownConTeXtReadAlignments#1%
14771     \markdownConTeXtTable={\bTABLE}%
14772   \else
14773     \markdownConTeXtTable=\expandafter{%
14774       \the\markdownConTeXtTable\bTR}%
14775     \markdownConTeXtRenderTableCell#1%
14776     \markdownConTeXtTable=\expandafter{%
14777       \the\markdownConTeXtTable\eTR}%
14778   \fi
14779   \advance\markdownConTeXtRowCounter by 1\relax
14780   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
14781     \markdownConTeXtTable=\expandafter{%
14782       \the\markdownConTeXtTable\eTABLE}%
14783     \the\markdownConTeXtTableFloat
14784     \endgroup
14785     \expandafter\gobbleoneargument
14786   \fi\markdownConTeXtRenderTableRow}
14787 \def\markdownConTeXtReadAlignments#1{%
14788   \advance\markdownConTeXtColumnCounter by 1\relax
14789   \if#1d%
14790     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
14791   \fi\if#1l%
14792     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
14793   \fi\if#1c%
14794     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
14795   \fi\if#1r%
14796     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
14797   \fi
14798   \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax

```

```

14799 \else
14800   \expandafter\gobbleoneargument
14801 \fi\markdownConTeXtReadAlignments}
14802 \def\markdownConTeXtRenderTableCell#1{%
14803   \advance\markdownConTeXtColumnCounter by 1\relax
14804   \markdownConTeXtTable=\expandafter{%
14805     \the\markdownConTeXtTable\bTD#1\eTD}%
14806   \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
14807   \else
14808     \expandafter\gobbleoneargument
14809   \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`.

```

14810 \ExplSyntaxOn
14811 \cs_gset:Npn
14812   \markdownRendererInputRawInlinePrototype#1#2
14813   {
14814     \str_case:nnF
14815       { #2 }
14816       {
14817         { latex }
14818         {
14819           \@@_plain_tex_default_input_raw_inline:nn
14820             { #1 }
14821             { context }
14822         }
14823       }
14824     {
14825       \@@_plain_tex_default_input_raw_inline:nn
14826         { #1 }
14827         { #2 }
14828     }
14829   }
14830 \cs_gset:Npn
14831   \markdownRendererInputRawBlockPrototype#1#2
14832   {
14833     \str_case:nnF
14834       { #2 }
14835       {
14836         { context }
14837         {
14838           \@@_plain_tex_default_input_raw_block:nn
14839             { #1 }
14840             { tex }

```

```

14841     }
14842   }
14843   {
14844     \@@_plain_tex_default_input_raw_block:nn
14845     { #1 }
14846     { #2 }
14847   }
14848 }
14849 \cs_gset_eq:NN
14850   \markdownRendererInputRawBlockPrototype
14851   \markdownRendererInputRawInlinePrototype
14852 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}
14853 \ExplSyntaxOff
14854 \stopmodule
14855 \protect

```

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

```

14856 \markdownIfOption{noDefaults}{}{
14857   \setupmarkdown[theme=witiko/markdown/defaults]
14858 }
14859 \stopmodule
14860 \protect

```

References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [8] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [9] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [10] Geoffrey M. Poore. *The minted Package. Highlighted source code in L^AT_EX*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [11] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [12] Johannes Braams et al. *The L^AT_EX_{2 ϵ} Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [13] Donald Ervin Knuth. *T_EX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [14] Victor Eijkhout. *T_EX by Topic. A T_EXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

Index

autoIdentifiers	21, 33, 77, 90
blankBeforeBlockquote	21
blankBeforeCodeFence	21
blankBeforeDivFence	22
blankBeforeHeading	22
blankBeforeList	22
bracketedSpans	23, 78
breakableBlockquotes	23
cacheDir	4, 17, 19, 57, 58, 139, 152, 353, 375, 391
citationNbsps	23
citations	24, 81
codeSpans	24
contentBlocks	20, 25, 34
contentBlocksLanguageMap	20
contentLevel	25

debugExtensions	9, 20, 26, 297
debugExtensionsFileName	20, 26
defaultOptions	10, 51, 351, 353
definitionLists	26, 85
eagerCache	17, 351
ensureJekyllData	27
entities.char_entity	198
entities.dec_entity	198
entities.hex_entity	198
entities.hex_entity_with_x_char	198
escape_minimal	202
escape_programmatic_text	202
escape_typographic_text	202
expandtabs	257
expectJekyllData	27, 27
extensions	28, 147, 302
extensions.bracketed_spans	303
extensions.citations	304
extensions.content_blocks	308
extensions.definition_lists	311
extensions.fancy_lists	313
extensions.fenced_code	319
extensions.fenced_divs	325
extensions.header_attributes	329
extensions.inline_code_attributes	331
extensions.jekyll_data	347
extensions.line_blocks	331
extensions.link_attributes	333
extensions.mark	332
extensions.notes	334
extensions.pipe_table	336
extensions.raw_inline	341
extensions.strike_through	342
extensions.subscripts	342
extensions.superscripts	343
extensions.tex_math	344
fancyLists	30, 99–104, 392
fencedCode	30, 39, 82, 89, 105, 390
fencedCodeAttributes	31, 77, 89
fencedDiv	89

fencedDivs	31, 41
finalizeCache	17, 20, 32, 32, 57, 58, 138, 351, 352
frozenCache	20, 32, 58, 138, 142, 390, 391
frozenCacheCounter	32, 352, 383
frozenCacheFileName	20, 32, 57, 352
gfmAutoIdentifiers	21, 32, 77, 90
hashEnumerators	33
headerAttributes	33, 41, 77, 90
html	34, 92, 93, 403
hybrid	34, 34, 40, 46, 48, 60, 69, 106, 139, 203, 258, 383
inlineCodeAttributes	35, 77, 83
inlineNotes	36
\input	55
\inputmarkdown	144, 144, 145, 416
inputTempFileName	58, 60, 376–378, 380
iterlines	257
jekyllData	3, 27, 28, 36, 114–116, 118
\l_file_search_path_seq	382
languages_json	308, 308
lineBlocks	37, 95
linkAttributes	37, 77, 94, 97, 278, 414
mark	38, 98, 413
\markdown	136, 137, 386
markdown	136, 136, 137, 385
markdown*	136, 136, 138, 385
\markdown_jekyll_data_concatenate_address:NN	369
\markdown_jekyll_data_pop:	369
\markdown_jekyll_data_push:nN	369
\markdown_jekyll_data_push_address_segment:n	368
\markdown_jekyll_data_set_keyval:Nn	370
\markdown_jekyll_data_set_keyvals:nn	370
\markdown_jekyll_data_update_address_tls:	369
\markdownBegin	53, 53–55, 134, 136, 144
\markdownCleanup	376
\markdownConvert	375
\markdownEnd	53, 53–55, 134, 136, 137, 144
\markdownError	134, 134

<code>\markdownEscape</code>	53, 55, 384
<code>\markdownIfOption</code>	56
<code>\markdownIfSnippetExists</code>	71
<code>\markdownInfo</code>	134, 134
<code>\markdownInput</code>	53, 55, 136, 138, 144, 381, 385
<code>\markdownInputFilename</code>	374
<code>\markdownInputFileStream</code>	376
<code>\markdownInputPlainTeX</code>	385
<code>\markdownLoadPlainTeXTheme</code>	140, 146, 362
<code>\markdownLuaExecute</code>	378, 381
<code>\markdownLuaOptions</code>	372, 375
<code>\markdownMakeOther</code>	134, 415, 416
<code>\markdownOptionFinalizeCache</code>	57
<code>\markdownOptionFrozenCache</code>	57
<code>\markdownOptionHybrid</code>	60
<code>\markdownOptionInputTempFileName</code>	58
<code>\markdownOptionNoDefaults</code>	59
<code>\markdownOptionOutputDir</code>	58, 58, 61, 62
<code>\markdownOptionPlain</code>	59
<code>\markdownOptionStripPercentSigns</code>	60
<code>\markdownOutputFileStream</code>	376
<code>\markdownPrepare</code>	375
<code>\markdownPrepareInputFilename</code>	374
<code>\markdownPrepareLuaOptions</code>	372
<code>\markdownReadAndConvert</code>	134, 376, 385, 386, 416
<code>\markdownReadAndConvertProcessLine</code>	377, 378
<code>\markdownReadAndConvertStripPercentSigns</code>	377
<code>\markdownReadAndConvertTab</code>	376
<code>\markdownRendererAttributeClassName</code>	77
<code>\markdownRendererAttributeIdentifier</code>	77
<code>\markdownRendererAttributeKeyValue</code>	77
<code>\markdownRendererBlockQuoteBegin</code>	78
<code>\markdownRendererBlockQuoteEnd</code>	78
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	78
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	78
<code>\markdownRendererCite</code>	81, 82
<code>\markdownRendererCodeSpan</code>	83
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	83
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	83
<code>\markdownRendererContentBlock</code>	84, 84
<code>\markdownRendererContentBlockCode</code>	84
<code>\markdownRendererContentBlockOnlineImage</code>	84

<code>\markdownRendererDisplayMath</code>	111
<code>\markdownRendererDlBegin</code>	85
<code>\markdownRendererDlBeginTight</code>	85
<code>\markdownRendererDlDefinitionBegin</code>	86
<code>\markdownRendererDlDefinitionEnd</code>	87
<code>\markdownRendererDlEnd</code>	87
<code>\markdownRendererDlEndTight</code>	87
<code>\markdownRendererDlItem</code>	86
<code>\markdownRendererDlItemEnd</code>	86
<code>\markdownRendererDocumentBegin</code>	98
<code>\markdownRendererDocumentEnd</code>	98
<code>\markdownRendererEllipsis</code>	42, 88
<code>\markdownRendererEmphasis</code>	88, 122
<code>\markdownRendererError</code>	113
<code>\markdownRendererFancyOlBegin</code>	100, 100
<code>\markdownRendererFancyOlBeginTight</code>	100
<code>\markdownRendererFancyOlEnd</code>	103
<code>\markdownRendererFancyOlEndTight</code>	104
<code>\markdownRendererFancyOlItem</code>	102
<code>\markdownRendererFancyOlItemEnd</code>	102
<code>\markdownRendererFancyOlItemWithNumber</code>	102
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	89
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	89
<code>\markdownRendererFencedDivAttributeContextBegin</code>	89
<code>\markdownRendererFencedDivAttributeContextEnd</code>	89
<code>\markdownRendererHalfTickedBox</code>	112
<code>\markdownRendererHardLineBreak</code>	96
<code>\markdownRendererHeaderAttributeContextBegin</code>	90
<code>\markdownRendererHeaderAttributeContextEnd</code>	90
<code>\markdownRendererHeadingFive</code>	92
<code>\markdownRendererHeadingFour</code>	91
<code>\markdownRendererHeadingOne</code>	91
<code>\markdownRendererHeadingSix</code>	92
<code>\markdownRendererHeadingThree</code>	91
<code>\markdownRendererHeadingTwo</code>	91
<code>\markdownRendererImage</code>	94
<code>\markdownRendererImageAttributeContextBegin</code>	94
<code>\markdownRendererImageAttributeContextEnd</code>	94
<code>\markdownRendererInlineHtmlComment</code>	92
<code>\markdownRendererInlineHtmlTag</code>	93
<code>\markdownRendererInlineMath</code>	111
<code>\markdownRendererInputBlockHtmlElement</code>	93

<code>\markdownRendererInputFencedCode</code>	82
<code>\markdownRendererInputRawBlock</code>	105
<code>\markdownRendererInputRawInline</code>	104
<code>\markdownRendererInputVerbatim</code>	82
<code>\markdownRendererInterblockSeparator</code>	95
<code>\markdownRendererJekyllDataBegin</code>	114
<code>\markdownRendererJekyllDataBoolean</code>	116
<code>\markdownRendererJekyllDataEmpty</code>	118
<code>\markdownRendererJekyllDataEnd</code>	114
<code>\markdownRendererJekyllDataMappingBegin</code>	114
<code>\markdownRendererJekyllDataMappingEnd</code>	115
<code>\markdownRendererJekyllDataNumber</code>	116
<code>\markdownRendererJekyllDataProgrammaticString</code>	116, 116, 117
<code>\markdownRendererJekyllDataSequenceBegin</code>	115
<code>\markdownRendererJekyllDataSequenceEnd</code>	115
<code>\markdownRendererJekyllDataString</code>	117, 120
<code>\markdownRendererJekyllDataStringPrototype</code>	130
<code>\markdownRendererJekyllDataTypographicString</code>	116, 116, 117, 348
<code>\markdownRendererLineBlockBegin</code>	95
<code>\markdownRendererLineBlockEnd</code>	95
<code>\markdownRendererLink</code>	97, 122
<code>\markdownRendererLinkAttributeContextBegin</code>	97
<code>\markdownRendererLinkAttributeContextEnd</code>	97
<code>\markdownRendererMark</code>	98
<code>\markdownRendererNbsp</code>	99
<code>\markdownRendererNote</code>	99
<code>\markdownRendererOlBegin</code>	99
<code>\markdownRendererOlBeginTight</code>	100
<code>\markdownRendererOlEnd</code>	103
<code>\markdownRendererOlEndTight</code>	103
<code>\markdownRendererOlItem</code>	42, 101
<code>\markdownRendererOlItemEnd</code>	101
<code>\markdownRendererOlItemWithNumber</code>	42, 101
<code>\markdownRendererParagraphSeparator</code>	95
<code>\markdownRendererReplacementCharacter</code>	106
<code>\markdownRendererSectionBegin</code>	105
<code>\markdownRendererSectionEnd</code>	105
<code>\markdownRendererSoftLineBreak</code>	96
<code>\markdownRendererStrikeThrough</code>	109
<code>\markdownRendererStrongEmphasis</code>	88
<code>\markdownRendererSubscript</code>	109
<code>\markdownRendererSuperscript</code>	109

<code>\markdownRendererTable</code>	110
<code>\markdownRendererTableAttributeContextBegin</code>	110
<code>\markdownRendererTableAttributeContextEnd</code>	110
<code>\markdownRendererTextCite</code>	81
<code>\markdownRendererThematicBreak</code>	112
<code>\markdownRendererTickedBox</code>	112
<code>\markdownRendererUlBegin</code>	79
<code>\markdownRendererUlBeginTight</code>	79
<code>\markdownRendererUlEnd</code>	80
<code>\markdownRendererUlEndTight</code>	81
<code>\markdownRendererUlItem</code>	80
<code>\markdownRendererUlItemEnd</code>	80
<code>\markdownRendererUntickedBox</code>	112
<code>\markdownRendererWarning</code>	113
<code>\markdownSetup</code>	56, 56, 60, 138, 145, 386, 392
<code>\markdownSetupSnippet</code>	70, 70
<code>\markdownWarning</code>	134, 134
<code>\markinline</code>	53, 54, 55, 136, 137, 380, 384
<code>\markinlinePlainTeX</code>	384
<code>new</code>	8, 17, 18, 351, 353
<code>notes</code>	38, 99
<code>parsers</code>	218, 256, 257
<code>parsers.punctuation</code>	219
<code>pipeTables</code>	7, 39, 45, 110
<code>preserveTabs</code>	39, 43, 257
<code>rawAttribute</code>	35, 39, 40, 104, 105
<code>reader</code>	8, 29, 147, 218, 256, 302
<code>reader->add_special_character</code>	8, 10, 29, 296
<code>reader->auto_link_email</code>	286
<code>reader->auto_link_url</code>	286
<code>reader->create_parser</code>	257
<code>reader->finalize_grammar</code>	292, 358
<code>reader->initialize_named_group</code>	297
<code>reader->insert_pattern</code>	8, 9, 10, 29, 292, 293, 298, 299
<code>reader->lookup_note_reference</code>	271
<code>reader->lookup_reference</code>	271
<code>reader->normalize_tag</code>	257
<code>reader->options</code>	256
<code>reader->parser_functions</code>	257
<code>reader->parser_functions.name</code>	257

reader->parsers	256, 256, 257
reader->register_link	271
reader->update_rule	292, 295, 299
reader->writer	256
reader.new	256, 256, 358
relativeReferences	40
\setupmarkdown	145
shiftHeadings	7, 41
singletonCache	17
slice	7, 41, 199, 211, 212
smartEllipses	42, 88, 139
\startmarkdown	144, 144, 416
startNumber	42, 101, 102
\stopmarkdown	144, 144, 416
strikeThrough	42, 109, 413
stripIndent	43, 258
stripPercentSigns	376, 377
subscripts	43, 109
superscripts	44, 109
syntax	294, 298
tableAttributes	44, 110
tableCaptions	7, 44, 45, 110
taskLists	45, 112, 402
texComments	46, 258
texMathDollars	35, 46, 111
texMathDoubleBackslash	35, 47, 111
texMathSingleBackslash	35, 47, 111
tightLists	47, 79, 81, 85, 87, 100, 103, 104, 392
underscores	48
unicodeNormalization	18, 19
unicodeNormalizationForm	18, 19
util.cache	147, 148
util.cache_verbatim	148
util.encode_json_string	148
util.err	147
util.escaper	151
util.expand_tabs_in_line	148
util.flatten	149
util.intersperse	150
util.map	150

util.pathname	151
util.rope_last	150
util.rope_to_string	150
util.salt	152
util.table_copy	148
util.walk	149, 150
util.warning	152
walkable_syntax	9, 20, 26, 292, 293, 295–298
writer	147, 147, 199, 302
writer->active_attributes	210, 210–212
writer->attribute_type_levels	211
writer->attributes	208
writer->block_html_element	207
writer->blockquote	207
writer->bulletitem	205
writer->bulletlist	205
writer->citations	304
writer->code	204
writer->contentblock	309
writer->defer_call	218, 218
writer->definitionlist	311
writer->display_math	344
writer->div_begin	325
writer->div_end	325
writer->document	208
writer->ellipsis	201
writer->emphasis	207
writer->error	203
writer->escape	203
writer->escaped_chars	202, 202
writer->escaped_minimal_strings	201, 202
writer->escaped_strings	202
writer->escaped_uri_chars	201, 202
writer->fancyitem	314
writer->fancylist	313
writer->fencedCode	320
writer->flatten_inlines	199, 199
writer->get_state	217
writer->hard_line_break	201
writer->heading	216
writer->identifier	203

writer->image	204
writer->infostring	203
writer->inline_html_comment	206
writer->inline_html_tag	206
writer->inline_math	344
writer->interblocksep	200
writer->is_writing	199, 199
writer->jekyllData	348
writer->lineblock	331
writer->link	204
writer->mark	332
writer->math	203
writer->nbsp	200
writer->note	335
writer->options	199
writer->ordereditem	206
writer->orderedlist	205
writer->paragraph	200
writer->paragraphsep	201
writer->plain	200
writer->pop_attributes	211, 211, 212
writer->push_attributes	211, 211, 212
writer->rawBlock	320
writer->rawInline	341
writer->set_state	217
writer->slice_begin	199
writer->slice_end	199
writer->soft_line_break	201
writer->space	200
writer->span	303
writer->strike_through	342
writer->string	203
writer->strong	207
writer->subscript	342
writer->superscript	343
writer->table	338
writer->thematic_break	201
writer->checkbox	207
writer->undosep	201, 301
writer->uri	203
writer->verbatim	207
writer->warning	152, 203

writer.new

199, 199, 358