

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.22.0-0-g5a3d0fe
2023-04-02

Contents

1	Introduction	1	3	Implementation	121
1.1	Requirements	2	3.1	Lua Implementation . . .	121
1.2	Feedback	6	3.2	Plain T _E X Implementation	261
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation . .	279
2	Interfaces	7	3.4	ConT _E Xt Implementation	307
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	47	References		314
2.3	L ^A T _E X Interface	98	Index		315
2.4	ConT _E Xt Interface	117			

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 . .	43
3	A sequence diagram of typesetting a document using the Lua CLI	43
4	Various formats of mathematical formulae	105
5	The banner of the Markdown package	106
6	A pushdown automaton that recognizes T _E X comments	187

1 Introduction

The Markdown package¹ converts markdown² markup to T_EX commands. The functionality is provided both as a Lua module and as plain T_EX, L^AT_EX, and ConT_EXt macro packages that can be used to directly typeset T_EX documents containing markdown markup. Unlike other convertors, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the

¹See <https://ctan.org/pkg/markdown>.

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

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

```

1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown to plain TeX",
4   author    = "John MacFarlane, Hans Hagen, Vít Novotný",
5   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
6               "2016-2023 Vít Novotný"},
7   license   = "LPPL 1.3c"
8 }
9
10 if not modules then modules = { } end
11 modules['markdown'] = metadata

```

1.1 Requirements

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

1.1.1 Lua Requirements

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

LPeg ≥ 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg ≥ 0.10 is included in LuaTeX $\geq 0.72.0$ (TeXLive ≥ 2013).

```

12 local lpeg = require("lpeg")

```

Selene Unicode A library that provides support for the processing of wide strings. It is used by the Lunamark library to cast image, link, and note tags to the lower case. Selene Unicode is included in all releases of LuaTeX (TeXLive ≥ 2008).

```

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

```

If the Selene Unicode library is unavailable (could be because we are using LuaMetaTeX) and we are using Lua ≥ 5.3 , we will use the built-in support for Unicode.

```

17   if not ran_ok then

```

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

```

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

```

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

```

21 local md5 = require("md5");

```

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

```

22 (function()

```

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

```

23   local should_initialize = package.loaded.kpse == nil
24                           or tex.initialize ~= nil
25   local ran_ok
26   ran_ok, kpse = pcall(require, "kpse")
27   if ran_ok and should_initialize then
28     kpse.set_program_name("luatex")
29   end

```

If the Kpathsea library is unavailable, we will look up files only in the current working directory.

```

30   if not ran_ok then
31     kpse = {lookup = function(f, _) return f end}
32   end
33 end)()

```

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

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

```

34 local uni_case
35 (function()
36   local ran_ok
37   ran_ok, uni_case = pcall(require, "lua-uni-case")

```

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

```

38   if not ran_ok then
39       if unicode.utf8.lower then
40           uni_case = {casefold = unicode.utf8.lower}
41       else

```

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

```

42           uni_case = {casefold = string.lower}
43       end
44   end
45 end)()

```

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

1.1.2 Plain TeX Requirements

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

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

```

46 <@@=markdown>
47 \ifx\ExplSyntaxOn\undefined
48   \input expl3-generic\relax
49 \fi

```

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

The plain 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 ≥ 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 2.1.6), 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) or if you enforce the use of shell using the `\markdownMode` macro, 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,

```
50 \NeedsTeXFormat{LaTeX2e}%
```

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 2.2.4 and 3.3.4) or L^ATeX themes (see Section 2.3.2.3) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.2):

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

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

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

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L^ATeX theme (see Section 2.3.2.3).

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

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

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

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

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the [witiko/graphicx/http](#) L^AT_EX theme, see Section 2.3.2.3.

graphicx A package that builds upon the graphics package, which is part of the L^AT_EX 2_ε kernel. It provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.

grffile A package that extends the name processing of the graphics package to support a larger range of file names in 2006 ≤ T_EX Live ≤ 2019. Since T_EX Live ≥ 2020, the functionality of the package has been integrated in the L^AT_EX 2_ε kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#) L^AT_EX themes, see Section 2.3.2.3.

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

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

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

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

```
51 \RequirePackage{expl3}
```

1.1.4 ConT_EXt Prerequisites

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

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

1.2 Feedback

Please use the Markdown project page on GitHub⁴ to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T_EX-L^AT_EX 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 \TeX implementation of the package draws inspiration from several sources including the source code of $\text{\LaTeX} 2_{\epsilon}$, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from \TeX , 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 \TeX 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 \TeX *token renderers* is exposed by the Lua layer. The plain \TeX layer exposes the conversion capabilities of Lua as \TeX macros. The \LaTeX and Con \TeX t layers provide syntactic sugar on top of plain \TeX 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 \TeX . This interface is used by the plain \TeX 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.

```
52 local M = {metadata = metadata}
```

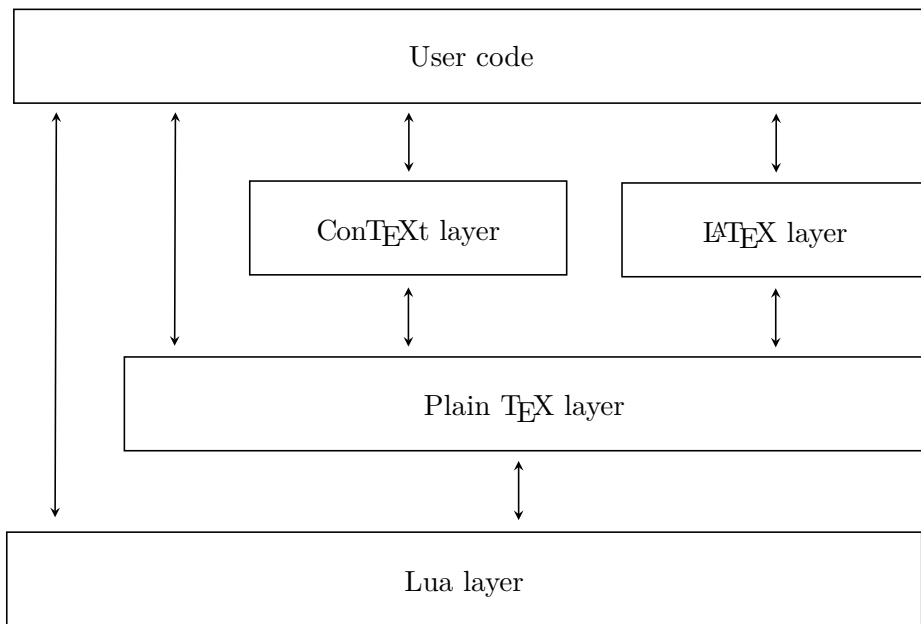


Figure 1: A block diagram of the Markdown package

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.

```
53 local walkable_syntax = {
54   Block = {
55     "Blockquote",
56     "Verbatim",
57     "ThematicBreak",
58     "BulletList",
59     "OrderedList",
60     "Heading",
61     "DisplayHtml",
62     "Paragraph",
63     "Plain",
64   },
65   Inline = {
66     "Str",
67     "Space",
68     "Endline",
69     "U1OrStarLine",
70     "Strong",
71     "Emph",
72     "Link",
73     "Image",
74     "Code",
75     "AutoLinkUrl",
76     "AutoLinkEmail",
77     "AutoLinkRelativeReference",
78     "InlineHtml",
79     "HtmlEntity",
80     "EscapedChar",
81     "Smart",
82     "Symbol",
83   },
84 }
```

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

For example. if we'd like to insert `pattern` into the grammar between the `Inline -> Emph` and `Inline -> Link` rules, we would call `reader->insert_pattern`

with "Inline after Emph" (or "Inline before Link") and `pattern` as the arguments.

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

2.1.3 Options

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

```
85 local defaultOptions = {}
```

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

```
86 \ExplSyntaxOn
87 \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.

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

```

114     { #2 }
115     { #4 }
116     \@@_typecheck_option:n
117     { #2 }
118 }
119 \cs_generate_variant:Nn
120 \@@_add_option:nnnn
121 { Vnnn }
122 \tl_const:Nn \c_@@_option_value_true_tl { true }
123 \tl_const:Nn \c_@@_option_value_false_tl { false }
124 \cs_new:Nn \@@_typecheck_option:n
125 {
126     \@@_get_option_type:nN
127     { #1 }
128     \l_tmpa_tl
129     \str_case_e:Vn
130     \l_tmpa_tl
131     {
132         { \c_@@_option_type_boolean_tl }
133         {
134             \@@_get_option_value:nN
135             { #1 }
136             \l_tmpa_tl
137             \bool_if:nF
138             {
139                 \str_if_eq_p:VV
140                 \l_tmpa_tl
141                 \c_@@_option_value_true_tl ||
142                 \str_if_eq_p:VV
143                 \l_tmpa_tl
144                 \c_@@_option_value_false_tl
145             }
146             {
147                 \msg_error:nnnV
148                 { @@ }
149                 { failed-typecheck-for-boolean-option }
150                 { #1 }
151                 \l_tmpa_tl
152             }
153         }
154     }
155 }
156 \msg_new:nnn
157 { @@ }
158 { failed-typecheck-for-boolean-option }
159 {
160     Option~#1~has~value~#2,~

```

```

161     but~a~boolean~(true~or~false)~was~expected.
162 }
163 \cs_generate_variant:Nn
164   \str_case_e:nn
165   { Vn }
166 \cs_generate_variant:Nn
167   \msg_error:nnnn
168   { nnnV }
169 \seq_new:N \g_@@_option_types_seq
170 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
171 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
172 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
173 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
174 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
175 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
176 \tl_const:Nn \c_@@_option_type_number_tl { number }
177 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
178 \tl_const:Nn \c_@@_option_type_path_tl { path }
179 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
180 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
181 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
182 \tl_const:Nn \c_@@_option_type_string_tl { string }
183 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
184 \cs_new:Nn
185   \@@_get_option_type:nN
186   {
187     \bool_set_false:N
188       \l_tmpa_bool
189     \seq_map_inline:Nn
190       \g_@@_option_layers_seq
191       {
192         \prop_get:cnNT
193           { g_@@_ ##1 _option_types_prop }
194           { #1 }
195         \l_tmpa_tl
196         {
197           \bool_set_true:N
198             \l_tmpa_bool
199           \seq_map_break:
200         }
201       }
202     \bool_if:nF
203       \l_tmpa_bool
204     {
205       \msg_error:nnn
206         { @@ }
207         { undefined-option }

```

```

208         { #1 }
209     }
210     \seq_if_in:NVF
211     \g_@@_option_types_seq
212     \l_tmpa_tl
213     {
214         \msg_error:nnnV
215         { @@ }
216         { unknown-option-type }
217         { #1 }
218         \l_tmpa_tl
219     }
220     \tl_set_eq:NN
221     #2
222     \l_tmpa_tl
223 }
224 \msg_new:nnn
225 { @@ }
226 { unknown-option-type }
227 {
228     Option~#1~has~unknown~type~#2.
229 }
230 \msg_new:nnn
231 { @@ }
232 { undefined-option }
233 {
234     Option~#1~is~undefined.
235 }
236 \cs_new:Nn
237 \@@_get_default_option_value:nN
238 {
239     \bool_set_false:N
240     \l_tmpa_bool
241     \seq_map_inline:Nn
242     \g_@@_option_layers_seq
243     {
244         \prop_get:cnNT
245         { g_@@_default_ ##1 _options_prop }
246         { #1 }
247         #2
248         {
249             \bool_set_true:N
250             \l_tmpa_bool
251             \seq_map_break:
252         }
253     }
254     \bool_if:nF

```

```

255     \l_tmpa_bool
256     {
257         \msg_error:nnn
258         { @@ }
259         { undefined-option }
260         { #1 }
261     }
262 }
263 \cs_new:Nn
264 \@@_get_option_value:nN
265 {
266     \@@_option_tl_to_csname:nN
267     { #1 }
268     \l_tmpa_tl
269     \cs_if_free:cTF
270     { \l_tmpa_tl }
271     {
272         \@@_get_default_option_value:nN
273         { #1 }
274         #2
275     }
276     {
277         \@@_get_option_type:nN
278         { #1 }
279         \l_tmpa_tl
280         \str_if_eq:NNTF
281         \c_@@_option_type_counter_tl
282         \l_tmpa_tl
283         {
284             \@@_option_tl_to_csname:nN
285             { #1 }
286             \l_tmpa_tl
287             \tl_set:Nx
288             #2
289             { \the \cs:w \l_tmpa_tl \cs_end: }
290         }
291         {
292             \@@_option_tl_to_csname:nN
293             { #1 }
294             \l_tmpa_tl
295             \tl_set:Nv
296             #2
297             { \l_tmpa_tl }
298         }
299     }
300 }
301 \cs_new:Nn \@@_option_tl_to_csname:nN

```

```

302 {
303   \tl_set:Nn
304     \l_tmpa_tl
305     { \str_uppercase:n { #1 } }
306   \tl_set:Nx
307     #2
308     {
309       markdownOption
310       \tl_head:f { \l_tmpa_tl }
311       \tl_tail:n { #1 }
312     }
313 }
314 \seq_new:N \g_@@_cases_seq
315 \cs_new:Nn \@@_with_various_cases:nn
316 {
317   \seq_clear:N
318     \l_tmpa_seq
319   \seq_map_inline:Nn
320     \g_@@_cases_seq
321     {
322       \tl_set:Nn
323         \l_tmpa_tl
324         { #1 }
325       \use:c { ##1 }
326       \l_tmpa_tl
327       \seq_put_right:NV
328         \l_tmpa_seq
329         \l_tmpa_tl
330     }
331   \seq_map_inline:Nn
332     \l_tmpa_seq
333     { #2 }
334 }
335 \cs_new:Nn \@@_camel_case:N
336 {
337   \regex_replace_all:nnN
338     { _ ([a-z]) }
339     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
340     #1
341   \tl_set:Nx
342     #1
343     { #1 }
344 }
345 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
346 \cs_new:Nn \@@_snake_case:N
347 {
348   \regex_replace_all:nnN

```

```

349      { ([a-z])([A-Z]) }
350      { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
351      #1
352      \tl_set:Nx
353      #1
354      { #1 }
355    }
356    \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 File and Directory Names

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

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain T_EX implementations. The Lua implementation expects that the entire path already exists.

When iteratively writing and typesetting a markdown document, the cache files are going to accumulate over time. You are advised to clean the cache directory every now and then, or to set it to a temporary filesystem (such as `/tmp` on UN*X systems), which gets periodically emptied.

```

357 \@@_add_lua_option:nnn
358   { cacheDir }
359   { path }
360   { \markdownOptionOutputDir / _markdown_\jobname }
361 defaultOptions.cacheDir = "."

```

`contentBlocksLanguageMap`= $\langle filename \rangle$
 default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA,Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.3.8 for more information.

```

362 \@@_add_lua_option:nnn
363   { contentBlocksLanguageMap }
364   { path }
365   { markdown-languages.json }
366 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"

```


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

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

```
367 \@@_add_lua_option:nnn
368   { debugExtensionsFileName }
369   { path }
370   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
371 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

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

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

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

```
372 \@@_add_lua_option:nnn
373   { frozenCacheFileName }
374   { path }
375   { \markdownOptionCacheDir / frozenCache.tex }
376 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

2.1.5 Parser Options

`blankBeforeBlockquote`=`true, false` default: `false`

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

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

```
382 \@@_add_lua_option:nnn
383   { blankBeforeCodeFence }
384   { boolean }
385   { false }

386 defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeDivFence=true, false` default: false

true Require a blank line before the closing fence of a fenced div.

false Do not require a blank line before the closing fence of a fenced div.

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

391 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.

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

396 defaultOptions.blankBeforeHeading = false
```

`bracketedSpans=true, false`

default: `false`

true Enable the Pandoc bracketed span syntax extension⁶:

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

false Disable the Pandoc bracketed span syntax extension.

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

401 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false`

default: `false`

true A blank line separates block quotes.

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

```
402 \@@_add_lua_option:nnn
403   { breakableBlockquotes }
404   { boolean }
405   { false }

406 defaultOptions.breakableBlockquotes = false
```

`citationNbsps=true, false`

default: `false`

true Replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations produced via the pandoc citation syntax extension.

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

```
407 \@@_add_lua_option:nnn
408   { citationNbsps }
409   { boolean }
410   { true }

411 defaultOptions.citationNbsps = true
```

⁶See https://pandoc.org/MANUAL.html#extension-bracketed_spans.

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension⁷:

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.

```
412 \@@_add_lua_option:nnn
413   { citations }
414   { boolean }
415   { false }
416 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.``

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

⁷See <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.

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

426 defaultOptions.contentBlocks = false
```

`debugExtensions=true, false`

default: false

`true`

Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the [walkable\\_syntax](#) hash table) after built-in syntax extensions (see Section 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the [debugExtensionsFileName](#) option.

`false`

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

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

431 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.

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

`eagerCache=true, false`

default: true

**true** Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing. This behavior will always be used if the `finalizeCache` option is enabled.

**false** Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing. This behavior will only be used when the `finalizeCache` option is disabled. Recursive nesting of markdown document fragments is undefined behavior when `eagerCache` is disabled.

```
437 \@@_add_lua_option:nnn
438   { eagerCache }
439   { boolean }
440   { true }
```

441 `defaultOptions.eagerCache = true`

`expectJekyllData=true, false`

default: `false`

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

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

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

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

```

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

```

```

442 \@@_add_lua_option:nnn
443   { expectJekyllData }
444   { boolean }
445   { false }

446 defaultOptions.expectJekyllData = false

```

`extensions=<filenames>`

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

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

```

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

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

    reader.insert_pattern("Inline after Emph", read_strike_through,
      "StrikeThrough")
    reader.add_special_character("/")
  end
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:

```
447 metadata.user_extension_api_version = 2
448 metadata.grammar_version = 2
```

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

The `finalize_grammar` field is a function that finalizes the grammar of markdown using the interface of a Lua `\luamref{reader}` object, such as the `\luamref{reader->insert_pattern}` and `\luamref{reader->add_special_character}` methods, see Section `<#luauserextensions>`.

```
449 \cs_generate_variant:Nn
450   \@@_add_lua_option:nnn
451   { nnV }
452 \@@_add_lua_option:nnV
453   { extensions }
454   { clist }
455   \c_empty_clist
456 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: false

**true** Enable the Pandoc fancy list syntax extension<sup>8</sup>:

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

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

<sup>8</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

```

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

461 defaultOptions.fancyLists = false

```

`fencedCode=true, false`

default: false

**true** Enable the commonmark fenced code block extension:

```

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

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

```

**false** Disable the commonmark fenced code block extension.

```

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

466 defaultOptions.fencedCode = false

```

`fencedCodeAttributes=true, false`

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>9</sup>:

```

~~~~ {#mycode .haskell .numberLines startFrom=100}
qsort [] = []

```

<sup>9</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-fenced_code_attributes).

```
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
 qsort (filter (>= x) xs)
~~~~~
```

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

```
467 \@@_add_lua_option:nnn
468   { fencedCodeAttributes }
469   { boolean }
470   { false }
471 defaultOptions.fencedCodeAttributes = false
```

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

**true**      Enable the Pandoc fenced div syntax extension<sup>10</sup>:

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

And another.
:::::
```

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

```
472 \@@_add_lua_option:nnn
473   { fencedDivs }
474   { boolean }
475   { false }
476 defaultOptions.fencedDivs = false
```

**finalizeCache=true, false** default: false

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

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

<sup>10</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

```

477 \@@_add_lua_option:nnn
478   { finalizeCache }
479   { boolean }
480   { false }
481 defaultOptions.finalizeCache = false

```

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

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

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

```

482 \@@_add_lua_option:nnn
483   { frozenCacheCounter }
484   { counter }
485   { 0 }
486 defaultOptions.frozenCacheCounter = 0

```

**hardLineBreaks**=true, false default: false

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

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

```

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

```

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

```

491 defaultOptions.hardLineBreaks = false

```

`hashEnumerators=true, false`

default: false

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

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

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

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

496 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.

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

501 defaultOptions.headerAttributes = false
```

`html=true, false`

default: false

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

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

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

506 defaultOptions.html = false
```

**hybrid=true, false**      default: false

**true**      Disable the escaping of special plain T<sub>E</sub>X characters, which makes it possible to intersperse your markdown markup with T<sub>E</sub>X code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix T<sub>E</sub>X and markdown markup freely.

**false**      Enable the escaping of special plain T<sub>E</sub>X characters outside verbatim environments, so that they are not interpreted by T<sub>E</sub>X. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```
507 \@@_add_lua_option:nnn
508   { hybrid }
509   { boolean }
510   { false }

511 defaultOptions.hybrid = false
```

**inlineCodeAttributes=true, false**      default: false

**true**      Enable the Pandoc inline code span attribute extension<sup>11</sup>:

``<$>`{.haskell}`

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

```
512 \@@_add_lua_option:nnn
513   { inlineCodeAttributes }
514   { boolean }
515   { false }

516 defaultOptions.inlineCodeAttributes = false
```

---

<sup>11</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-inline_code_attributes).

`inlineNotes=true, false`

default: false

**true** Enable the Pandoc inline note syntax extension<sup>12</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.

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

```
517 \@@_add_lua_option:nnn
518   { inlineFootnotes }
519   { boolean }
520   { false }
521 \@@_add_lua_option:nnn
522   { inlineNotes }
523   { boolean }
524   { false }

525 defaultOptions.inlineFootnotes = false
526 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`

default: false

**true** Enable the Pandoc YAML metadata block syntax extension<sup>13</sup> for entering metadata in YAML:

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

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

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

<sup>12</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).

```

527 \@@_add_lua_option:nnn
528   { jekyllData }
529   { boolean }
530   { false }

531 defaultOptions.jekyllData = false

```

`linkAttributes=true, false`

default: false

**true** Enable the Pandoc link and image attribute syntax extension<sup>14</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.

```

532 \@@_add_lua_option:nnn
533   { linkAttributes }
534   { boolean }
535   { false }

536 defaultOptions.linkAttributes = false

```

`lineBlocks=true, false`

default: false

**true** Enable the Pandoc line block syntax extension<sup>15</sup>:

```

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

```

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

```

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

541 defaultOptions.lineBlocks = false

```

<sup>14</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).



`notes=true, false`

default: false

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

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

```
542 \@@_add_lua_option:nnn
543   { footnotes }
544   { boolean }
545   { false }
546 \@@_add_lua_option:nnn
547   { notes }
548   { boolean }
549   { false }

550 defaultOptions.footnotes = false
551 defaultOptions.notes = false
```

`pipeTables=true, false`

default: false

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

```
Right	Left	Default	Center
```

<sup>16</sup>See <https://pandoc.org/MANUAL.html#extension-footnotes>.

|  |     |  |     |  |     |  |     |  |
|--|-----|--|-----|--|-----|--|-----|--|
|  | 12  |  | 12  |  | 12  |  | 12  |  |
|  | 123 |  | 123 |  | 123 |  | 123 |  |
|  | 1   |  | 1   |  | 1   |  | 1   |  |

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

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

556 defaultOptions.pipeTables = false
```

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

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

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

```
557 \@@_add_lua_option:nnn
558 { preserveTabs }
559 { boolean }
560 { false }

561 defaultOptions.preserveTabs = false
```

**rawAttribute=true, false** default: false

**true**      Enable the Pandoc raw attribute syntax extension<sup>17</sup>:

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

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

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

<sup>17</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).

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.

```
562 \@@_add_lua_option:nnn
563   { rawAttribute }
564   { boolean }
565   { false }
566 defaultOptions.rawAttribute = true
```

`relativeReferences=true, false` default: false

`true`      Enable relative references<sup>18</sup> in autolinks:

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

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

`false`      Disable relative references in autolinks.

```
567 \@@_add_lua_option:nnn
568   { relativeReferences }
569   { boolean }
570   { false }
571 defaultOptions.relativeReferences = false
```

`shiftHeadings=<shift amount>` default: 0

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

```
572 \@@_add_lua_option:nnn
573   { shiftHeadings }
574   { number }
575   { 0 }
576 defaultOptions.shiftHeadings = 0
```

<sup>18</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

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

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

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

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

```
577 \@@_add_lua_option:nnn
578   { slice }
579   { slice }
580   { ^~$ }

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

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

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

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

```
582 \@@_add_lua_option:nnn
583   { smartEllipses }
584   { boolean }
585   { false }

586 defaultOptions.smartEllipses = false
```

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

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

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

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

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

**true** Enable the Pandoc strike-through syntax extension<sup>19</sup>:

This ~~is deleted text.~~

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

```
592 \@@_add_lua_option:nnn
593   { strikeThrough }
594   { boolean }
595   { false }
596 defaultOptions.strikeThrough = false
```

**stripIndent=true, false** default: false

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

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

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

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

---

<sup>19</sup>See <https://pandoc.org/MANUAL.html#extension-strikeout>.

`subscripts=true, false` default: false

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

H~2~O is a liquid.

`false` Disable the Pandoc subscript syntax extension.

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

`superscripts=true, false` default: false

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

2<sup>10</sup> is 1024.

`false` Disable the Pandoc superscript syntax extension.

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

`tableCaptions=true, false` default: false

`true`

: Enable the Pandoc table caption syntax extension<sup>22</sup> for pipe tables (see the `pipeTables` option).

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

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

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

<sup>22</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

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

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

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

**taskLists=true, false**

default: false

**true**      Enable the Pandoc task list syntax extension<sup>23</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.

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

**texComments=true, false**

default: false

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

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

Always enabled when **hybrid** is enabled.

<sup>23</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

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

```
622 \@@_add_lua_option:nnn
623   { texComments }
624   { boolean }
625   { false }

626 defaultOptions.texComments = false
```

**texMathDollars**=true, false default: false

**true** Enable the Pandoc dollar math syntax extension<sup>24</sup>:

<pre>inline math: \$E=mc^2\$  display math: \$\$E=mc^2\$\$</pre>
------------------------------------------------------------------

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

```
627 \@@_add_lua_option:nnn
628   { texMathDollars }
629   { boolean }
630   { false }

631 defaultOptions.texMathDollars = false
```

**texMathDoubleBackslash**=true, false default: false

**true** Enable the Pandoc double backslash math syntax extension<sup>25</sup>:

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

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

```
632 \@@_add_lua_option:nnn
633   { texMathDoubleBackslash }
634   { boolean }
635   { false }

636 defaultOptions.texMathDoubleBackslash = false
```

---

<sup>24</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_dollars](https://pandoc.org/MANUAL.html#extension-tex_math_dollars).

<sup>25</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).



`texMathSingleBackslash=true, false`

default: false

**true** Enable the Pandoc single backslash math syntax extension<sup>26</sup>:

```
inline math: \(\text{E}=\text{mc}^2\)
```

```
display math: \[\text{E}=\text{mc}^2\]
```

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

```
637 \@@_add_lua_option:nnn
638   { texMathSingleBackslash }
639   { boolean }
640   { false }
641 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:

```
- 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.

```
642 \@@_add_lua_option:nnn
643   { tightLists }
644   { boolean }
645   { true }
646 defaultOptions.tightLists = true
```

---

<sup>26</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_single\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash).

`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.

```
647 \@@_add_lua_option:nnn
648   { underscores }
649   { boolean }
650   { true }
651 \ExplSyntaxOff
652 defaultOptions.underscores = true
```

### 2.1.6 Command-Line Interface

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

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

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

```
653
654 local HELP_STRING = [[
655 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
656 where OPTIONS are documented in the Lua interface section of the
657 technical Markdown package documentation.
658
659 When OUTPUT_FILE is unspecified, the result of the conversion will be
660 written to the standard output. When INPUT_FILE is also unspecified, the
```

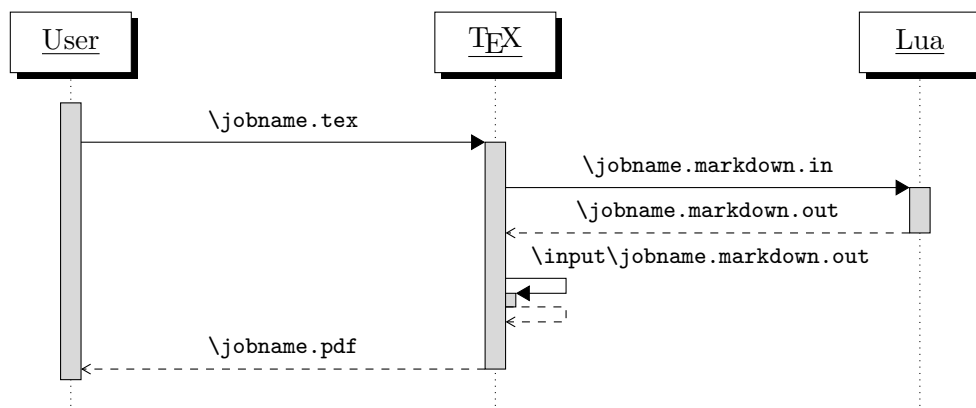


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TeX interface

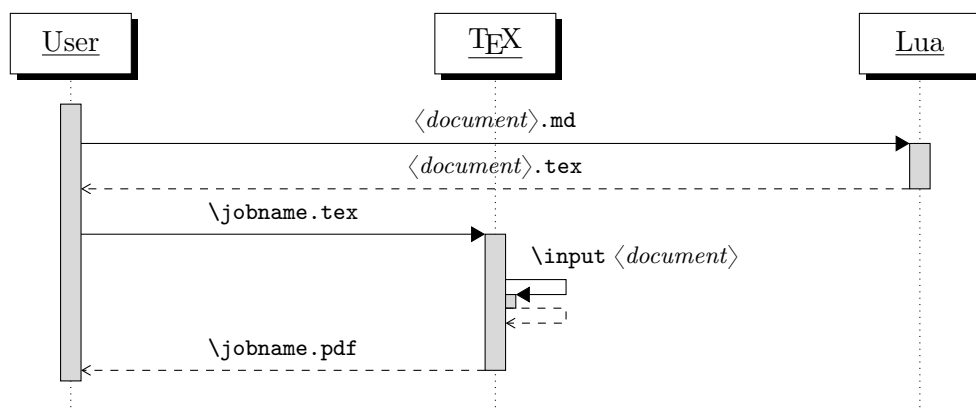


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

661 result of the conversion will be read from the standard input.
662
663 Report bugs to: witiko@mail.muni.cz
664 Markdown package home page: <https://github.com/witiko/markdown>]]
665
666 local VERSION_STRING = [[
667 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
668
669 Copyright (C) ]] .. table.concat(metadata.copyright,
670                                     "\nCopyright (C) ") .. [[
671
672 License: ]] .. metadata.license
673
674 local function warn(s)
675   io.stderr:write("Warning: " .. s .. "\n") end
676
677 local function error(s)
678   io.stderr:write("Error: " .. s .. "\n")
679   os.exit(1)
680 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.

```

681 local function camel_case(option_name)
682   local cased_option_name = option_name:gsub("_(%l)", function(match)
683     return match:sub(2, 2):upper()
684   end)
685   return cased_option_name
686 end
687
688 local function snake_case(option_name)
689   local cased_option_name = option_name:gsub("%l%u", function(match)
690     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
691   end)
692   return cased_option_name
693 end
694
695 local cases = {camel_case, snake_case}
696 local various_case_options = {}
697 for option_name, _ in pairs(defaultOptions) do
698   for _, case in ipairs(cases) do
699     various_case_options[case(option_name)] = option_name
700   end
701 end
702

```

```

703 local process_options = true
704 local options = {}
705 local input_filename
706 local output_filename
707 for i = 1, #arg do
708   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.

```

709     if arg[i] == "--" then
710       process_options = false
711       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.

```

712     elseif arg[i]:match("=") then
713       local key, value = arg[i]:match("(.-)=(.*)")
714       if defaultOptions[key] == nil and
715         various_case_options[key] ~= nil then
716         key = various_case_options[key]
717       end

```

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

```

718       local default_type = type(defaultOptions[key])
719       if default_type == "boolean" then
720         options[key] = (value == "true")
721       elseif default_type == "number" then
722         options[key] = tonumber(value)
723       elseif default_type == "table" then
724         options[key] = {}
725         for item in value:gmatch("[^ ,]+") do
726           table.insert(options[key], item)
727         end
728       else
729         if default_type ~= "string" then
730           if default_type == "nil" then
731             warn('Option "' .. key .. '" not recognized.')
732           else
733             warn('Option "' .. key .. '" type not recognized, please file ' ..
734               'a report to the package maintainer.')
735           end
736           warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
737             key .. '" as a string.')
738         end

```

```

739         options[key] = value
740     end
741     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.

```

742     elseif arg[i] == "--help" or arg[i] == "-h" then
743         print(HELP_STRING)
744         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.

```

745     elseif arg[i] == "--version" or arg[i] == "-v" then
746         print(VERSION_STRING)
747         os.exit()
748     end
749 end

```

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

```

750 if input_filename == nil then
751     input_filename = arg[i]

```

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

```

752 elseif output_filename == nil then
753     output_filename = arg[i]
754 else
755     error('Unexpected argument: "' .. arg[i] .. '".')
756 end
757 ::continue::
758 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  $\text{\TeX}$  document `hello.tex`. After the Markdown package for our  $\text{\TeX}$  format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

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

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

```
759 \def\markdownLastModified{((LASTMODIFIED))}%  
760 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown

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

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

```
761 \let\markdownBegin\relax  
762 \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
```

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.

```
763 \let\markdownInput\relax
```

This macro is not subject to the abovelisted 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.

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



```

765 \ExplSyntaxOn
766 \seq_new:N \g_@@_plain_tex_options_seq

```

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

```

767 \prop_new:N \g_@@_plain_tex_option_types_prop
768 \prop_new:N \g_@@_default_plain_tex_options_prop
769 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
770 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
771 \cs_new:Nn
772   \@@_add_plain_tex_option:nnn
773   {
774     \@@_add_option:Vnnn
775     \c_@@_option_layer_plain_tex_tl
776     { #1 }
777     { #2 }
778     { #3 }
779   }

```

**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`.

```

780 \@@_add_plain_tex_option:nnn
781   { frozenCache }
782   { boolean }
783   { 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 `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain TeX in TeX engines without the `\directlua`

primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

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

```
784 \@@_add_plain_tex_option:nnn
785 { helperScriptFileName }
786 { path }
787 { \jobname.markdown.lua }
```

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

```
788 \str_new:N
789 \g_luabridge_helper_script_filename_str
790 \tl_gset:Nn
791 \g_luabridge_helper_script_filename_str
792 { \markdownOptionHelperScriptFileName }
```

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

```
793 \@@_add_plain_tex_option:nnn
794 { inputTempFileName }
795 { path }
796 { \jobname.markdown.in }
```

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

```
797 \@@_add_plain_tex_option:nnn
798 { outputTempFileName }
799 { path }
800 { \jobname.markdown.out }
```

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

```
801 \str_new:N
802 \g_luabridge_standard_output_filename_str
803 \tl_gset:Nn
804 \g_luabridge_standard_output_filename_str
805 { \markdownOptionOutputTempFileName }
```

The `\markdownOptionErrorTempFileName` macro sets the filename of the temporary output file that is created when a Lua error is encountered during the conversion from markdown to plain T<sub>E</sub>X in `\markdownMode` other than 2. It defaults to

`\jobname.markdown.err`. The same limitations apply here as in the case of the `helperScriptFileName` macro.

```

806 \@@_add_plain_tex_option:nnn
807   { errorTempFileName }
808   { path }
809   { \jobname.markdown.err }

```

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

```

810 \str_new:N
811   \g_luabridge_error_output_filename_str
812 \tl_gset:Nn
813   \g_luabridge_error_output_filename_str
814   { \markdownOptionErrorTempFileName }

```

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

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 `helperScriptFileName` macro.

```

815 \@@_add_plain_tex_option:nnn
816   { outputDir }
817   { path }
818   { . }

```

Here, we automatically define plain  $\TeX$  macros for the above plain  $\TeX$  options.

Furthermore, we also define macros that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain  $\TeX$  implementation, only passed along to Lua.

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

```

819 \cs_new:Nn \@@_plain_tex_define_option_commands:
820   {
821     \seq_map_inline:Nn
822       \g_@@_option_layers_seq
823       {
824         \seq_map_inline:cn
825           { g_@@_ ##1 _options_seq }
826           {
827             \@@_plain_tex_define_option_command:n
828               { ####1 }

```

```

829         }
830     }
831 }
832 \cs_new:Nn \@@_plain_tex_define_option_command:n
833 {
834     \@@_get_default_option_value:nN
835     { #1 }
836     \l_tmpa_tl
837     \@@_set_option_value:nV
838     { #1 }
839     \l_tmpa_tl
840 }
841 \cs_new:Nn
842 \@@_set_option_value:nn
843 {
844     \@@_define_option:n
845     { #1 }
846     \@@_get_option_type:nN
847     { #1 }
848     \l_tmpa_tl
849     \str_if_eq:NNTF
850     \c_@@_option_type_counter_tl
851     \l_tmpa_tl
852     {
853         \@@_option_tl_to_csname:nN
854         { #1 }
855         \l_tmpa_tl
856         \int_gset:cn
857         { \l_tmpa_tl }
858         { #2 }
859     }
860     {
861         \@@_option_tl_to_csname:nN
862         { #1 }
863         \l_tmpa_tl
864         \cs_set:cpn
865         { \l_tmpa_tl }
866         { #2 }
867     }
868 }
869 \cs_generate_variant:Nn
870 \@@_set_option_value:nn
871 { nV }
872 \cs_new:Nn
873 \@@_define_option:n
874 {
875     \@@_option_tl_to_csname:nN

```

```

876     { #1 }
877     \l_tmpa_tl
878     \cs_if_free:cT
879     { \l_tmpa_tl }
880     {
881         \@@_get_option_type:nN
882         { #1 }
883         \l_tmpb_tl
884         \str_if_eq:NNT
885         \c_@@_option_type_counter_tl
886         \l_tmpb_tl
887         {
888             \@@_option_tl_to_csname:nN
889             { #1 }
890             \l_tmpa_tl
891             \int_new:c
892             { \l_tmpa_tl }
893         }
894     }
895 }
896 \@@_plain_tex_define_option_commands:

```

**2.2.2.3 Miscellaneous Options** The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.4) or not. Notably, this enables the use of markdown when writing T<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`.

```

897 \seq_gput_right:Nn
898 \g_@@_plain_tex_options_seq
899 { stripPercentSigns }
900 \prop_gput:Nnn
901 \g_@@_plain_tex_option_types_prop
902 { stripPercentSigns }
903 { boolean }
904 \prop_gput:Nnx
905 \g_@@_default_plain_tex_options_prop
906 { stripPercentSigns }
907 { false }
908 \ExplSyntaxOff

```

## 2.2.3 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 2.2.4).

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

```
909 \ExplSyntaxOn
910 \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.

```
911 \prop_new:N \g_@@_renderer_arities_prop
912 \ExplSyntaxOff
```

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

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

`\markdownRendererAttributeClassName` represents the  $\langle class name \rangle$  of a mark-down element (`class="⟨class name⟩ ..."` in HTML and `.⟨class name⟩` in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the  $\langle class name \rangle$ .

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

```
913 \def\markdownRendererAttributeIdentifier{%
914   \markdownRendererAttributeIdentifierPrototype}%
915 \ExplSyntaxOn
916 \seq_gput_right:Nn
917   \g_@@_renderers_seq
918   { attributeIdentifier }
919 \prop_gput:Nnn
920   \g_@@_renderer_arities_prop
921   { attributeIdentifier }
922   { 1 }
923 \ExplSyntaxOff
924 \def\markdownRendererAttributeClassName{%
925   \markdownRendererAttributeClassNamePrototype}%
926 \ExplSyntaxOn
927 \seq_gput_right:Nn
928   \g_@@_renderers_seq
929   { attributeClassName }
930 \prop_gput:Nnn
931   \g_@@_renderer_arities_prop
932   { attributeClassName }
```

```

933 { 1 }
934 \ExplSyntaxOff
935 \def\markdownRendererAttributeKeyValue{%
936 \markdownRendererAttributeKeyValuePrototype}%
937 \ExplSyntaxOn
938 \seq_gput_right:Nn
939 \g_@@_renderers_seq
940 { attributeKeyValue }
941 \prop_gput:Nnn
942 \g_@@_renderer_arities_prop
943 { attributeKeyValue }
944 { 2 }
945 \ExplSyntaxOff

```

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

```

946 \def\markdownRendererBlockQuoteBegin{%
947 \markdownRendererBlockQuoteBeginPrototype}%
948 \ExplSyntaxOn
949 \seq_gput_right:Nn
950 \g_@@_renderers_seq
951 { blockQuoteBegin }
952 \prop_gput:Nnn
953 \g_@@_renderer_arities_prop
954 { blockQuoteBegin }
955 { 0 }
956 \ExplSyntaxOff

```

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

```

957 \def\markdownRendererBlockQuoteEnd{%
958 \markdownRendererBlockQuoteEndPrototype}%
959 \ExplSyntaxOn
960 \seq_gput_right:Nn
961 \g_@@_renderers_seq
962 { blockQuoteEnd }
963 \prop_gput:Nnn
964 \g_@@_renderer_arities_prop
965 { blockQuoteEnd }
966 { 0 }
967 \ExplSyntaxOff

```

**2.2.3.3 Bracketed Spans Attribute Context Renderers** The following macros are only produced, when the `bracketedSpans` option is enabled.

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

```

968 \def\markdownRendererBracketedSpanAttributeContextBegin{%
969   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
970 \ExplSyntaxOn
971 \seq_gput_right:Nn
972   \g_@@_renderers_seq
973   { bracketedSpanAttributeContextBegin }
974 \prop_gput:Nnn
975   \g_@@_renderer_arities_prop
976   { bracketedSpanAttributeContextBegin }
977   { 0 }
978 \ExplSyntaxOff
979 \def\markdownRendererBracketedSpanAttributeContextEnd{%
980   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
981 \ExplSyntaxOn
982 \seq_gput_right:Nn
983   \g_@@_renderers_seq
984   { bracketedSpanAttributeContextEnd }
985 \prop_gput:Nnn
986   \g_@@_renderer_arities_prop
987   { bracketedSpanAttributeContextEnd }
988   { 0 }
989 \ExplSyntaxOff

```

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

```

990 \def\markdownRendererUlBegin{%
991   \markdownRendererUlBeginPrototype}%
992 \ExplSyntaxOn
993 \seq_gput_right:Nn
994   \g_@@_renderers_seq
995   { ulBegin }
996 \prop_gput:Nnn
997   \g_@@_renderer_arities_prop
998   { ulBegin }
999   { 0 }
1000 \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.



```

1001 \def\markdownRendererUlBeginTight{%
1002   \markdownRendererUlBeginTightPrototype}%
1003 \ExplSyntaxOn
1004 \seq_gput_right:Nn
1005   \g_@@_renderers_seq
1006   { ulBeginTight }
1007 \prop_gput:Nnn
1008   \g_@@_renderer_arities_prop
1009   { ulBeginTight }
1010   { 0 }
1011 \ExplSyntaxOff

```

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

```

1012 \def\markdownRendererUlItem{%
1013   \markdownRendererUlItemPrototype}%
1014 \ExplSyntaxOn
1015 \seq_gput_right:Nn
1016   \g_@@_renderers_seq
1017   { ulItem }
1018 \prop_gput:Nnn
1019   \g_@@_renderer_arities_prop
1020   { ulItem }
1021   { 0 }
1022 \ExplSyntaxOff

```

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

```

1023 \def\markdownRendererUlItemEnd{%
1024   \markdownRendererUlItemEndPrototype}%
1025 \ExplSyntaxOn
1026 \seq_gput_right:Nn
1027   \g_@@_renderers_seq
1028   { ulItemEnd }
1029 \prop_gput:Nnn
1030   \g_@@_renderer_arities_prop
1031   { ulItemEnd }
1032   { 0 }
1033 \ExplSyntaxOff

```

The `\markdownRendererUlEnd` 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.

```

1034 \def\markdownRendererUlEnd{%
1035   \markdownRendererUlEndPrototype}%
1036 \ExplSyntaxOn

```

```

1037 \seq_gput_right:Nn
1038   \g_@@_renderers_seq
1039   { ulEnd }
1040 \prop_gput:Nnn
1041   \g_@@_renderer_arities_prop
1042   { ulEnd }
1043   { 0 }
1044 \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.

```

1045 \def\markdownRendererUEndTight{%
1046   \markdownRendererUEndTightPrototype}%
1047 \ExplSyntaxOn
1048 \seq_gput_right:Nn
1049   \g_@@_renderers_seq
1050   { ulEndTight }
1051 \prop_gput:Nnn
1052   \g_@@_renderer_arities_prop
1053   { ulEndTight }
1054   { 0 }
1055 \ExplSyntaxOff

```

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

```

1056 \def\markdownRendererInputVerbatim{%
1057   \markdownRendererInputVerbatimPrototype}%
1058 \ExplSyntaxOn
1059 \seq_gput_right:Nn
1060   \g_@@_renderers_seq
1061   { inputVerbatim }
1062 \prop_gput:Nnn
1063   \g_@@_renderer_arities_prop
1064   { inputVerbatim }
1065   { 1 }
1066 \ExplSyntaxOff

```

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

```

1067 \def\markdownRendererInputFencedCode{%

```

```

1068 \markdownRendererInputFencedCodePrototype}%
1069 \ExplSyntaxOn
1070 \seq_gput_right:Nn
1071 \g_@@_renderers_seq
1072 { inputFencedCode }
1073 \prop_gput:Nnn
1074 \g_@@_renderer_arities_prop
1075 { inputFencedCode }
1076 { 2 }
1077 \ExplSyntaxOff

```

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

```

1078 \def\markdownRendererCodeSpan{%
1079 \markdownRendererCodeSpanPrototype}%
1080 \ExplSyntaxOn
1081 \seq_gput_right:Nn
1082 \g_@@_renderers_seq
1083 { codeSpan }
1084 \prop_gput:Nnn
1085 \g_@@_renderer_arities_prop
1086 { codeSpan }
1087 { 1 }
1088 \ExplSyntaxOff

```

**2.2.3.7 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 an inline code span in which the attributes of the inline code span apply. The macros receive no arguments.

```

1089 \def\markdownRendererCodeSpanAttributeContextBegin{%
1090 \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1091 \ExplSyntaxOn
1092 \seq_gput_right:Nn
1093 \g_@@_renderers_seq
1094 { codeSpanAttributeContextBegin }
1095 \prop_gput:Nnn
1096 \g_@@_renderer_arities_prop
1097 { codeSpanAttributeContextBegin }
1098 { 0 }
1099 \ExplSyntaxOff
1100 \def\markdownRendererCodeSpanAttributeContextEnd{%
1101 \markdownRendererCodeSpanAttributeContextEndPrototype}%
1102 \ExplSyntaxOn

```

```

1103 \seq_gput_right:Nn
1104   \g_@@_renderers_seq
1105   { codeSpanAttributeContextEnd }
1106 \prop_gput:Nnn
1107   \g_@@_renderer_arities_prop
1108   { codeSpanAttributeContextEnd }
1109   { 0 }
1110 \ExplSyntaxOff

```

**2.2.3.8 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.

```

1111 \def\markdownRendererContentBlock{%
1112   \markdownRendererContentBlockPrototype}%
1113 \ExplSyntaxOn
1114 \seq_gput_right:Nn
1115   \g_@@_renderers_seq
1116   { contentBlock }
1117 \prop_gput:Nnn
1118   \g_@@_renderer_arities_prop
1119   { contentBlock }
1120   { 4 }
1121 \ExplSyntaxOff

```

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

```

1122 \def\markdownRendererContentBlockOnlineImage{%
1123   \markdownRendererContentBlockOnlineImagePrototype}%
1124 \ExplSyntaxOn
1125 \seq_gput_right:Nn
1126   \g_@@_renderers_seq
1127   { contentBlockOnlineImage }
1128 \prop_gput:Nnn
1129   \g_@@_renderer_arities_prop
1130   { contentBlockOnlineImage }
1131   { 4 }
1132 \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>27</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s$ ,  $s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

1133 \def\markdownRendererContentBlockCode{%
1134   \markdownRendererContentBlockCodePrototype}%
1135 \ExplSyntaxOn
1136 \seq_gput_right:Nn
1137   \g_@@_renderers_seq
1138   { contentBlockCode }
1139 \prop_gput:Nnn
1140   \g_@@_renderer_arities_prop
1141   { contentBlockCode }
1142   { 5 }
1143 \ExplSyntaxOff

```

**2.2.3.9 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.

```

1144 \def\markdownRendererDlBegin{%
1145   \markdownRendererDlBeginPrototype}%
1146 \ExplSyntaxOn
1147 \seq_gput_right:Nn
1148   \g_@@_renderers_seq
1149   { dlBegin }
1150 \prop_gput:Nnn
1151   \g_@@_renderer_arities_prop
1152   { dlBegin }
1153   { 0 }
1154 \ExplSyntaxOff

```

---

<sup>27</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

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

```

1155 \def\markdownRendererDlBeginTight{%
1156   \markdownRendererDlBeginTightPrototype}%
1157 \ExplSyntaxOn
1158 \seq_gput_right:Nn
1159   \g_@@_renderers_seq
1160   { dlBeginTight }
1161 \prop_gput:Nnn
1162   \g_@@_renderer_arities_prop
1163   { dlBeginTight }
1164   { 0 }
1165 \ExplSyntaxOff

```

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

```

1166 \def\markdownRendererDlItem{%
1167   \markdownRendererDlItemPrototype}%
1168 \ExplSyntaxOn
1169 \seq_gput_right:Nn
1170   \g_@@_renderers_seq
1171   { dlItem }
1172 \prop_gput:Nnn
1173   \g_@@_renderer_arities_prop
1174   { dlItem }
1175   { 1 }
1176 \ExplSyntaxOff

```

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

```

1177 \def\markdownRendererDlItemEnd{%
1178   \markdownRendererDlItemEndPrototype}%
1179 \ExplSyntaxOn
1180 \seq_gput_right:Nn
1181   \g_@@_renderers_seq
1182   { dlItemEnd }
1183 \prop_gput:Nnn
1184   \g_@@_renderer_arities_prop
1185   { dlItemEnd }
1186   { 0 }
1187 \ExplSyntaxOff

```

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

```

1188 \def\markdownRendererDlDefinitionBegin{%
1189   \markdownRendererDlDefinitionBeginPrototype}%
1190 \ExplSyntaxOn
1191 \seq_gput_right:Nn
1192   \g_@@_renderers_seq
1193   { dlDefinitionBegin }
1194 \prop_gput:Nnn
1195   \g_@@_renderer_arities_prop
1196   { dlDefinitionBegin }
1197   { 0 }
1198 \ExplSyntaxOff

```

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

```

1199 \def\markdownRendererDlDefinitionEnd{%
1200   \markdownRendererDlDefinitionEndPrototype}%
1201 \ExplSyntaxOn
1202 \seq_gput_right:Nn
1203   \g_@@_renderers_seq
1204   { dlDefinitionEnd }
1205 \prop_gput:Nnn
1206   \g_@@_renderer_arities_prop
1207   { dlDefinitionEnd }
1208   { 0 }
1209 \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.

```

1210 \def\markdownRendererDlEnd{%
1211   \markdownRendererDlEndPrototype}%
1212 \ExplSyntaxOn
1213 \seq_gput_right:Nn
1214   \g_@@_renderers_seq
1215   { dlEnd }
1216 \prop_gput:Nnn
1217   \g_@@_renderer_arities_prop
1218   { dlEnd }
1219   { 0 }
1220 \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.

```

1221 \def\markdownRendererDlEndTight{%

```

```

1222 \markdownRendererDlEndTightPrototype}%
1223 \ExplSyntaxOn
1224 \seq_gput_right:Nn
1225 \g_@@_renderers_seq
1226 { dlEndTight }
1227 \prop_gput:Nnn
1228 \g_@@_renderer_arities_prop
1229 { dlEndTight }
1230 { 0 }
1231 \ExplSyntaxOff

```

**2.2.3.10 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.

```

1232 \def\markdownRendererEllipsis{%
1233 \markdownRendererEllipsisPrototype}%
1234 \ExplSyntaxOn
1235 \seq_gput_right:Nn
1236 \g_@@_renderers_seq
1237 { ellipsis }
1238 \prop_gput:Nnn
1239 \g_@@_renderer_arities_prop
1240 { ellipsis }
1241 { 0 }
1242 \ExplSyntaxOff

```

**2.2.3.11 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.

```

1243 \def\markdownRendererEmphasis{%
1244 \markdownRendererEmphasisPrototype}%
1245 \ExplSyntaxOn
1246 \seq_gput_right:Nn
1247 \g_@@_renderers_seq
1248 { emphasis }
1249 \prop_gput:Nnn
1250 \g_@@_renderer_arities_prop
1251 { emphasis }
1252 { 1 }
1253 \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.



```

1254 \def\markdownRendererStrongEmphasis{%
1255   \markdownRendererStrongEmphasisPrototype}%
1256 \ExplSyntaxOn
1257 \seq_gput_right:Nn
1258   \g_@@_renderers_seq
1259   { strongEmphasis }
1260 \prop_gput:Nnn
1261   \g_@@_renderer_arities_prop
1262   { strongEmphasis }
1263   { 1 }
1264 \ExplSyntaxOff

```

**2.2.3.12 Fenced Code Attribute Context Renderers** The following macros are only produced, when the `fencedCode` option is enabled.

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

```

1265 \def\markdownRendererFencedCodeAttributeContextBegin{%
1266   \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1267 \ExplSyntaxOn
1268 \seq_gput_right:Nn
1269   \g_@@_renderers_seq
1270   { fencedCodeAttributeContextBegin }
1271 \prop_gput:Nnn
1272   \g_@@_renderer_arities_prop
1273   { fencedCodeAttributeContextBegin }
1274   { 0 }
1275 \ExplSyntaxOff
1276 \def\markdownRendererFencedCodeAttributeContextEnd{%
1277   \markdownRendererFencedCodeAttributeContextEndPrototype}%
1278 \ExplSyntaxOn
1279 \seq_gput_right:Nn
1280   \g_@@_renderers_seq
1281   { fencedCodeAttributeContextEnd }
1282 \prop_gput:Nnn
1283   \g_@@_renderer_arities_prop
1284   { fencedCodeAttributeContextEnd }
1285   { 0 }
1286 \ExplSyntaxOff

```

**2.2.3.13 Fenced Div Attribute Context Renderers** The following macros are only produced, when the `fencedDiv` option is enabled.

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

```

1287 \def\markdownRendererFencedDivAttributeContextBegin{%
1288   \markdownRendererFencedDivAttributeContextBeginPrototype}%
1289 \ExplSyntaxOn
1290 \seq_gput_right:Nn
1291   \g_@@_renderers_seq
1292   { fencedDivAttributeContextBegin }
1293 \prop_gput:Nnn
1294   \g_@@_renderer_arities_prop
1295   { fencedDivAttributeContextBegin }
1296   { 0 }
1297 \ExplSyntaxOff
1298 \def\markdownRendererFencedDivAttributeContextEnd{%
1299   \markdownRendererFencedDivAttributeContextEndPrototype}%
1300 \ExplSyntaxOn
1301 \seq_gput_right:Nn
1302   \g_@@_renderers_seq
1303   { fencedDivAttributeContextEnd }
1304 \prop_gput:Nnn
1305   \g_@@_renderer_arities_prop
1306   { fencedDivAttributeContextEnd }
1307   { 0 }
1308 \ExplSyntaxOff

```

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

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

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

```

1309 \def\markdownRendererHeaderAttributeContextBegin{%
1310   \markdownRendererHeaderAttributeContextBeginPrototype}%
1311 \ExplSyntaxOn
1312 \seq_gput_right:Nn
1313   \g_@@_renderers_seq
1314   { headerAttributeContextBegin }
1315 \prop_gput:Nnn
1316   \g_@@_renderer_arities_prop
1317   { headerAttributeContextBegin }
1318   { 0 }
1319 \ExplSyntaxOff

```

```

1320 \def\markdownRendererHeaderAttributeContextEnd{%
1321   \markdownRendererHeaderAttributeContextEndPrototype}%
1322 \ExplSyntaxOn
1323 \seq_gput_right:Nn
1324   \g_@@_renderers_seq
1325   { headerAttributeContextEnd }
1326 \prop_gput:Nnn
1327   \g_@@_renderer_arities_prop
1328   { headerAttributeContextEnd }
1329   { 0 }
1330 \ExplSyntaxOff

```

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

```

1331 \def\markdownRendererHeadingOne{%
1332   \markdownRendererHeadingOnePrototype}%
1333 \ExplSyntaxOn
1334 \seq_gput_right:Nn
1335   \g_@@_renderers_seq
1336   { headingOne }
1337 \prop_gput:Nnn
1338   \g_@@_renderer_arities_prop
1339   { headingOne }
1340   { 1 }
1341 \ExplSyntaxOff

```

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

```

1342 \def\markdownRendererHeadingTwo{%
1343   \markdownRendererHeadingTwoPrototype}%
1344 \ExplSyntaxOn
1345 \seq_gput_right:Nn
1346   \g_@@_renderers_seq
1347   { headingTwo }
1348 \prop_gput:Nnn
1349   \g_@@_renderer_arities_prop
1350   { headingTwo }
1351   { 1 }
1352 \ExplSyntaxOff

```

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

```

1353 \def\markdownRendererHeadingThree{%
1354   \markdownRendererHeadingThreePrototype}%
1355 \ExplSyntaxOn

```

```

1356 \seq_gput_right:Nn
1357   \g_@@_renderers_seq
1358   { headingThree }
1359 \prop_gput:Nnn
1360   \g_@@_renderer_arities_prop
1361   { headingThree }
1362   { 1 }
1363 \ExplSyntaxOff

```

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

```

1364 \def\markdownRendererHeadingFour{%
1365   \markdownRendererHeadingFourPrototype}%
1366 \ExplSyntaxOn
1367 \seq_gput_right:Nn
1368   \g_@@_renderers_seq
1369   { headingFour }
1370 \prop_gput:Nnn
1371   \g_@@_renderer_arities_prop
1372   { headingFour }
1373   { 1 }
1374 \ExplSyntaxOff

```

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

```

1375 \def\markdownRendererHeadingFive{%
1376   \markdownRendererHeadingFivePrototype}%
1377 \ExplSyntaxOn
1378 \seq_gput_right:Nn
1379   \g_@@_renderers_seq
1380   { headingFive }
1381 \prop_gput:Nnn
1382   \g_@@_renderer_arities_prop
1383   { headingFive }
1384   { 1 }
1385 \ExplSyntaxOff

```

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

```

1386 \def\markdownRendererHeadingSix{%
1387   \markdownRendererHeadingSixPrototype}%
1388 \ExplSyntaxOn
1389 \seq_gput_right:Nn
1390   \g_@@_renderers_seq
1391   { headingSix }
1392 \prop_gput:Nnn
1393   \g_@@_renderer_arities_prop

```

```

1394 { headingSix }
1395 { 1 }
1396 \ExplSyntaxOff

```

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

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

```

1397 \def\markdownRendererInlineHtmlComment{%
1398   \markdownRendererInlineHtmlCommentPrototype}%
1399 \ExplSyntaxOn
1400 \seq_gput_right:Nn
1401   \g_@@_renderers_seq
1402   { inlineHtmlComment }
1403 \prop_gput:Nnn
1404   \g_@@_renderer_arities_prop
1405   { inlineHtmlComment }
1406   { 1 }
1407 \ExplSyntaxOff
1408 \def\markdownRendererBlockHtmlCommentBegin{%
1409   \markdownRendererBlockHtmlCommentBeginPrototype}%
1410 \ExplSyntaxOn
1411 \seq_gput_right:Nn
1412   \g_@@_renderers_seq
1413   { blockHtmlCommentBegin }
1414 \prop_gput:Nnn
1415   \g_@@_renderer_arities_prop
1416   { blockHtmlCommentBegin }
1417   { 0 }
1418 \ExplSyntaxOff
1419 \def\markdownRendererBlockHtmlCommentEnd{%
1420   \markdownRendererBlockHtmlCommentEndPrototype}%
1421 \ExplSyntaxOn
1422 \seq_gput_right:Nn
1423   \g_@@_renderers_seq
1424   { blockHtmlCommentEnd }
1425 \prop_gput:Nnn
1426   \g_@@_renderer_arities_prop
1427   { blockHtmlCommentEnd }
1428   { 0 }
1429 \ExplSyntaxOff

```

**2.2.3.17 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.

```

1430 \def\markdownRendererInlineHtmlTag{%
1431   \markdownRendererInlineHtmlTagPrototype}%
1432 \ExplSyntaxOn
1433 \seq_gput_right:Nn
1434   \g_@@_renderers_seq
1435   { inlineHtmlTag }
1436 \prop_gput:Nnn
1437   \g_@@_renderer_arities_prop
1438   { inlineHtmlTag }
1439   { 1 }
1440 \ExplSyntaxOff
1441 \def\markdownRendererInputBlockHtmlElement{%
1442   \markdownRendererInputBlockHtmlElementPrototype}%
1443 \ExplSyntaxOn
1444 \seq_gput_right:Nn
1445   \g_@@_renderers_seq
1446   { inputBlockHtmlElement }
1447 \prop_gput:Nnn
1448   \g_@@_renderer_arities_prop
1449   { inputBlockHtmlElement }
1450   { 1 }
1451 \ExplSyntaxOff

```

**2.2.3.18 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.

```

1452 \def\markdownRendererImage{%
1453   \markdownRendererImagePrototype}%
1454 \ExplSyntaxOn
1455 \seq_gput_right:Nn
1456   \g_@@_renderers_seq
1457   { image }
1458 \prop_gput:Nnn
1459   \g_@@_renderer_arities_prop
1460   { image }

```

```

1461 { 4 }
1462 \ExplSyntaxOff

```

**2.2.3.19 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 an image in which the attributes of the image apply. The macros receive no arguments.

```

1463 \def\markdownRendererImageAttributeContextBegin{%
1464   \markdownRendererImageAttributeContextBeginPrototype}%
1465 \ExplSyntaxOn
1466 \seq_gput_right:Nn
1467   \g_@@_renderers_seq
1468   { imageAttributeContextBegin }
1469 \prop_gput:Nnn
1470   \g_@@_renderer_arities_prop
1471   { imageAttributeContextBegin }
1472   { 0 }
1473 \ExplSyntaxOff
1474 \def\markdownRendererImageAttributeContextEnd{%
1475   \markdownRendererImageAttributeContextEndPrototype}%
1476 \ExplSyntaxOn
1477 \seq_gput_right:Nn
1478   \g_@@_renderers_seq
1479   { imageAttributeContextEnd }
1480 \prop_gput:Nnn
1481   \g_@@_renderer_arities_prop
1482   { imageAttributeContextEnd }
1483   { 0 }
1484 \ExplSyntaxOff

```

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

```

1485 \def\markdownRendererInterblockSeparator{%
1486   \markdownRendererInterblockSeparatorPrototype}%
1487 \ExplSyntaxOn
1488 \seq_gput_right:Nn
1489   \g_@@_renderers_seq
1490   { interblockSeparator }
1491 \prop_gput:Nnn
1492   \g_@@_renderer_arities_prop
1493   { interblockSeparator }
1494   { 0 }
1495 \ExplSyntaxOff

```

**2.2.3.21 Line Block Renderer** The following macros are only produced, when the `lineBlocks` option is enabled.

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

```

1496 \def\markdownRendererLineBlockBegin{%
1497   \markdownRendererLineBlockBeginPrototype}%
1498 \ExplSyntaxOn
1499 \seq_gput_right:Nn
1500   \g_@@_renderers_seq
1501   { lineBlockBegin }
1502 \prop_gput:Nnn
1503   \g_@@_renderer_arities_prop
1504   { lineBlockBegin }
1505   { 0 }
1506 \ExplSyntaxOff
1507 \def\markdownRendererLineBlockEnd{%
1508   \markdownRendererLineBlockEndPrototype}%
1509 \ExplSyntaxOn
1510 \seq_gput_right:Nn
1511   \g_@@_renderers_seq
1512   { lineBlockEnd }
1513 \prop_gput:Nnn
1514   \g_@@_renderer_arities_prop
1515   { lineBlockEnd }
1516   { 0 }
1517 \ExplSyntaxOff

```

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

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

```

1518 \ExplSyntaxOn
1519 \cs_new:Npn
1520   \markdownRendererHardLineBreak
1521   {
1522     \cs_if_exist:NTF
1523       \markdownRendererLineBreak
1524       {
1525         \markdownWarning
1526         {
1527           Line~break~renderer~has~been~deprecated,~
1528           to~be~removed~in~Markdown~3.0.0
1529         }
1530       }
1531     \markdownRendererLineBreak

```



```

1531     }
1532     {
1533         \cs_if_exist:NTF
1534         \markdownRendererLineBreakPrototype
1535         {
1536             \markdownWarning
1537             {
1538                 Line~break~renderer~prototype~has~been~deprecated,~
1539                 to~be~removed~in~Markdown~3.0.0
1540             }
1541             \markdownRendererLineBreakPrototype
1542         }
1543         {
1544             \markdownRendererHardLineBreakPrototype
1545         }
1546     }
1547 }
1548 \seq_gput_right:Nn
1549 \g_@@_renderers_seq
1550 { lineBreak }
1551 \prop_gput:Nnn
1552 \g_@@_renderer_arities_prop
1553 { lineBreak }
1554 { 0 }
1555 \seq_gput_right:Nn
1556 \g_@@_renderers_seq
1557 { hardLineBreak }
1558 \prop_gput:Nnn
1559 \g_@@_renderer_arities_prop
1560 { hardLineBreak }
1561 { 0 }
1562 \ExplSyntaxOff

```

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

```

1563 \def\markdownRendererLink{%
1564     \markdownRendererLinkPrototype}%
1565 \ExplSyntaxOn
1566 \seq_gput_right:Nn
1567 \g_@@_renderers_seq
1568 { link }
1569 \prop_gput:Nnn
1570 \g_@@_renderer_arities_prop
1571 { link }
1572 { 4 }

```

1573 \ExplSyntaxOff

**2.2.3.24 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 hyperlink in which the attributes of the hyperlink apply. The macros receive no arguments.

```
1574 \def\markdownRendererLinkAttributeContextBegin{%
1575   \markdownRendererLinkAttributeContextBeginPrototype}%
1576 \ExplSyntaxOn
1577 \seq_gput_right:Nn
1578   \g_@@_renderers_seq
1579   { linkAttributeContextBegin }
1580 \prop_gput:Nnn
1581   \g_@@_renderer_arities_prop
1582   { linkAttributeContextBegin }
1583   { 0 }
1584 \ExplSyntaxOff
1585 \def\markdownRendererLinkAttributeContextEnd{%
1586   \markdownRendererLinkAttributeContextEndPrototype}%
1587 \ExplSyntaxOn
1588 \seq_gput_right:Nn
1589   \g_@@_renderers_seq
1590   { linkAttributeContextEnd }
1591 \prop_gput:Nnn
1592   \g_@@_renderer_arities_prop
1593   { linkAttributeContextEnd }
1594   { 0 }
1595 \ExplSyntaxOff
```

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

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

```
1596 \def\markdownRendererDocumentBegin{%
1597   \markdownRendererDocumentBeginPrototype}%
1598 \ExplSyntaxOn
1599 \seq_gput_right:Nn
1600   \g_@@_renderers_seq
1601   { documentBegin }
1602 \prop_gput:Nnn
```

```

1603 \g_@@_renderer_arities_prop
1604 { documentBegin }
1605 { 0 }
1606 \ExplSyntaxOff
1607 \def\markdownRendererDocumentEnd{%
1608 \markdownRendererDocumentEndPrototype}%
1609 \ExplSyntaxOn
1610 \seq_gput_right:Nn
1611 \g_@@_renderers_seq
1612 { documentEnd }
1613 \prop_gput:Nnn
1614 \g_@@_renderer_arities_prop
1615 { documentEnd }
1616 { 0 }
1617 \ExplSyntaxOff

```

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

```

1618 \def\markdownRendererNbsp{%
1619 \markdownRendererNbspPrototype}%
1620 \ExplSyntaxOn
1621 \seq_gput_right:Nn
1622 \g_@@_renderers_seq
1623 { nbsp }
1624 \prop_gput:Nnn
1625 \g_@@_renderer_arities_prop
1626 { nbsp }
1627 { 0 }
1628 \ExplSyntaxOff

```

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

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

```

1629 \ExplSyntaxOn
1630 \cs_new:Npn
1631 \markdownRendererNote
1632 {
1633 \cs_if_exist:NTF
1634 \markdownRendererFootnote
1635 {
1636 \markdownWarning
1637 {
1638 Footnote~renderer~has~been~deprecated,~

```

```

1639         to~be~removed~in~Markdown~3.0.0
1640     }
1641     \markdownRendererFootnote
1642 }
1643 {
1644     \cs_if_exist:NTF
1645         \markdownRendererFootnotePrototype
1646     {
1647         \markdownWarning
1648         {
1649             Footnote~renderer~prototype~has~been~deprecated,~
1650             to~be~removed~in~Markdown~3.0.0
1651         }
1652         \markdownRendererFootnotePrototype
1653     }
1654     {
1655         \markdownRendererNotePrototype
1656     }
1657 }
1658 }
1659 \seq_gput_right:Nn
1660     \g_@@_renderers_seq
1661     { footnote }
1662 \prop_gput:Nnn
1663     \g_@@_renderer_arities_prop
1664     { footnote }
1665     { 1 }
1666 \seq_gput_right:Nn
1667     \g_@@_renderers_seq
1668     { note }
1669 \prop_gput:Nnn
1670     \g_@@_renderer_arities_prop
1671     { note }
1672     { 1 }
1673 \ExplSyntaxOff

```

**2.2.3.28 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.

```

1674 \def\markdownRendererOlBegin{%
1675     \markdownRendererOlBeginPrototype}%
1676 \ExplSyntaxOn
1677 \seq_gput_right:Nn
1678     \g_@@_renderers_seq
1679     { olBegin }

```

```

1680 \prop_gput:Nnn
1681 \g_@@_renderer_arities_prop
1682 { olBegin }
1683 { 0 }
1684 \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.

```

1685 \def\markdownRendererOlBeginTight{%
1686 \markdownRendererOlBeginTightPrototype}%
1687 \ExplSyntaxOn
1688 \seq_gput_right:Nn
1689 \g_@@_renderers_seq
1690 { olBeginTight }
1691 \prop_gput:Nnn
1692 \g_@@_renderer_arities_prop
1693 { olBeginTight }
1694 { 0 }
1695 \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`).

```

1696 \def\markdownRendererFancyOlBegin{%
1697 \markdownRendererFancyOlBeginPrototype}%
1698 \ExplSyntaxOn
1699 \seq_gput_right:Nn
1700 \g_@@_renderers_seq
1701 { fancyOlBegin }
1702 \prop_gput:Nnn
1703 \g_@@_renderer_arities_prop
1704 { fancyOlBegin }
1705 { 2 }
1706 \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.

```

1707 \def\markdownRendererFancyOlBeginTight{%
1708   \markdownRendererFancyOlBeginTightPrototype}%
1709 \ExplSyntaxOn
1710 \seq_gput_right:Nn
1711   \g_@@_renderers_seq
1712   { fancyOlBeginTight }
1713 \prop_gput:Nnn
1714   \g_@@_renderer_arities_prop
1715   { fancyOlBeginTight }
1716   { 2 }
1717 \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.

```

1718 \def\markdownRendererOlItem{%
1719   \markdownRendererOlItemPrototype}%
1720 \ExplSyntaxOn
1721 \seq_gput_right:Nn
1722   \g_@@_renderers_seq
1723   { olItem }
1724 \prop_gput:Nnn
1725   \g_@@_renderer_arities_prop
1726   { olItem }
1727   { 0 }
1728 \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.

```

1729 \def\markdownRendererOlItemEnd{%
1730   \markdownRendererOlItemEndPrototype}%
1731 \ExplSyntaxOn
1732 \seq_gput_right:Nn
1733   \g_@@_renderers_seq
1734   { olItemEnd }
1735 \prop_gput:Nnn
1736   \g_@@_renderer_arities_prop
1737   { olItemEnd }
1738   { 0 }
1739 \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.

```

1740 \def\markdownRendererOliItemWithNumber{%
1741   \markdownRendererOliItemWithNumberPrototype}%
1742 \ExplSyntaxOn
1743 \seq_gput_right:Nn
1744   \g_@@_renderers_seq
1745   { oliItemWithNumber }
1746 \prop_gput:Nnn
1747   \g_@@_renderer_arities_prop
1748   { oliItemWithNumber }
1749   { 1 }
1750 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItem` 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.

```

1751 \def\markdownRendererFancyOliItem{%
1752   \markdownRendererFancyOliItemPrototype}%
1753 \ExplSyntaxOn
1754 \seq_gput_right:Nn
1755   \g_@@_renderers_seq
1756   { fancyOliItem }
1757 \prop_gput:Nnn
1758   \g_@@_renderer_arities_prop
1759   { fancyOliItem }
1760   { 0 }
1761 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItemEnd` 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.

```

1762 \def\markdownRendererFancyOliItemEnd{%
1763   \markdownRendererFancyOliItemEndPrototype}%
1764 \ExplSyntaxOn
1765 \seq_gput_right:Nn
1766   \g_@@_renderers_seq
1767   { fancyOliItemEnd }
1768 \prop_gput:Nnn
1769   \g_@@_renderer_arities_prop
1770   { fancyOliItemEnd }
1771   { 0 }
1772 \ExplSyntaxOff

```

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

```

1773 \def\markdownRendererFancyOliItemWithNumber{%
1774   \markdownRendererFancyOliItemWithNumberPrototype}%
1775 \ExplSyntaxOn
1776 \seq_gput_right:Nn
1777   \g_@@_renderers_seq
1778   { fancyOliItemWithNumber }
1779 \prop_gput:Nnn
1780   \g_@@_renderer_arities_prop
1781   { fancyOliItemWithNumber }
1782   { 1 }
1783 \ExplSyntaxOff

```

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

```

1784 \def\markdownRendererOliEnd{%
1785   \markdownRendererOliEndPrototype}%
1786 \ExplSyntaxOn
1787 \seq_gput_right:Nn
1788   \g_@@_renderers_seq
1789   { oliEnd }
1790 \prop_gput:Nnn
1791   \g_@@_renderer_arities_prop
1792   { oliEnd }
1793   { 0 }
1794 \ExplSyntaxOff

```

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

```

1795 \def\markdownRendererOliEndTight{%
1796   \markdownRendererOliEndTightPrototype}%
1797 \ExplSyntaxOn
1798 \seq_gput_right:Nn
1799   \g_@@_renderers_seq
1800   { oliEndTight }
1801 \prop_gput:Nnn
1802   \g_@@_renderer_arities_prop
1803   { oliEndTight }
1804   { 0 }
1805 \ExplSyntaxOff

```

The `\markdownRendererFancyOliEnd` 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.

```

1806 \def\markdownRendererFancy01End{%
1807   \markdownRendererFancy01EndPrototype}%
1808 \ExplSyntaxOn
1809 \seq_gput_right:Nn
1810   \g_@@_renderers_seq
1811   { fancy01End }
1812 \prop_gput:Nnn
1813   \g_@@_renderer_arities_prop
1814   { fancy01End }
1815   { 0 }
1816 \ExplSyntaxOff

```

The `\markdownRendererFancy01EndTight` 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.

```

1817 \def\markdownRendererFancy01EndTight{%
1818   \markdownRendererFancy01EndTightPrototype}%
1819 \ExplSyntaxOn
1820 \seq_gput_right:Nn
1821   \g_@@_renderers_seq
1822   { fancy01EndTight }
1823 \prop_gput:Nnn
1824   \g_@@_renderer_arities_prop
1825   { fancy01EndTight }
1826   { 0 }
1827 \ExplSyntaxOff

```

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

```

1828 \def\markdownRendererCite{%
1829   \markdownRendererCitePrototype}%
1830 \ExplSyntaxOn
1831 \seq_gput_right:Nn
1832   \g_@@_renderers_seq
1833   { cite }
1834 \prop_gput:Nnn

```

```

1835 \g_@@_renderer_arities_prop
1836 { cite }
1837 { 1 }
1838 \ExplSyntaxOff

```

**2.2.3.30 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.

```

1839 \def\markdownRendererInputRawInline{%
1840 \markdownRendererInputRawInlinePrototype}%
1841 \ExplSyntaxOn
1842 \seq_gput_right:Nn
1843 \g_@@_renderers_seq
1844 { inputRawInline }
1845 \prop_gput:Nnn
1846 \g_@@_renderer_arities_prop
1847 { inputRawInline }
1848 { 2 }
1849 \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.

```

1850 \def\markdownRendererInputRawBlock{%
1851 \markdownRendererInputRawBlockPrototype}%
1852 \ExplSyntaxOn
1853 \seq_gput_right:Nn
1854 \g_@@_renderers_seq
1855 { inputRawBlock }
1856 \prop_gput:Nnn
1857 \g_@@_renderer_arities_prop
1858 { inputRawBlock }
1859 { 2 }
1860 \ExplSyntaxOff

```

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

```

1861 \def\markdownRendererSectionBegin{%
1862 \markdownRendererSectionBeginPrototype}%
1863 \ExplSyntaxOn
1864 \seq_gput_right:Nn

```

```

1865 \g_@@_renderers_seq
1866 { sectionBegin }
1867 \prop_gput:Nnn
1868 \g_@@_renderer_arities_prop
1869 { sectionBegin }
1870 { 0 }
1871 \ExplSyntaxOff
1872 \def\markdownRendererSectionEnd{%
1873 \markdownRendererSectionEndPrototype}%
1874 \ExplSyntaxOn
1875 \seq_gput_right:Nn
1876 \g_@@_renderers_seq
1877 { sectionEnd }
1878 \prop_gput:Nnn
1879 \g_@@_renderer_arities_prop
1880 { sectionEnd }
1881 { 0 }
1882 \ExplSyntaxOff

```

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

```

1883 \def\markdownRendererReplacementCharacter{%
1884 \markdownRendererReplacementCharacterPrototype}%
1885 \ExplSyntaxOn
1886 \seq_gput_right:Nn
1887 \g_@@_renderers_seq
1888 { replacementCharacter }
1889 \prop_gput:Nnn
1890 \g_@@_renderer_arities_prop
1891 { replacementCharacter }
1892 { 0 }
1893 \ExplSyntaxOff

```

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

```

1894 \def\markdownRendererLeftBrace{%
1895 \markdownRendererLeftBracePrototype}%
1896 \ExplSyntaxOn
1897 \seq_gput_right:Nn
1898 \g_@@_renderers_seq
1899 { leftBrace }
1900 \prop_gput:Nnn
1901 \g_@@_renderer_arities_prop

```

```

1902 { leftBrace }
1903 { 0 }
1904 \ExplSyntaxOff
1905 \def\markdownRendererRightBrace{%
1906   \markdownRendererRightBracePrototype}%
1907 \ExplSyntaxOn
1908 \seq_gput_right:Nn
1909   \g_@@_renderers_seq
1910   { rightBrace }
1911 \prop_gput:Nnn
1912   \g_@@_renderer_arities_prop
1913   { rightBrace }
1914   { 0 }
1915 \ExplSyntaxOff
1916 \def\markdownRendererDollarSign{%
1917   \markdownRendererDollarSignPrototype}%
1918 \ExplSyntaxOn
1919 \seq_gput_right:Nn
1920   \g_@@_renderers_seq
1921   { dollarSign }
1922 \prop_gput:Nnn
1923   \g_@@_renderer_arities_prop
1924   { dollarSign }
1925   { 0 }
1926 \ExplSyntaxOff
1927 \def\markdownRendererPercentSign{%
1928   \markdownRendererPercentSignPrototype}%
1929 \ExplSyntaxOn
1930 \seq_gput_right:Nn
1931   \g_@@_renderers_seq
1932   { percentSign }
1933 \prop_gput:Nnn
1934   \g_@@_renderer_arities_prop
1935   { percentSign }
1936   { 0 }
1937 \ExplSyntaxOff
1938 \def\markdownRendererAmpersand{%
1939   \markdownRendererAmpersandPrototype}%
1940 \ExplSyntaxOn
1941 \seq_gput_right:Nn
1942   \g_@@_renderers_seq
1943   { ampersand }
1944 \prop_gput:Nnn
1945   \g_@@_renderer_arities_prop
1946   { ampersand }
1947   { 0 }
1948 \ExplSyntaxOff

```

```

1949 \def\markdownRendererUnderscore{%
1950   \markdownRendererUnderscorePrototype}%
1951 \ExplSyntaxOn
1952 \seq_gput_right:Nn
1953   \g_@@_renderers_seq
1954   { underscore }
1955 \prop_gput:Nnn
1956   \g_@@_renderer_arities_prop
1957   { underscore }
1958   { 0 }
1959 \ExplSyntaxOff
1960 \def\markdownRendererHash{%
1961   \markdownRendererHashPrototype}%
1962 \ExplSyntaxOn
1963 \seq_gput_right:Nn
1964   \g_@@_renderers_seq
1965   { hash }
1966 \prop_gput:Nnn
1967   \g_@@_renderer_arities_prop
1968   { hash }
1969   { 0 }
1970 \ExplSyntaxOff
1971 \def\markdownRendererCircumflex{%
1972   \markdownRendererCircumflexPrototype}%
1973 \ExplSyntaxOn
1974 \seq_gput_right:Nn
1975   \g_@@_renderers_seq
1976   { circumflex }
1977 \prop_gput:Nnn
1978   \g_@@_renderer_arities_prop
1979   { circumflex }
1980   { 0 }
1981 \ExplSyntaxOff
1982 \def\markdownRendererBackslash{%
1983   \markdownRendererBackslashPrototype}%
1984 \ExplSyntaxOn
1985 \seq_gput_right:Nn
1986   \g_@@_renderers_seq
1987   { backslash }
1988 \prop_gput:Nnn
1989   \g_@@_renderer_arities_prop
1990   { backslash }
1991   { 0 }
1992 \ExplSyntaxOff
1993 \def\markdownRendererTilde{%
1994   \markdownRendererTildePrototype}%
1995 \ExplSyntaxOn

```

```

1996 \seq_gput_right:Nn
1997   \g_@@_renderers_seq
1998   { tilde }
1999 \prop_gput:Nnn
2000   \g_@@_renderer_arities_prop
2001   { tilde }
2002   { 0 }
2003 \ExplSyntaxOff
2004 \def\markdownRendererPipe{%
2005   \markdownRendererPipePrototype}%
2006 \ExplSyntaxOn
2007 \seq_gput_right:Nn
2008   \g_@@_renderers_seq
2009   { pipe }
2010 \prop_gput:Nnn
2011   \g_@@_renderer_arities_prop
2012   { pipe }
2013   { 0 }
2014 \ExplSyntaxOff

```

**2.2.3.34 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.

```

2015 \def\markdownRendererStrikeThrough{%
2016   \markdownRendererStrikeThroughPrototype}%
2017 \ExplSyntaxOn
2018 \seq_gput_right:Nn
2019   \g_@@_renderers_seq
2020   { strikeThrough }
2021 \prop_gput:Nnn
2022   \g_@@_renderer_arities_prop
2023   { strikeThrough }
2024   { 1 }
2025 \ExplSyntaxOff

```

**2.2.3.35 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.

```

2026 \def\markdownRendererSubscript{%
2027   \markdownRendererSubscriptPrototype}%
2028 \ExplSyntaxOn
2029 \seq_gput_right:Nn
2030   \g_@@_renderers_seq

```

```

2031 { subscript }
2032 \prop_gput:Nnn
2033 \g_@@_renderer_arities_prop
2034 { subscript }
2035 { 1 }

```

**2.2.3.36 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.

```

2036 \def\markdownRendererSuperscript{%
2037   \markdownRendererSuperscriptPrototype}%
2038 \ExplSyntaxOn
2039 \seq_gput_right:Nn
2040   \g_@@_renderers_seq
2041   { superscript }
2042 \prop_gput:Nnn
2043   \g_@@_renderer_arities_prop
2044   { superscript }
2045   { 1 }
2046 \ExplSyntaxOff

```

**2.2.3.37 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.

```

2047 \def\markdownRendererTable{%
2048   \markdownRendererTablePrototype}%
2049 \ExplSyntaxOn
2050 \seq_gput_right:Nn
2051   \g_@@_renderers_seq
2052   { table }
2053 \prop_gput:Nnn
2054   \g_@@_renderer_arities_prop
2055   { table }
2056   { 3 }
2057 \ExplSyntaxOff

```

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

```

2058 \def\markdownRendererInlineMath{%
2059   \markdownRendererInlineMathPrototype}%
2060 \ExplSyntaxOn
2061 \seq_gput_right:Nn
2062   \g_@@_renderers_seq
2063   { inlineMath }
2064 \prop_gput:Nnn
2065   \g_@@_renderer_arities_prop
2066   { inlineMath }
2067   { 1 }
2068 \ExplSyntaxOff
2069 \def\markdownRendererDisplayMath{%
2070   \markdownRendererDisplayMathPrototype}%
2071 \ExplSyntaxOn
2072 \seq_gput_right:Nn
2073   \g_@@_renderers_seq
2074   { displayMath }
2075 \prop_gput:Nnn
2076   \g_@@_renderer_arities_prop
2077   { displayMath }
2078   { 1 }
2079 \ExplSyntaxOff

```

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

```

2080 \def\markdownRendererTextCite{%
2081   \markdownRendererTextCitePrototype}%
2082 \ExplSyntaxOn
2083 \seq_gput_right:Nn
2084   \g_@@_renderers_seq
2085   { textCite }
2086 \prop_gput:Nnn
2087   \g_@@_renderer_arities_prop
2088   { textCite }
2089   { 1 }
2090 \ExplSyntaxOff

```



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

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

```

2091 \ExplSyntaxOn
2092 \cs_new:Npn
2093   \markdownRendererThematicBreak
2094   {
2095     \cs_if_exist:NTF
2096       \markdownRendererHorizontalRule
2097       {
2098         \markdownWarning
2099         {
2100           Horizontal~rule~renderer~has~been~deprecated,~
2101           to~be~removed~in~Markdown~3.0.0
2102         }
2103         \markdownRendererHorizontalRule
2104       }
2105       {
2106         \cs_if_exist:NTF
2107           \markdownRendererHorizontalRulePrototype
2108           {
2109             \markdownWarning
2110             {
2111               Horizontal~rule~renderer~prototype~has~been~deprecated,~
2112               to~be~removed~in~Markdown~3.0.0
2113             }
2114             \markdownRendererHorizontalRulePrototype
2115           }
2116           {
2117             \markdownRendererThematicBreakPrototype
2118           }
2119       }
2120   }
2121 \seq_gput_right:Nn
2122   \g_@@_renderers_seq
2123   { horizontalRule }
2124 \prop_gput:Nnn
2125   \g_@@_renderer_arities_prop
2126   { horizontalRule }
2127   { 0 }
2128 \seq_gput_right:Nn
2129   \g_@@_renderers_seq
2130   { thematicBreak }
2131 \prop_gput:Nnn
2132   \g_@@_renderer_arities_prop

```

```

2133 { thematicBreak }
2134 { 0 }
2135 \ExplSyntaxOff

```

**2.2.3.41 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.

```

2136 \def\markdownRendererTickedBox{%
2137   \markdownRendererTickedBoxPrototype}%
2138 \ExplSyntaxOn
2139 \seq_gput_right:Nn
2140   \g_@@_renderers_seq
2141   { tickedBox }
2142 \prop_gput:Nnn
2143   \g_@@_renderer_arities_prop
2144   { tickedBox }
2145   { 0 }
2146 \ExplSyntaxOff
2147 \def\markdownRendererHalfTickedBox{%
2148   \markdownRendererHalfTickedBoxPrototype}%
2149 \ExplSyntaxOn
2150 \seq_gput_right:Nn
2151   \g_@@_renderers_seq
2152   { halfTickedBox }
2153 \prop_gput:Nnn
2154   \g_@@_renderer_arities_prop
2155   { halfTickedBox }
2156   { 0 }
2157 \ExplSyntaxOff
2158 \def\markdownRendererUntickedBox{%
2159   \markdownRendererUntickedBoxPrototype}%
2160 \ExplSyntaxOn
2161 \seq_gput_right:Nn
2162   \g_@@_renderers_seq
2163   { untickedBox }
2164 \prop_gput:Nnn
2165   \g_@@_renderer_arities_prop
2166   { untickedBox }
2167   { 0 }
2168 \ExplSyntaxOff

```

**2.2.3.42 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.

```

2169 \def\markdownRendererJekyllDataBegin{%
2170   \markdownRendererJekyllDataBeginPrototype}%
2171 \ExplSyntaxOn
2172 \seq_gput_right:Nn
2173   \g_@@_renderers_seq
2174   { jekyllDataBegin }
2175 \prop_gput:Nnn
2176   \g_@@_renderer_arities_prop
2177   { jekyllDataBegin }
2178   { 0 }
2179 \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.

```

2180 \def\markdownRendererJekyllDataEnd{%
2181   \markdownRendererJekyllDataEndPrototype}%
2182 \ExplSyntaxOn
2183 \seq_gput_right:Nn
2184   \g_@@_renderers_seq
2185   { jekyllDataEnd }
2186 \prop_gput:Nnn
2187   \g_@@_renderer_arities_prop
2188   { jekyllDataEnd }
2189   { 0 }
2190 \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.

```

2191 \def\markdownRendererJekyllDataMappingBegin{%
2192   \markdownRendererJekyllDataMappingBeginPrototype}%
2193 \ExplSyntaxOn
2194 \seq_gput_right:Nn
2195   \g_@@_renderers_seq
2196   { jekyllDataMappingBegin }
2197 \prop_gput:Nnn
2198   \g_@@_renderer_arities_prop
2199   { jekyllDataMappingBegin }
2200   { 2 }
2201 \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.

```

2202 \def\markdownRendererJekyllDataMappingEnd{%
2203   \markdownRendererJekyllDataMappingEndPrototype}%
2204 \ExplSyntaxOn
2205 \seq_gput_right:Nn
2206   \g_@@_renderers_seq
2207   { jekyllDataMappingEnd }
2208 \prop_gput:Nnn
2209   \g_@@_renderer_arities_prop
2210   { jekyllDataMappingEnd }
2211   { 0 }
2212 \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.

```

2213 \def\markdownRendererJekyllDataSequenceBegin{%
2214   \markdownRendererJekyllDataSequenceBeginPrototype}%
2215 \ExplSyntaxOn
2216 \seq_gput_right:Nn
2217   \g_@@_renderers_seq
2218   { jekyllDataSequenceBegin }
2219 \prop_gput:Nnn
2220   \g_@@_renderer_arities_prop
2221   { jekyllDataSequenceBegin }
2222   { 2 }
2223 \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.

```

2224 \def\markdownRendererJekyllDataSequenceEnd{%
2225   \markdownRendererJekyllDataSequenceEndPrototype}%
2226 \ExplSyntaxOn
2227 \seq_gput_right:Nn
2228   \g_@@_renderers_seq
2229   { jekyllDataSequenceEnd }
2230 \prop_gput:Nnn
2231   \g_@@_renderer_arities_prop
2232   { jekyllDataSequenceEnd }
2233   { 0 }
2234 \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.

```

2235 \def\markdownRendererJekyllDataBoolean{%
2236   \markdownRendererJekyllDataBooleanPrototype}%
2237 \ExplSyntaxOn
2238 \seq_gput_right:Nn
2239   \g_@@_renderers_seq
2240   { jekyllDataBoolean }
2241 \prop_gput:Nnn
2242   \g_@@_renderer_arities_prop
2243   { jekyllDataBoolean }
2244   { 2 }
2245 \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.

```

2246 \def\markdownRendererJekyllDataNumber{%
2247   \markdownRendererJekyllDataNumberPrototype}%
2248 \ExplSyntaxOn
2249 \seq_gput_right:Nn
2250   \g_@@_renderers_seq
2251   { jekyllDataNumber }
2252 \prop_gput:Nnn
2253   \g_@@_renderer_arities_prop
2254   { jekyllDataNumber }
2255   { 2 }
2256 \ExplSyntaxOff

```

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

```

2257 \def\markdownRendererJekyllDataString{%
2258   \markdownRendererJekyllDataStringPrototype}%
2259 \ExplSyntaxOn
2260 \seq_gput_right:Nn
2261   \g_@@_renderers_seq
2262   { jekyllDataString }
2263 \prop_gput:Nnn

```

```

2264 \g_@@_renderer_arities_prop
2265 { jekyllDataString }
2266 { 2 }
2267 \ExplSyntaxOff

```

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

See also Section 2.2.4.1 for the description of the high-level expl3 interface that you can also use to react to YAML metadata.

```

2268 \def\markdownRendererJekyllDataEmpty{%
2269 \markdownRendererJekyllDataEmptyPrototype}%
2270 \ExplSyntaxOn
2271 \seq_gput_right:Nn
2272 \g_@@_renderers_seq
2273 { jekyllDataEmpty }
2274 \prop_gput:Nnn
2275 \g_@@_renderer_arities_prop
2276 { jekyllDataEmpty }
2277 { 1 }
2278 \ExplSyntaxOff

```

## 2.2.4 Token Renderer Prototypes

**2.2.4.1 YAML Metadata Renderer Prototypes** By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the  $\text{\LaTeX}$ 3 kernel.

```

2279 \ExplSyntaxOn
2280 \keys_define:nn
2281 { markdown/jekyllData }
2282 { }
2283 \ExplSyntaxOff

```

The following  $\text{\TeX}$  macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the  $\text{\LaTeX}$  and  $\text{\ConTeXt}$  implementations (see sections 3.3 and 3.4).

```

2284 \ExplSyntaxOn
2285 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
2286 {
2287 \seq_map_function:NN
2288 \g_@@_renderers_seq
2289 \@@_plaintex_define_renderer_prototype:n
2290 \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse

```

```

2291 \let\markdownRendererBlockHtmlCommentBegin=\iffalse
2292 \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
2293 \let\markdownRendererBlockHtmlCommentEnd=\fi

```

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

```

2294 \cs_undefine:N \markdownRendererFootnote
2295 \cs_undefine:N \markdownRendererFootnotePrototype

```

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

```

2296 \cs_undefine:N \markdownRendererHorizontalRule
2297 \cs_undefine:N \markdownRendererHorizontalRulePrototype
2298 }
2299 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
2300 {
2301   \@@_renderer_prototype_tl_to_csname:nN
2302   { #1 }
2303   \l_tmpa_tl
2304   \prop_get:NnN
2305   \g_@@_renderer_arities_prop
2306   { #1 }
2307   \l_tmpb_tl
2308   \@@_plaintex_define_renderer_prototype:cV
2309   { \l_tmpa_tl }
2310   \l_tmpb_tl
2311 }
2312 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
2313 {
2314   \tl_set:Nn
2315   \l_tmpa_tl
2316   { \str_uppercase:n { #1 } }
2317   \tl_set:Nx
2318   #2
2319   {
2320     markdownRenderer
2321     \tl_head:f { \l_tmpa_tl }
2322     \tl_tail:n { #1 }
2323     Prototype
2324   }
2325 }
2326 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
2327 {
2328   \cs_generate_from_arg_count:NNnn
2329   #1
2330   \cs_set:Npn
2331   { #2 }
2332   { }

```

```

2333 }
2334 \cs_generate_variant:Nn
2335 \@@_plaintex_define_renderer_prototype:Nn
2336 { cV }
2337 \@@_plaintex_define_renderer_prototypes:
2338 \ExplSyntaxOff

```

### 2.2.5 Logging Facilities

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

### 2.2.6 Miscellanea

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

```

2339 \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 T<sub>E</sub>X 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.

```

2340 \let\markdownReadAndConvert\relax
2341 \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*.

```

2342 \catcode`\|=0\catcode`\=12%
2343 |gdef|markdownBegin{%
2344   |markdownReadAndConvert{\markdownEnd}%
2345                               {|markdownEnd}}%
2346 |endgroup

```

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



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

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

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

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

```

2347 \ExplSyntaxOn
2348 \cs_if_exist:NF
2349   \markdownMode
2350   {
2351     \file_if_exist:nTF
2352       { lt3luabridge.tex }
2353       {
2354         \cs_new:Npn
2355           \markdownMode
2356           { 3 }
2357       }
2358     {
2359       \cs_if_exist:NTF
2360         \directlua
2361         {
2362           \cs_new:Npn
2363             \markdownMode
2364             { 2 }
2365         }
2366       {
2367         \cs_new:Npn
2368           \markdownMode
2369           { 0 }
2370       }
2371     }
2372   }
2373 \ExplSyntaxOff

```

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

```

2374 \def\markdownLuaRegisterIBCallback#1{\relax}%
2375 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

## 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).

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```
2376 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2377 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2378 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
2379 \input markdown/markdown
```

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

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

where *<options>* are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.6) and `markdownRendererPrototypes` (see Section 2.3.2.7) keys. Furthermore, although the base variant of the `import` key that loads a single L<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.2.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.3.2.4). This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
2380 \newenvironment{markdown}\relax\relax
2381 \newenvironment{markdown*}[1]\relax\relax
```

You may prepend your own code to the `\markdown` macro and append your own code to the `\endmarkdown` macro to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

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

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

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

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

```

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

```

### 2.3.2 Options

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

Except for the `plain` option described in Section 2.3.2.2, and the L<sup>A</sup>T<sub>E</sub>X themes described in Section 2.3.2.3, and the L<sup>A</sup>T<sub>E</sub>X snippets described in Section 2.3.2.1, L<sup>A</sup>T<sub>E</sub>X options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain T<sub>E</sub>X interface (see Sections 2.2.3 and 2.2.4).

The L<sup>A</sup>T<sub>E</sub>X options may be specified when loading the L<sup>A</sup>T<sub>E</sub>X package, when using the `markdown*` L<sup>A</sup>T<sub>E</sub>X environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```

2382 \ExplSyntaxOn
2383 \cs_new:Nn
2384   \@@_setup:n
2385   {
2386     \keys_set:nn
2387       { markdown/latex-options }
2388       { #1 }
2389   }
2390 \let\markdownSetup=\@@_setup:n
2391 \ExplSyntaxOff

```

**2.3.2.1 L<sup>A</sup>T<sub>E</sub>X snippets** We may also set up L<sup>A</sup>T<sub>E</sub>X 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:

```

2392 \ExplSyntaxOn
2393 \cs_new:Nn
2394   \@@_latex_setup_snippet:nn
2395   {
2396     \markdownIfSnippetExists
2397       { #1 }
2398       {
2399         \markdownWarning
2400           {Redefined~snippet~\markdownLaTeXThemeName#1}
2401         \csname markdownLaTeXSetupSnippet%
2402           \markdownLaTeXThemeName#1\endcsname={#2}
2403       }
2404       {
2405         \newtoks\next
2406         \next={#2}
2407         \expandafter\let\csname markdownLaTeXSetupSnippet%
2408           \markdownLaTeXThemeName#1\endcsname=\next
2409       }
2410   }
2411 \let\markdownSetupSnippet=\@@_latex_setup_snippet:nn
2412 \ExplSyntaxOff

```

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

```

2413 \newcommand\markdownIfSnippetExists[3]{%
2414   \ifundefined
2415     {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
2416     {#3}{#2}}%

```

See Section 2.3.2.3 for information on interactions between snippets and L<sup>A</sup>T<sub>E</sub>X themes. See Section 2.3.2.4 for information about invoking the set-up snippets.

To enable the enumeration of L<sup>A</sup>T<sub>E</sub>X options, we will maintain the `\g_@@_latex_options_seq` sequence.

```
2417 \ExplSyntaxOn
2418 \seq_new:N \g_@@_latex_options_seq
```

To enable the reflection of default L<sup>A</sup>T<sub>E</sub>X options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```
2419 \prop_new:N \g_@@_latex_option_types_prop
2420 \prop_new:N \g_@@_default_latex_options_prop
2421 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
2422 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
2423 \cs_new:Nn
2424   \@@_add_latex_option:nnn
2425   {
2426     \@@_add_option:Vnnn
2427     \c_@@_option_layer_latex_tl
2428     { #1 }
2429     { #2 }
2430     { #3 }
2431   }
```

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

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

```
2432 \@@_add_latex_option:nnn
2433   { plain }
2434   { boolean }
2435   { false }
2436 \ExplSyntaxOff
```

**2.3.2.3 L<sup>A</sup>T<sub>E</sub>X themes** User-defined L<sup>A</sup>T<sub>E</sub>X themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to L<sup>A</sup>T<sub>E</sub>X packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The  $\text{\LaTeX}$  option `import=<theme name>` loads a  $\text{\LaTeX}$  package (further referred to as a *theme*) named `markdowntheme<munged theme name>.sty`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_), the *theme name* is *qualified* and contains no underscores, and a value is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer  $\text{\LaTeX}$  package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single  $\text{\LaTeX}$  document class or for a single  $\text{\LaTeX}$  package. The preferred format of a theme name is `<theme author>/<target  $\text{\LaTeX}$  document class or package>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged, because  $\text{\LaTeX}$  packages are identified only by their filenames, not by their pathnames. [9] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a  $\text{\LaTeX}$  package named `markdownthemewitiko_beamer_MU.sty`.

If the  $\text{\LaTeX}$  option with key `theme` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown  $\text{\LaTeX}$  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`  $\text{\LaTeX}$  package, and finally the `markdownthemewitiko_dot.sty`  $\text{\LaTeX}$  package:

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

```
2437 \newif\ifmarkdownLaTeXLoaded
2438   \markdownLaTeXLoadedfalse
2439 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
2440 \ExplSyntaxOn
2441 \tl_new:N \markdownLaTeXThemePackageName
2442 \cs_new:Nn
2443   \@@_set_latex_theme:n
2444   {
2445     \str_if_in:nnF
2446       { #1 }
```

```

2447     { / }
2448     {
2449         \markdownError
2450         { Won't~load~theme~with~unqualified~name~#1 }
2451         { Theme~names~must~contain~at~least~one~forward~slash }
2452     }
2453     \str_if_in:nnT
2454     { #1 }
2455     { _ }
2456     {
2457         \markdownError
2458         { Won't~load~theme~with~an~underscore~in~its~name~#1 }
2459         { Theme~names~must~not~contain~underscores~in~their~names }
2460     }
2461     \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
2462     \str_replace_all:Nnn
2463     \markdownLaTeXThemePackageName
2464     { / }
2465     { _ }
2466     \edef\markdownLaTeXThemePackageName{
2467         markdowntheme\markdownLaTeXThemePackageName}
2468     \expandafter\markdownLaTeXThemeLoad\expandafter{
2469         \markdownLaTeXThemePackageName}{#1/}
2470 }
2471 \keys_define:nn
2472 { markdown/latex-options }
2473 {
2474     import .code:n = {
2475         \tl_set:Nn
2476         \l_tmpa_tl
2477         { #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.

```

2478     \tl_replace_all:NnV
2479     \l_tmpa_tl
2480     { / }
2481     \c_backslash_str
2482     \keys_set:nV
2483     { markdown/latex-options/import }
2484     \l_tmpa_tl
2485 },
2486 }
2487 \cs_generate_variant:Nn

```

```

2488 \tl_replace_all:Nnn
2489 { NnV }

```

The L<sup>A</sup>T<sub>E</sub>X option `theme` has been deprecated and will be removed in Markdown 3.0.0.

```

2490 \keys_define:nn
2491 { markdown/latex-options }
2492 {
2493   theme .code:n = { \@@_set_latex_theme:n { #1 } },
2494 }
2495 \ExplSyntaxOff

```

The L<sup>A</sup>T<sub>E</sub>X themes have a useful synergy with snippets (see Section 2.3.2.1): To make it less likely that different themes will set up snippets with the same name, we will prepend `<theme name>/` before the snippet name and use the result as the snippet name. For example, if the `witiko/dot` theme sets up the `product` snippet, the snippet will be available under the name `witiko/dot/product`. 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.

```

2496 \ExplSyntaxOn
2497 \@onlypreamble
2498   \@@_set_latex_theme:n
2499 \ExplSyntaxOff

```

Example themes provided with the Markdown package include:

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

```

\documentclass{article}
\usepackage[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;

```



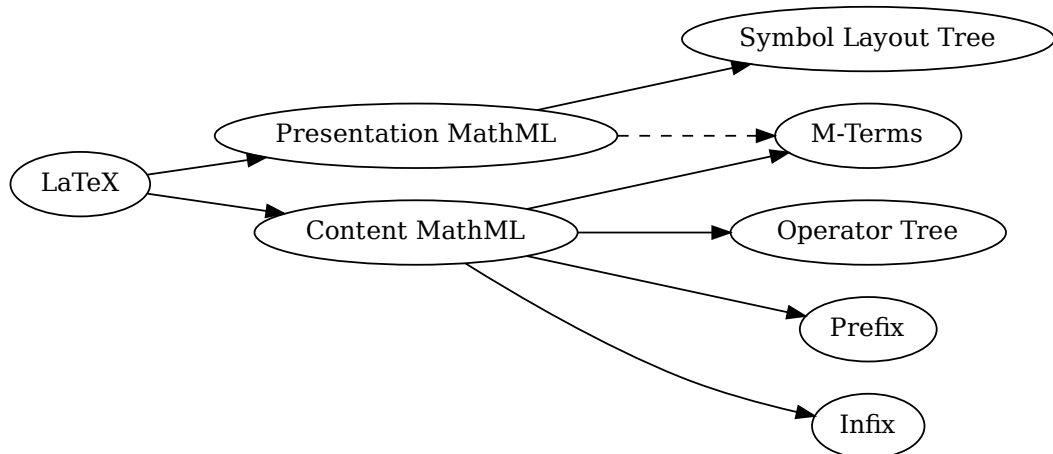
```

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.

2500 \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}


"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.

```
2501 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

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

```

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

```

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

```
2502 \ProvidesPackage{markdownthemewitiko_tilde}[2021/03/22]%
```

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

**2.3.2.4 L<sup>A</sup>T<sub>E</sub>X snippets** The L<sup>A</sup>T<sub>E</sub>X option with key `snippet` invokes a snippet named *<value>*:

```

2503 \ExplSyntaxOn
2504 \keys_define:nn
2505   { markdown/latex-options }
2506   {
2507     snippet .code:n = {
2508       \markdownIfSnippetExists{#1}
2509       {
2510         \expandafter\markdownSetup\expandafter{
2511           \the\csname markdownLaTeXSetupSnippet
2512             \markdownLaTeXThemeName#1\endcsname}
2513       }{
2514         \markdownError
2515           {Can't~invoke~setup~snippet~#1}
2516           {The~setup~snippet~is~undefined}
2517       }
2518     }
2519   }
2520 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```
\end{markdown}
```

```
\begin{markdown*}{snippet=romanNumerals}
```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown*}
```

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

```
\markdownSetup{import=jdoe/lists}
```

```
\begin{markdown*}{snippet=jdoe/lists/romanNumerals}
```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown*}
```

Alternatively, we can use the extended variant of the `import` L<sup>A</sup>T<sub>E</sub>X option that allows us to import the `romanNumerals` snippet to the current namespace, so that we can invoke the snippet with less typing:

```
\markdownSetup{
```

```
  import = {
```

```
    jdoe/lists = romanNumerals,
```

```
  },
```

```
}
```

```
\begin{markdown*}{snippet=romanNumerals}
```

The following ordered list will be preceded by roman numerals:

```

3. tres
4. quattuor

\end{markdown*}

```

Furthermore, we can also specify the name of the snippet in the current namespace, which is different from the name of the snippet in the `jdoe/lists` theme:

```

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

```

The following ordered list will be preceded by roman numerals:

```

3. tres
4. quattuor

\end{markdown*}

```

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

```

2521 \ExplSyntaxOn
2522 \tl_new:N
2523   \l_@@_latex_import_current_theme_tl
2524 \keys_define:nn
2525   { markdown/latex-options/import }
2526   {

```

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

```

2527     unknown .default:n = {},
2528     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.

```

2529     \tl_set_eq:NN

```

```

2530     \l_@@_latex_import_current_theme_tl
2531     \l_keys_key_str
2532     \tl_replace_all:NVn
2533     \l_@@_latex_import_current_theme_tl
2534     \c_backslash_str
2535     { / }

```

Here, we load the L<sup>A</sup>T<sub>E</sub>X theme.

```

2536     \@@_set_latex_theme:V
2537     \l_@@_latex_import_current_theme_tl

```

Here, we import the L<sup>A</sup>T<sub>E</sub>X snippets.

```

2538     \clist_map_inline:nn
2539     { #1 }
2540     {
2541         \regex_extract_once:nnNTF
2542         { ^(.*)\s+as\s+(.*)$ }
2543         { ##1 }
2544         \l_tmpa_seq
2545         {
2546             \seq_pop:NN
2547             \l_tmpa_seq
2548             \l_tmpa_tl
2549             \seq_pop:NN
2550             \l_tmpa_seq
2551             \l_tmpa_tl
2552             \seq_pop:NN
2553             \l_tmpa_seq
2554             \l_tmpb_tl
2555         }
2556         {
2557             \tl_set:Nn
2558             \l_tmpa_tl
2559             { ##1 }
2560             \tl_set:Nn
2561             \l_tmpb_tl
2562             { ##1 }
2563         }
2564         \tl_put_left:Nn
2565         \l_tmpa_tl
2566         { / }
2567         \tl_put_left:NV
2568         \l_tmpa_tl
2569         \l_@@_latex_import_current_theme_tl
2570         \@@_latex_setup_snippet:Vx
2571         \l_tmpb_tl
2572         { snippet = { \l_tmpa_tl } }
2573     }

```

```

2574     },
2575   }
2576   \cs_generate_variant:Nn
2577     \tl_replace_all:Nnn
2578     { NVn }
2579   \cs_generate_variant:Nn
2580     \@@_set_latex_theme:n
2581     { V }
2582   \cs_generate_variant:Nn
2583     \@@_latex_setup_snippet:nn
2584     { Vx }
2585   \ExplSyntaxOff

```

**2.3.2.5 Plain T<sub>E</sub>X Interface Options** Here, we automatically define plain T<sub>E</sub>X macros and the  $\langle key \rangle = \langle value \rangle$  interface for the above L<sup>A</sup>T<sub>E</sub>X options.

```

2586 \ExplSyntaxOn
2587 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
2588 {
2589   \seq_map_inline:Nn
2590     \g_@@_latex_options_seq
2591     {
2592       \@@_plain_tex_define_option_command:n
2593       { ##1 }
2594     }

```

Furthermore, we also define the  $\langle key \rangle = \langle value \rangle$  interface for all option macros recognized by the Lua and plain T<sub>E</sub>X interfaces.

```

2595   \seq_map_inline:Nn
2596     \g_@@_option_layers_seq
2597     {
2598       \seq_map_inline:cn
2599       { g_@@_ ##1 _options_seq }
2600     }

```

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.

```

2601   \@@_with_various_cases:nn
2602   { #####1 }
2603   {
2604     \@@_latex_define_option_keyval:nnn
2605     { ##1 }
2606     { #####1 }
2607     { #####1 }
2608   }
2609 }

```

```

2610     }
2611   }
2612   \cs_new:Nn \@@_latex_define_option_keyval:nnn
2613   {
2614     \prop_get:cnN
2615       { g_@@_ #1 _option_types_prop }
2616       { #2 }
2617     \l_tmpa_tl
2618     \keys_define:nn
2619       { markdown/latex-options }
2620       {
2621         #3 .code:n = {
2622           \@@_set_option_value:nn
2623             { #2 }
2624             { ##1 }
2625         },
2626       }
2627     \str_if_eq:VVT
2628       \l_tmpa_tl
2629       \c_@@_option_type_boolean_tl
2630       {
2631         \keys_define:nn
2632           { markdown/latex-options }
2633           {
2634             #3 .default:n = { true },
2635           }
2636       }

```

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.

```

2637   \str_if_eq:VVT
2638     \l_tmpa_tl
2639     \c_@@_option_type_clist_tl
2640     {
2641       \tl_set:Nn
2642         \l_tmpa_tl
2643         { #3 }
2644       \tl_reverse:N
2645         \l_tmpa_tl
2646       \str_if_eq:enF
2647         {
2648           \tl_head:V
2649             \l_tmpa_tl
2650         }

```



```

2651         { s }
2652     {
2653         \msg_error:nnn
2654         { @@ }
2655         { malformed-name-for-clist-option }
2656         { #3 }
2657     }
2658     \tl_set:Nx
2659     \l_tmpa_tl
2660     {
2661         \tl_tail:V
2662         \l_tmpa_tl
2663     }
2664     \tl_reverse:N
2665     \l_tmpa_tl
2666     \tl_put_right:Nn
2667     \l_tmpa_tl
2668     {
2669         .code:n = {
2670             \@@_get_option_value:nN
2671             { #2 }
2672             \l_tmpa_tl
2673             \clist_set:NV
2674             \l_tmpa_clist
2675             { \l_tmpa_tl, { ##1 } }
2676             \@@_set_option_value:nV
2677             { #2 }
2678             \l_tmpa_clist
2679         }
2680     }
2681     \keys_define:nV
2682     { markdown/latex-options }
2683     \l_tmpa_tl
2684 }
2685 }
2686 \cs_generate_variant:Nn
2687 \clist_set:Nn
2688 { NV }
2689 \cs_generate_variant:Nn
2690 \keys_define:nn
2691 { nV }
2692 \cs_generate_variant:Nn
2693 \@@_set_option_value:nn
2694 { nV }
2695 \prg_generate_conditional_variant:Nnn
2696 \str_if_eq:nn
2697 { en }

```

```

2698 { F }
2699 \msg_new:nnn
2700 { @@ }
2701 { malformed-name-for-clist-option }
2702 {
2703     Clist-option-name~#1~does~not~end~with~-s.
2704 }
2705 \@@_latex_define_option_commands_and_keyvals:
2706 \ExplSyntaxOff

```

The `finalizeCache` and `frozenCache` plain TeX options are exposed through LaTeX options with keys `finalizeCache` and `frozenCache`.

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

```

2707 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
2708 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}

```

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

```

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

```

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

```

2709 \ExplSyntaxOn

```

```

2710 \cs_new:Nn \@@_latex_define_renderers:
2711 {
2712     \seq_map_function:NN
2713         \g_@@_renderers_seq
2714         \@@_latex_define_renderer:n
2715 }
2716 \cs_new:Nn \@@_latex_define_renderer:n
2717 {
2718     \@@_renderer_tl_to_csname:nN
2719     { #1 }
2720     \l_tmpa_tl
2721     \prop_get:NnN
2722         \g_@@_renderer_arities_prop
2723         { #1 }
2724     \l_tmpb_tl
2725     \@@_latex_define_renderer:ncV
2726     { #1 }
2727     { \l_tmpa_tl }
2728     \l_tmpb_tl
2729 }
2730 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2731 {
2732     \tl_set:Nn
2733         \l_tmpa_tl
2734         { \str_uppercase:n { #1 } }
2735     \tl_set:Nx
2736         #2
2737         {
2738             markdownRenderer
2739             \tl_head:f { \l_tmpa_tl }
2740             \tl_tail:n { #1 }
2741         }
2742 }
2743 \cs_new:Nn \@@_latex_define_renderer:nNn
2744 {
2745     \@@_with_various_cases:nn
2746     { #1 }
2747     {
2748         \keys_define:nn
2749             { markdown/latex-options/renderers }
2750             {
2751                 ##1 .code:n = {
2752                     \cs_generate_from_arg_count:NNnn
2753                     #2
2754                     \cs_set:Npn
2755                         { #3 }
2756                     { ####1 }

```

```

2757         },
2758     }
2759 }
2760 }
2761 \cs_generate_variant:Nn
2762   \@@_latex_define_renderer:nNn
2763   { ncV }
2764 \ExplSyntaxOff

```

The following example  $\text{\LaTeX}$  code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` markdown token renderers.

```

\markdownSetup{
  renderers = {
    link = {#4}, % Render links as the link title.
    emphasis = {\emph{#1}}, % Render emphasized text via \emph.
  }
}

```

**2.3.2.7 Plain  $\text{\TeX}$  Markdown Token Renderer Prototypes** The  $\text{\LaTeX}$  interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain  $\text{\TeX}$  interface (see Section 2.2.4).

```

2765 \ExplSyntaxOn
2766 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2767   {
2768     \seq_map_function:NN
2769       \g_@@_renderers_seq
2770       \@@_latex_define_renderer_prototype:n
2771   }
2772 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2773   {
2774     \@@_renderer_prototype_tl_to_csname:nN
2775       { #1 }
2776       \l_tmpa_tl
2777     \prop_get:NnN
2778       \g_@@_renderer_arities_prop
2779       { #1 }
2780       \l_tmpb_tl
2781     \@@_latex_define_renderer_prototype:ncV
2782       { #1 }
2783       { \l_tmpa_tl }
2784       \l_tmpb_tl
2785   }

```

```

2786 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2787 {
2788   \@@_with_various_cases:nn
2789   { #1 }
2790   {
2791     \keys_define:nn
2792     { markdown/latex-options/renderer-prototypes }
2793     {
2794       ##1 .code:n = {
2795         \cs_generate_from_arg_count:NNnn
2796         #2
2797         \cs_set:Npn
2798         { #3 }
2799         { #####1 }
2800       },
2801     }
2802   }
2803 }
2804 \cs_generate_variant:Nn
2805 \@@_latex_define_renderer_prototype:nNn
2806 { ncV }
2807 \ExplSyntaxOff

```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```

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

```

## 2.4 ConT<sub>E</sub>Xt Interface

The ConT<sub>E</sub>Xt interface provides a start-stop macro pair for the typesetting of markdown input from within ConT<sub>E</sub>Xt and facilities for setting Lua, plain T<sub>E</sub>X, and ConT<sub>E</sub>Xt options used during the conversion from markdown to plain T<sub>E</sub>X. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

```

2808 \writestatus{loading}{ConTEXt User Module / markdown}%
2809 \startmodule[markdown]
2810 \unprotect

```

The ConT<sub>E</sub>Xt implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the ConT<sub>E</sub>Xt `\writestatus` macro.

```

2811 \def\markdownInfo#1{\writestatus{markdown}{#1.}}}%
2812 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}}%
2813 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2814   \do\#\do\^\do\_do\%\do\~}%
2815 \input markdown/markdown

```

The ConT<sub>E</sub>Xt interface is implemented by the `t-markdown.tex` ConT<sub>E</sub>Xt module file that can be loaded as follows:

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

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

### 2.4.1 Typesetting Markdown

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

```

2816 \let\startmarkdown\relax
2817 \let\stopmarkdown\relax
2818 \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 T<sub>E</sub>X interface.

The following example ConT<sub>E</sub>Xt 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 T<sub>E</sub>X. Unlike the `\markdownInput` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts ConT<sub>E</sub>Xt interface

options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

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

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

## 2.4.2 Options

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

ConT<sub>E</sub>Xt options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2).

The ConT<sub>E</sub>Xt options may be specified when using the `\inputmarkdown` macro (see Section 2.4), or via the `\setupmarkdown` macro. The `\setupmarkdown` macro receives the options to set up as its only argument:

```
2819 \ExplSyntaxOn
2820 \cs_new:Nn
2821   \@@_setup:n
2822   {
2823     \keys_set:nn
2824       { markdown/context-options }
2825       { #1 }
2826   }
2827 \long\def\setupmarkdown[#1]
2828   {
2829     \@@_setup:n
2830       { #1 }
2831   }
2832 \ExplSyntaxOff
```

**2.4.2.1 ConT<sub>E</sub>Xt Interface Options** We define the  $\langle key \rangle = \langle value \rangle$  interface for all option macros recognized by the Lua and plain T<sub>E</sub>X interfaces.

```
2833 \ExplSyntaxOn
2834 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2835   {
2836     \seq_map_inline:Nn
2837       \g_@@_option_layers_seq
2838       {
2839         \seq_map_inline:cn
```

```

2840         { g_@@_ ##1 _options_seq }
2841     {

```

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`.

```

2842         \@@_with_various_cases:nn
2843         { #####1 }
2844     {
2845         \@@_context_define_option_keyval:nnn
2846         { ##1 }
2847         { #####1 }
2848         { #####1 }
2849     }
2850 }
2851 }
2852 }

```

Furthermore, we also accept caseless variants of options in line with the style of ConT<sub>E</sub>Xt.

```

2853 \cs_new:Nn \@@_caseless:N
2854 {
2855     \regex_replace_all:nnN
2856     { ([a-z])([A-Z]) }
2857     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
2858     #1
2859     \tl_set:Nx
2860     #1
2861     { #1 }
2862 }
2863 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
2864 \cs_new:Nn \@@_context_define_option_keyval:nnn
2865 {
2866     \prop_get:cnN
2867     { g_@@_ #1 _option_types_prop }
2868     { #2 }
2869     \l_tmpa_tl
2870     \keys_define:nn
2871     { markdown/context-options }
2872     {
2873         #3 .code:n = {
2874             \tl_set:Nx
2875             \l_tmpa_tl
2876             {
2877                 \str_case:nnF
2878                 { ##1 }

```



```

2879         {
2880             { yes } { true }
2881             { no } { false }
2882         }
2883         { ##1 }
2884     }
2885     \@@_set_option_value:nV
2886     { #2 }
2887     \l_tmpa_tl
2888 },
2889 }
2890 \str_if_eq:VVT
2891 \l_tmpa_tl
2892 \c_@@_option_type_boolean_tl
2893 {
2894     \keys_define:nn
2895     { markdown/context-options }
2896     {
2897         #3 .default:n = { true },
2898     }
2899 }
2900 }
2901 \cs_generate_variant:Nn
2902 \@@_set_option_value:nn
2903 { nV }
2904 \@@_context_define_option_commands_and_keyvals:
2905 \ExplSyntaxOff

```

## 3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{\TeX}$  *token renderers* is performed by the Lua layer. The plain  $\text{\TeX}$  layer provides default definitions for the token renderers. The  $\text{\LaTeX}$  and  $\text{ConTeXt}$  layers correct idiosyncrasies of the respective  $\text{\TeX}$  formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

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

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

```
2906 local upper, format, length =
2907     string.upper, string.format, string.len
2908 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2909     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2910     lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.P(1)
```

### 3.1.1 Utility Functions

This section documents the utility functions used by the plain T<sub>E</sub>X writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
2911 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.

```
2912 function util.err(msg, exit_code)
2913     io.stderr:write("markdown.lua: " .. msg .. "\n")
2914     os.exit(exit_code or 1)
2915 end
```

The `util.cache` method computes the digest of `string` and `salt`, adds the `suffix` and looks into the directory `dir`, whether a file with such a name exists. If it does not, it gets created with `transform(string)` as its content. The filename is then returned.

```
2916 function util.cache(dir, string, salt, transform, suffix)
2917     local digest = md5.sumhexa(string .. (salt or ""))
2918     local name = util.pathname(dir, digest .. suffix)
2919     local file = io.open(name, "r")
2920     if file == nil then -- If no cache entry exists, then create a new one.
2921         file = assert(io.open(name, "w"),
2922             [[Could not open file ]] .. name .. [[ for writing]])
2923         local result = string
2924         if transform ~= nil then
2925             result = transform(result)
2926         end
2927         assert(file:write(result))
2928         assert(file:close())
2929     end
2930     return name
2931 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.

```
2932 function util.cache_verbatim(dir, string)
2933   local name = util.cache(dir, string, nil, nil, ".verbatim")
2934   return name
2935 end
```

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

```
2936 function util.table_copy(t)
2937   local u = { }
2938   for k, v in pairs(t) do u[k] = v end
2939   return setmetatable(u, getmetatable(t))
2940 end
```

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

```
2941 function util.encode_json_string(s)
2942   s = s:gsub([[\\]], [[\\]])
2943   s = s:gsub([[\"]], [[\"]])
2944   return [[\"]] .. s .. [[\"]]
2945 end
```

The `util.lookup_files` method looks up files with filename `f` and returns their paths. Further options for the Kpathsea library can be specified in table `options`. [1, Section 10.7.4]

```
2946 function util.lookup_files(f, options)
2947   return kpse.lookup(f, options)
2948 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].

```
2949 function util.expand_tabs_in_line(s, tabstop)
2950   local tab = tabstop or 4
2951   local corr = 0
2952   return (s:gsub(")\t", function(p)
2953     local sp = tab - (p - 1 + corr) % tab
2954     corr = corr - 1 + sp
2955     return string.rep(" ", sp)
2956   end))
2957 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.

```

2958 function util.walk(t, f)
2959   local typ = type(t)
2960   if typ == "string" then
2961     f(t)
2962   elseif typ == "table" then
2963     local i = 1
2964     local n
2965     n = t[i]
2966     while n do
2967       util.walk(n, f)
2968       i = i + 1
2969       n = t[i]
2970     end
2971   elseif typ == "function" then
2972     local ok, val = pcall(t)
2973     if ok then
2974       util.walk(val, f)
2975     end
2976   else
2977     f(tostring(t))
2978   end
2979 end

```

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

```

2980 function util.flatten(ary)
2981   local new = {}
2982   for _,v in ipairs(ary) do
2983     if type(v) == "table" then
2984       for _,w in ipairs(util.flatten(v)) do
2985         new[#new + 1] = w
2986       end
2987     else
2988       new[#new + 1] = v
2989     end
2990   end
2991   return new
2992 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.

```

2993 function util.rope_to_string(rope)
2994   local buffer = {}
2995   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2996   return table.concat(buffer)
2997 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.

```
2998 function util.rope_last(rope)
2999   if #rope == 0 then
3000     return nil
3001   else
3002     local l = rope[#rope]
3003     if type(l) == "table" then
3004       return util.rope_last(l)
3005     else
3006       return l
3007     end
3008   end
3009 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  $1 \leq i \leq \#ary$ .

```
3010 function util.intersperse(ary, x)
3011   local new = {}
3012   local l = #ary
3013   for i,v in ipairs(ary) do
3014     local n = #new
3015     new[n + 1] = v
3016     if i ~= l then
3017       new[n + 2] = x
3018     end
3019   end
3020   return new
3021 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  $1 \leq i \leq \#ary$ .

```
3022 function util.map(ary, f)
3023   local new = {}
3024   for i,v in ipairs(ary) do
3025     new[i] = f(v)
3026   end
3027   return new
3028 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.

```
3029 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3030 local char_escapes_list = ""
3031 for i,_ in pairs(char_escapes) do
3032     char_escapes_list = char_escapes_list .. i
3033 end
```

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

```
3034 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.

```
3035 if string_escapes then
3036     for k,v in pairs(string_escapes) do
3037         escapable = P(k) / v + escapable
3038     end
3039 end
```

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

```
3040 local escape_string = Cs((escapable + any)^0)
```

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

```
3041 return function(s)
3042     return lpeg.match(escape_string, s)
3043 end
3044 end
```

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

```
3045 function util.pathname(dir, file)
3046     if #dir == 0 then
3047         return file
3048     else
3049         return dir .. "/" .. file
3050     end
3051 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.

```
3052 local entities = {}
3053
3054 local character_entities = {
3055     ["Tab"] = 9,
3056     ["NewLine"] = 10,
3057     ["excl"] = 33,
3058     ["quot"] = 34,
3059     ["QUOT"] = 34,
3060     ["num"] = 35,
3061     ["dollar"] = 36,
3062     ["percnt"] = 37,
3063     ["amp"] = 38,
3064     ["AMP"] = 38,
3065     ["apos"] = 39,
3066     ["lpar"] = 40,
3067     ["rpar"] = 41,
3068     ["ast"] = 42,
3069     ["midast"] = 42,
3070     ["plus"] = 43,
3071     ["comma"] = 44,
3072     ["period"] = 46,
3073     ["sol"] = 47,
3074     ["colon"] = 58,
3075     ["semi"] = 59,
3076     ["lt"] = 60,
3077     ["LT"] = 60,
3078     ["equals"] = 61,
3079     ["gt"] = 62,
3080     ["GT"] = 62,
3081     ["quest"] = 63,
3082     ["commat"] = 64,
3083     ["lsqb"] = 91,
3084     ["lbrack"] = 91,
3085     ["bsol"] = 92,
3086     ["rsqb"] = 93,
3087     ["rbrack"] = 93,
3088     ["Hat"] = 94,
3089     ["lowbar"] = 95,
3090     ["grave"] = 96,
3091     ["DiacriticalGrave"] = 96,
3092     ["lcub"] = 123,
3093     ["lbrace"] = 123,
```

```

3094 ["verbar"] = 124,
3095 ["vert"] = 124,
3096 ["VerticalLine"] = 124,
3097 ["rcub"] = 125,
3098 ["rbrace"] = 125,
3099 ["nbsp"] = 160,
3100 ["NonBreakingSpace"] = 160,
3101 ["iexcl"] = 161,
3102 ["cent"] = 162,
3103 ["pound"] = 163,
3104 ["curren"] = 164,
3105 ["yen"] = 165,
3106 ["brvbar"] = 166,
3107 ["sect"] = 167,
3108 ["Dot"] = 168,
3109 ["die"] = 168,
3110 ["DoubleDot"] = 168,
3111 ["uml"] = 168,
3112 ["copy"] = 169,
3113 ["COPY"] = 169,
3114 ["ordf"] = 170,
3115 ["laquo"] = 171,
3116 ["not"] = 172,
3117 ["shy"] = 173,
3118 ["reg"] = 174,
3119 ["circledR"] = 174,
3120 ["REG"] = 174,
3121 ["macr"] = 175,
3122 ["OverBar"] = 175,
3123 ["strns"] = 175,
3124 ["deg"] = 176,
3125 ["plusmn"] = 177,
3126 ["pm"] = 177,
3127 ["PlusMinus"] = 177,
3128 ["sup2"] = 178,
3129 ["sup3"] = 179,
3130 ["acute"] = 180,
3131 ["DiacriticalAcute"] = 180,
3132 ["micro"] = 181,
3133 ["para"] = 182,
3134 ["middot"] = 183,
3135 ["centerdot"] = 183,
3136 ["CenterDot"] = 183,
3137 ["cedil"] = 184,
3138 ["Cedilla"] = 184,
3139 ["sup1"] = 185,
3140 ["ordm"] = 186,

```



3141 ["raquo"] = 187,  
 3142 ["frac14"] = 188,  
 3143 ["frac12"] = 189,  
 3144 ["half"] = 189,  
 3145 ["frac34"] = 190,  
 3146 ["iquest"] = 191,  
 3147 ["Agrave"] = 192,  
 3148 ["Aacute"] = 193,  
 3149 ["Acirc"] = 194,  
 3150 ["Atilde"] = 195,  
 3151 ["Auml"] = 196,  
 3152 ["Aring"] = 197,  
 3153 ["AElig"] = 198,  
 3154 ["Ccedil"] = 199,  
 3155 ["Egrave"] = 200,  
 3156 ["Eacute"] = 201,  
 3157 ["Ecirc"] = 202,  
 3158 ["Euml"] = 203,  
 3159 ["Igrave"] = 204,  
 3160 ["Iacute"] = 205,  
 3161 ["Icirc"] = 206,  
 3162 ["Iuml"] = 207,  
 3163 ["ETH"] = 208,  
 3164 ["Ntilde"] = 209,  
 3165 ["Ograve"] = 210,  
 3166 ["Oacute"] = 211,  
 3167 ["Ocirc"] = 212,  
 3168 ["Otilde"] = 213,  
 3169 ["Ouml"] = 214,  
 3170 ["times"] = 215,  
 3171 ["Oslash"] = 216,  
 3172 ["Ugrave"] = 217,  
 3173 ["Uacute"] = 218,  
 3174 ["Ucirc"] = 219,  
 3175 ["Uuml"] = 220,  
 3176 ["Yacute"] = 221,  
 3177 ["THORN"] = 222,  
 3178 ["szlig"] = 223,  
 3179 ["agrave"] = 224,  
 3180 ["aacute"] = 225,  
 3181 ["acirc"] = 226,  
 3182 ["atilde"] = 227,  
 3183 ["auml"] = 228,  
 3184 ["aring"] = 229,  
 3185 ["aelig"] = 230,  
 3186 ["ccedil"] = 231,  
 3187 ["egrave"] = 232,

3188 ["eacute"] = 233,  
 3189 ["ecirc"] = 234,  
 3190 ["euml"] = 235,  
 3191 ["igrave"] = 236,  
 3192 ["iacute"] = 237,  
 3193 ["icirc"] = 238,  
 3194 ["iuml"] = 239,  
 3195 ["eth"] = 240,  
 3196 ["ntilde"] = 241,  
 3197 ["ograve"] = 242,  
 3198 ["oacute"] = 243,  
 3199 ["ocirc"] = 244,  
 3200 ["otilde"] = 245,  
 3201 ["ouml"] = 246,  
 3202 ["divide"] = 247,  
 3203 ["div"] = 247,  
 3204 ["oslash"] = 248,  
 3205 ["ugrave"] = 249,  
 3206 ["uacute"] = 250,  
 3207 ["ucirc"] = 251,  
 3208 ["uuml"] = 252,  
 3209 ["yacute"] = 253,  
 3210 ["thorn"] = 254,  
 3211 ["yuml"] = 255,  
 3212 ["Amacr"] = 256,  
 3213 ["amacr"] = 257,  
 3214 ["Abreve"] = 258,  
 3215 ["abreve"] = 259,  
 3216 ["Aogon"] = 260,  
 3217 ["aogon"] = 261,  
 3218 ["Cacute"] = 262,  
 3219 ["cacute"] = 263,  
 3220 ["Ccirc"] = 264,  
 3221 ["ccirc"] = 265,  
 3222 ["Cdot"] = 266,  
 3223 ["cdot"] = 267,  
 3224 ["Ccaron"] = 268,  
 3225 ["ccaron"] = 269,  
 3226 ["Dcaron"] = 270,  
 3227 ["dcaron"] = 271,  
 3228 ["Dstrok"] = 272,  
 3229 ["dstrok"] = 273,  
 3230 ["Emacr"] = 274,  
 3231 ["emacr"] = 275,  
 3232 ["Edot"] = 278,  
 3233 ["edot"] = 279,  
 3234 ["Eogon"] = 280,

```

3235 ["eogon"] = 281,
3236 ["Ecaron"] = 282,
3237 ["ecaron"] = 283,
3238 ["Gcirc"] = 284,
3239 ["gcirc"] = 285,
3240 ["Gbreve"] = 286,
3241 ["gbreve"] = 287,
3242 ["Gdot"] = 288,
3243 ["gdot"] = 289,
3244 ["Gcedil"] = 290,
3245 ["Hcirc"] = 292,
3246 ["hcirc"] = 293,
3247 ["Hstrook"] = 294,
3248 ["hstrook"] = 295,
3249 ["Itilde"] = 296,
3250 ["itilde"] = 297,
3251 ["Imacr"] = 298,
3252 ["imacr"] = 299,
3253 ["Iogon"] = 302,
3254 ["iogon"] = 303,
3255 ["Idot"] = 304,
3256 ["imath"] = 305,
3257 ["inodot"] = 305,
3258 ["IJlig"] = 306,
3259 ["ijlig"] = 307,
3260 ["Jcirc"] = 308,
3261 ["jcirc"] = 309,
3262 ["Kcedil"] = 310,
3263 ["kcedil"] = 311,
3264 ["kgreen"] = 312,
3265 ["Lacute"] = 313,
3266 ["lacute"] = 314,
3267 ["Lcedil"] = 315,
3268 ["lcedil"] = 316,
3269 ["Lcaron"] = 317,
3270 ["lcaron"] = 318,
3271 ["Lmidot"] = 319,
3272 ["lmidot"] = 320,
3273 ["Lstrook"] = 321,
3274 ["lstrook"] = 322,
3275 ["Nacute"] = 323,
3276 ["nacute"] = 324,
3277 ["Ncedil"] = 325,
3278 ["ncedil"] = 326,
3279 ["Ncaron"] = 327,
3280 ["ncaron"] = 328,
3281 ["napos"] = 329,

```

```

3282 ["ENG"] = 330,
3283 ["eng"] = 331,
3284 ["Omacr"] = 332,
3285 ["omacr"] = 333,
3286 ["Odblac"] = 336,
3287 ["odblac"] = 337,
3288 ["OElig"] = 338,
3289 ["oelig"] = 339,
3290 ["Racute"] = 340,
3291 ["racute"] = 341,
3292 ["Rcedil"] = 342,
3293 ["rcedil"] = 343,
3294 ["Rcaron"] = 344,
3295 ["rcaron"] = 345,
3296 ["Sacute"] = 346,
3297 ["sacute"] = 347,
3298 ["Scirc"] = 348,
3299 ["scirc"] = 349,
3300 ["Scedil"] = 350,
3301 ["scedil"] = 351,
3302 ["Scaron"] = 352,
3303 ["scaron"] = 353,
3304 ["Tcedil"] = 354,
3305 ["tcedil"] = 355,
3306 ["Tcaron"] = 356,
3307 ["tcaron"] = 357,
3308 ["Tstrok"] = 358,
3309 ["tstrok"] = 359,
3310 ["Utilde"] = 360,
3311 ["utilde"] = 361,
3312 ["Umacr"] = 362,
3313 ["umacr"] = 363,
3314 ["Ubreve"] = 364,
3315 ["ubreve"] = 365,
3316 ["Uring"] = 366,
3317 ["uring"] = 367,
3318 ["Udblac"] = 368,
3319 ["udblac"] = 369,
3320 ["Uogon"] = 370,
3321 ["uogon"] = 371,
3322 ["Wcirc"] = 372,
3323 ["wcirc"] = 373,
3324 ["Ycirc"] = 374,
3325 ["ycirc"] = 375,
3326 ["Yuml"] = 376,
3327 ["Zacute"] = 377,
3328 ["zacute"] = 378,

```

3329 ["Zdot"] = 379,  
 3330 ["zdot"] = 380,  
 3331 ["Zcaron"] = 381,  
 3332 ["zcaron"] = 382,  
 3333 ["fnof"] = 402,  
 3334 ["imped"] = 437,  
 3335 ["gacute"] = 501,  
 3336 ["jmath"] = 567,  
 3337 ["circ"] = 710,  
 3338 ["caron"] = 711,  
 3339 ["Hacek"] = 711,  
 3340 ["breve"] = 728,  
 3341 ["Breve"] = 728,  
 3342 ["dot"] = 729,  
 3343 ["DiacriticalDot"] = 729,  
 3344 ["ring"] = 730,  
 3345 ["ogon"] = 731,  
 3346 ["tilde"] = 732,  
 3347 ["DiacriticalTilde"] = 732,  
 3348 ["dblac"] = 733,  
 3349 ["DiacriticalDoubleAcute"] = 733,  
 3350 ["DownBreve"] = 785,  
 3351 ["UnderBar"] = 818,  
 3352 ["Alpha"] = 913,  
 3353 ["Beta"] = 914,  
 3354 ["Gamma"] = 915,  
 3355 ["Delta"] = 916,  
 3356 ["Epsilon"] = 917,  
 3357 ["Zeta"] = 918,  
 3358 ["Eta"] = 919,  
 3359 ["Theta"] = 920,  
 3360 ["Iota"] = 921,  
 3361 ["Kappa"] = 922,  
 3362 ["Lambda"] = 923,  
 3363 ["Mu"] = 924,  
 3364 ["Nu"] = 925,  
 3365 ["Xi"] = 926,  
 3366 ["Omicron"] = 927,  
 3367 ["Pi"] = 928,  
 3368 ["Rho"] = 929,  
 3369 ["Sigma"] = 931,  
 3370 ["Tau"] = 932,  
 3371 ["Upsilon"] = 933,  
 3372 ["Phi"] = 934,  
 3373 ["Chi"] = 935,  
 3374 ["Psi"] = 936,  
 3375 ["Omega"] = 937,

```

3376 ["alpha"] = 945,
3377 ["beta"] = 946,
3378 ["gamma"] = 947,
3379 ["delta"] = 948,
3380 ["epsiv"] = 949,
3381 ["varepsilon"] = 949,
3382 ["epsilon"] = 949,
3383 ["zeta"] = 950,
3384 ["eta"] = 951,
3385 ["theta"] = 952,
3386 ["iota"] = 953,
3387 ["kappa"] = 954,
3388 ["lambda"] = 955,
3389 ["mu"] = 956,
3390 ["nu"] = 957,
3391 ["xi"] = 958,
3392 ["omicron"] = 959,
3393 ["pi"] = 960,
3394 ["rho"] = 961,
3395 ["sigmav"] = 962,
3396 ["varsigma"] = 962,
3397 ["sigmaf"] = 962,
3398 ["sigma"] = 963,
3399 ["tau"] = 964,
3400 ["upsi"] = 965,
3401 ["upsilon"] = 965,
3402 ["phi"] = 966,
3403 ["phiv"] = 966,
3404 ["varphi"] = 966,
3405 ["chi"] = 967,
3406 ["psi"] = 968,
3407 ["omega"] = 969,
3408 ["thetav"] = 977,
3409 ["vartheta"] = 977,
3410 ["thetasym"] = 977,
3411 ["Upsi"] = 978,
3412 ["upsih"] = 978,
3413 ["straightphi"] = 981,
3414 ["piv"] = 982,
3415 ["varpi"] = 982,
3416 ["Gammad"] = 988,
3417 ["gammad"] = 989,
3418 ["digamma"] = 989,
3419 ["kappav"] = 1008,
3420 ["varkappa"] = 1008,
3421 ["rhov"] = 1009,
3422 ["varrho"] = 1009,

```

```

3423 ["epsi"] = 1013,
3424 ["straightepsilon"] = 1013,
3425 ["bepsi"] = 1014,
3426 ["backepsilon"] = 1014,
3427 ["IOcy"] = 1025,
3428 ["DJcy"] = 1026,
3429 ["GJcy"] = 1027,
3430 ["Jukcy"] = 1028,
3431 ["DScy"] = 1029,
3432 ["Iukcy"] = 1030,
3433 ["YIcy"] = 1031,
3434 ["Jsercy"] = 1032,
3435 ["LJcy"] = 1033,
3436 ["NJcy"] = 1034,
3437 ["TSHcy"] = 1035,
3438 ["KJcy"] = 1036,
3439 ["Ubrcy"] = 1038,
3440 ["DZcy"] = 1039,
3441 ["Acy"] = 1040,
3442 ["Bcy"] = 1041,
3443 ["Vcy"] = 1042,
3444 ["Gcy"] = 1043,
3445 ["Dcy"] = 1044,
3446 ["IEcy"] = 1045,
3447 ["ZHcy"] = 1046,
3448 ["Zcy"] = 1047,
3449 ["Icy"] = 1048,
3450 ["Jcy"] = 1049,
3451 ["Kcy"] = 1050,
3452 ["Lcy"] = 1051,
3453 ["Mcy"] = 1052,
3454 ["Ncy"] = 1053,
3455 ["Ocy"] = 1054,
3456 ["Pcy"] = 1055,
3457 ["Rcy"] = 1056,
3458 ["Scy"] = 1057,
3459 ["Tcy"] = 1058,
3460 ["Ucy"] = 1059,
3461 ["Fcy"] = 1060,
3462 ["KHcy"] = 1061,
3463 ["TScy"] = 1062,
3464 ["CHcy"] = 1063,
3465 ["SHcy"] = 1064,
3466 ["SHCHcy"] = 1065,
3467 ["HARDcy"] = 1066,
3468 ["Ycy"] = 1067,
3469 ["SOFTcy"] = 1068,

```

```

3470 ["Ecy"] = 1069,
3471 ["YUcy"] = 1070,
3472 ["YAcy"] = 1071,
3473 ["acy"] = 1072,
3474 ["bcy"] = 1073,
3475 ["vcy"] = 1074,
3476 ["gcy"] = 1075,
3477 ["dcy"] = 1076,
3478 ["iecy"] = 1077,
3479 ["zhcy"] = 1078,
3480 ["zcy"] = 1079,
3481 ["icy"] = 1080,
3482 ["jcy"] = 1081,
3483 ["kcy"] = 1082,
3484 ["lcy"] = 1083,
3485 ["mcy"] = 1084,
3486 ["ncy"] = 1085,
3487 ["ocy"] = 1086,
3488 ["pcy"] = 1087,
3489 ["rcy"] = 1088,
3490 ["scy"] = 1089,
3491 ["tcy"] = 1090,
3492 ["ucy"] = 1091,
3493 ["fcy"] = 1092,
3494 ["khcy"] = 1093,
3495 ["tscy"] = 1094,
3496 ["chcy"] = 1095,
3497 ["shcy"] = 1096,
3498 ["shchcy"] = 1097,
3499 ["hardcy"] = 1098,
3500 ["ycy"] = 1099,
3501 ["softcy"] = 1100,
3502 ["ecy"] = 1101,
3503 ["yucy"] = 1102,
3504 ["yacy"] = 1103,
3505 ["iocy"] = 1105,
3506 ["djcy"] = 1106,
3507 ["gjcy"] = 1107,
3508 ["jukcy"] = 1108,
3509 ["dscy"] = 1109,
3510 ["iukcy"] = 1110,
3511 ["yicy"] = 1111,
3512 ["jsercy"] = 1112,
3513 ["ljcy"] = 1113,
3514 ["njcy"] = 1114,
3515 ["tshcy"] = 1115,
3516 ["kjcy"] = 1116,

```



```

3517 ["ubrcy"] = 1118,
3518 ["dzcy"] = 1119,
3519 ["ensp"] = 8194,
3520 ["emsp"] = 8195,
3521 ["emsp13"] = 8196,
3522 ["emsp14"] = 8197,
3523 ["numsp"] = 8199,
3524 ["puncsp"] = 8200,
3525 ["thinsp"] = 8201,
3526 ["ThinSpace"] = 8201,
3527 ["hairsp"] = 8202,
3528 ["VeryThinSpace"] = 8202,
3529 ["ZeroWidthSpace"] = 8203,
3530 ["NegativeVeryThinSpace"] = 8203,
3531 ["NegativeThinSpace"] = 8203,
3532 ["NegativeMediumSpace"] = 8203,
3533 ["NegativeThickSpace"] = 8203,
3534 ["zwnj"] = 8204,
3535 ["zwj"] = 8205,
3536 ["lrm"] = 8206,
3537 ["rlm"] = 8207,
3538 ["hyphen"] = 8208,
3539 ["dash"] = 8208,
3540 ["ndash"] = 8211,
3541 ["mdash"] = 8212,
3542 ["horbar"] = 8213,
3543 ["Verbar"] = 8214,
3544 ["Vert"] = 8214,
3545 ["lsquo"] = 8216,
3546 ["OpenCurlyQuote"] = 8216,
3547 ["rsquo"] = 8217,
3548 ["rsquor"] = 8217,
3549 ["CloseCurlyQuote"] = 8217,
3550 ["lsquor"] = 8218,
3551 ["sbquo"] = 8218,
3552 ["ldquo"] = 8220,
3553 ["OpenCurlyDoubleQuote"] = 8220,
3554 ["rdquo"] = 8221,
3555 ["rdquor"] = 8221,
3556 ["CloseCurlyDoubleQuote"] = 8221,
3557 ["ldquor"] = 8222,
3558 ["bdquo"] = 8222,
3559 ["dagger"] = 8224,
3560 ["Dagger"] = 8225,
3561 ["ddagger"] = 8225,
3562 ["bull"] = 8226,
3563 ["bullet"] = 8226,

```

```

3564 ["nldr"] = 8229,
3565 ["hellip"] = 8230,
3566 ["mldr"] = 8230,
3567 ["permil"] = 8240,
3568 ["pertenk"] = 8241,
3569 ["prime"] = 8242,
3570 ["Prime"] = 8243,
3571 ["tprime"] = 8244,
3572 ["bprime"] = 8245,
3573 ["backprime"] = 8245,
3574 ["lsaquo"] = 8249,
3575 ["rsaquo"] = 8250,
3576 ["oline"] = 8254,
3577 ["caret"] = 8257,
3578 ["hybull"] = 8259,
3579 ["frasl"] = 8260,
3580 ["bsemi"] = 8271,
3581 ["qprime"] = 8279,
3582 ["MediumSpace"] = 8287,
3583 ["NoBreak"] = 8288,
3584 ["ApplyFunction"] = 8289,
3585 ["af"] = 8289,
3586 ["InvisibleTimes"] = 8290,
3587 ["it"] = 8290,
3588 ["InvisibleComma"] = 8291,
3589 ["ic"] = 8291,
3590 ["euro"] = 8364,
3591 ["tdot"] = 8411,
3592 ["TripleDot"] = 8411,
3593 ["DotDot"] = 8412,
3594 ["Copf"] = 8450,
3595 ["complexes"] = 8450,
3596 ["incare"] = 8453,
3597 ["gscr"] = 8458,
3598 ["hamilt"] = 8459,
3599 ["HilbertSpace"] = 8459,
3600 ["Hscr"] = 8459,
3601 ["Hfr"] = 8460,
3602 ["Poincareplane"] = 8460,
3603 ["quaternions"] = 8461,
3604 ["Hopf"] = 8461,
3605 ["planckh"] = 8462,
3606 ["planck"] = 8463,
3607 ["hbar"] = 8463,
3608 ["plankv"] = 8463,
3609 ["hslash"] = 8463,
3610 ["Iscr"] = 8464,

```

```

3611 ["imagline"] = 8464,
3612 ["image"] = 8465,
3613 ["Im"] = 8465,
3614 ["imagpart"] = 8465,
3615 ["Ifr"] = 8465,
3616 ["Lscr"] = 8466,
3617 ["lagran"] = 8466,
3618 ["Laplacetrif"] = 8466,
3619 ["ell"] = 8467,
3620 ["Nopf"] = 8469,
3621 ["naturals"] = 8469,
3622 ["numero"] = 8470,
3623 ["copysr"] = 8471,
3624 ["weierp"] = 8472,
3625 ["wp"] = 8472,
3626 ["Popf"] = 8473,
3627 ["primes"] = 8473,
3628 ["rationals"] = 8474,
3629 ["Qopf"] = 8474,
3630 ["Rscr"] = 8475,
3631 ["realine"] = 8475,
3632 ["real"] = 8476,
3633 ["Re"] = 8476,
3634 ["realpart"] = 8476,
3635 ["Rfr"] = 8476,
3636 ["reals"] = 8477,
3637 ["Ropf"] = 8477,
3638 ["rx"] = 8478,
3639 ["trade"] = 8482,
3640 ["TRADE"] = 8482,
3641 ["integers"] = 8484,
3642 ["Zopf"] = 8484,
3643 ["ohm"] = 8486,
3644 ["mho"] = 8487,
3645 ["Zfr"] = 8488,
3646 ["zeetrif"] = 8488,
3647 ["iiota"] = 8489,
3648 ["angst"] = 8491,
3649 ["bernou"] = 8492,
3650 ["Bernoullis"] = 8492,
3651 ["Bscr"] = 8492,
3652 ["Cfr"] = 8493,
3653 ["Cayleys"] = 8493,
3654 ["escr"] = 8495,
3655 ["Escr"] = 8496,
3656 ["expectation"] = 8496,
3657 ["Fscr"] = 8497,

```

```

3658 ["Fouriertrf"] = 8497,
3659 ["phmmat"] = 8499,
3660 ["Mellintrf"] = 8499,
3661 ["Mscr"] = 8499,
3662 ["order"] = 8500,
3663 ["orderof"] = 8500,
3664 ["oscr"] = 8500,
3665 ["alefsym"] = 8501,
3666 ["aleph"] = 8501,
3667 ["beth"] = 8502,
3668 ["gimel"] = 8503,
3669 ["daleth"] = 8504,
3670 ["CapitalDifferentialD"] = 8517,
3671 ["DD"] = 8517,
3672 ["DifferentialD"] = 8518,
3673 ["dd"] = 8518,
3674 ["ExponentialE"] = 8519,
3675 ["exponentiale"] = 8519,
3676 ["ee"] = 8519,
3677 ["ImaginaryI"] = 8520,
3678 ["ii"] = 8520,
3679 ["frac13"] = 8531,
3680 ["frac23"] = 8532,
3681 ["frac15"] = 8533,
3682 ["frac25"] = 8534,
3683 ["frac35"] = 8535,
3684 ["frac45"] = 8536,
3685 ["frac16"] = 8537,
3686 ["frac56"] = 8538,
3687 ["frac18"] = 8539,
3688 ["frac38"] = 8540,
3689 ["frac58"] = 8541,
3690 ["frac78"] = 8542,
3691 ["larr"] = 8592,
3692 ["leftarrow"] = 8592,
3693 ["LeftArrow"] = 8592,
3694 ["slarr"] = 8592,
3695 ["ShortLeftArrow"] = 8592,
3696 ["uarr"] = 8593,
3697 ["uparrow"] = 8593,
3698 ["UpArrow"] = 8593,
3699 ["ShortUpArrow"] = 8593,
3700 ["rarr"] = 8594,
3701 ["rightarrow"] = 8594,
3702 ["RightArrow"] = 8594,
3703 ["srarr"] = 8594,
3704 ["ShortRightArrow"] = 8594,

```

```

3705 ["darr"] = 8595,
3706 ["downarrow"] = 8595,
3707 ["DownArrow"] = 8595,
3708 ["ShortDownArrow"] = 8595,
3709 ["harr"] = 8596,
3710 ["leftrightharpoon"] = 8596,
3711 ["LeftRightArrow"] = 8596,
3712 ["varr"] = 8597,
3713 ["updownarrow"] = 8597,
3714 ["UpDownArrow"] = 8597,
3715 ["nwarr"] = 8598,
3716 ["UpperLeftArrow"] = 8598,
3717 ["nwarpoon"] = 8598,
3718 ["nearr"] = 8599,
3719 ["UpperRightArrow"] = 8599,
3720 ["nearrow"] = 8599,
3721 ["searr"] = 8600,
3722 ["searrow"] = 8600,
3723 ["LowerRightArrow"] = 8600,
3724 ["swarr"] = 8601,
3725 ["swarrow"] = 8601,
3726 ["LowerLeftArrow"] = 8601,
3727 ["nlarr"] = 8602,
3728 ["nleftarrow"] = 8602,
3729 ["nrarr"] = 8603,
3730 ["nrightarrow"] = 8603,
3731 ["rarrw"] = 8605,
3732 ["rightsquigarrow"] = 8605,
3733 ["Larr"] = 8606,
3734 ["twoheadleftarrow"] = 8606,
3735 ["Uarr"] = 8607,
3736 ["Rarr"] = 8608,
3737 ["twoheadrightarrow"] = 8608,
3738 ["Darr"] = 8609,
3739 ["larrtl"] = 8610,
3740 ["leftarrowtail"] = 8610,
3741 ["rarrtl"] = 8611,
3742 ["rightarrowtail"] = 8611,
3743 ["LeftTeeArrow"] = 8612,
3744 ["mapstoleft"] = 8612,
3745 ["UpTeeArrow"] = 8613,
3746 ["mapstoup"] = 8613,
3747 ["map"] = 8614,
3748 ["RightTeeArrow"] = 8614,
3749 ["mapsto"] = 8614,
3750 ["DownTeeArrow"] = 8615,
3751 ["mapstodown"] = 8615,

```

```

3752 ["larrhk"] = 8617,
3753 ["hookleftarrow"] = 8617,
3754 ["rarrhk"] = 8618,
3755 ["hookrightarrow"] = 8618,
3756 ["larrlp"] = 8619,
3757 ["looparrowleft"] = 8619,
3758 ["rarrlp"] = 8620,
3759 ["looparrowright"] = 8620,
3760 ["harrw"] = 8621,
3761 ["leftrightsquigarrow"] = 8621,
3762 ["nharr"] = 8622,
3763 ["nletrightarrow"] = 8622,
3764 ["lsh"] = 8624,
3765 ["Lsh"] = 8624,
3766 ["rsh"] = 8625,
3767 ["Rsh"] = 8625,
3768 ["ldsh"] = 8626,
3769 ["rdsh"] = 8627,
3770 ["crarr"] = 8629,
3771 ["cularr"] = 8630,
3772 ["curvearrowleft"] = 8630,
3773 ["curarr"] = 8631,
3774 ["curvearrowright"] = 8631,
3775 ["olarr"] = 8634,
3776 ["circlearrowleft"] = 8634,
3777 ["orarr"] = 8635,
3778 ["circlearrowright"] = 8635,
3779 ["lharu"] = 8636,
3780 ["LeftVector"] = 8636,
3781 ["leftharpoonup"] = 8636,
3782 ["lhard"] = 8637,
3783 ["leftharpoondown"] = 8637,
3784 ["DownLeftVector"] = 8637,
3785 ["uharr"] = 8638,
3786 ["upharpoonright"] = 8638,
3787 ["RightUpVector"] = 8638,
3788 ["uharl"] = 8639,
3789 ["upharpoonleft"] = 8639,
3790 ["LeftUpVector"] = 8639,
3791 ["rharu"] = 8640,
3792 ["RightVector"] = 8640,
3793 ["rightharpoonup"] = 8640,
3794 ["rhard"] = 8641,
3795 ["rightharpoondown"] = 8641,
3796 ["DownRightVector"] = 8641,
3797 ["dharr"] = 8642,
3798 ["RightDownVector"] = 8642,

```

```

3799 ["downharpoonright"] = 8642,
3800 ["dharl"] = 8643,
3801 ["LeftDownVector"] = 8643,
3802 ["downharpoonleft"] = 8643,
3803 ["rlarr"] = 8644,
3804 ["rightleftarrows"] = 8644,
3805 ["RightArrowLeftArrow"] = 8644,
3806 ["udarr"] = 8645,
3807 ["UpArrowDownArrow"] = 8645,
3808 ["lrarr"] = 8646,
3809 ["leftrightarrows"] = 8646,
3810 ["LeftArrowRightArrow"] = 8646,
3811 ["llarr"] = 8647,
3812 ["leftleftarrows"] = 8647,
3813 ["uuarr"] = 8648,
3814 ["upuparrows"] = 8648,
3815 ["rrarr"] = 8649,
3816 ["rightrightarrows"] = 8649,
3817 ["ddarr"] = 8650,
3818 ["downdownarrows"] = 8650,
3819 ["lrhar"] = 8651,
3820 ["ReverseEquilibrium"] = 8651,
3821 ["leftrightharpoons"] = 8651,
3822 ["rlhar"] = 8652,
3823 ["rightleftharpoons"] = 8652,
3824 ["Equilibrium"] = 8652,
3825 ["nlArr"] = 8653,
3826 ["nLeftarrow"] = 8653,
3827 ["nhArr"] = 8654,
3828 ["nLeftrightarrow"] = 8654,
3829 ["nrArr"] = 8655,
3830 ["nRightarrow"] = 8655,
3831 ["lArr"] = 8656,
3832 ["Leftarrow"] = 8656,
3833 ["DoubleLeftArrow"] = 8656,
3834 ["uArr"] = 8657,
3835 ["Uparrow"] = 8657,
3836 ["DoubleUpArrow"] = 8657,
3837 ["rArr"] = 8658,
3838 ["Rightarrow"] = 8658,
3839 ["Implies"] = 8658,
3840 ["DoubleRightArrow"] = 8658,
3841 ["dArr"] = 8659,
3842 ["Downarrow"] = 8659,
3843 ["DoubleDownArrow"] = 8659,
3844 ["hArr"] = 8660,
3845 ["Leftrightarrow"] = 8660,

```

```

3846 ["DoubleLeftRightArrow"] = 8660,
3847 ["iff"] = 8660,
3848 ["vArr"] = 8661,
3849 ["Updownarrow"] = 8661,
3850 ["DoubleUpDownArrow"] = 8661,
3851 ["nwArr"] = 8662,
3852 ["neArr"] = 8663,
3853 ["seArr"] = 8664,
3854 ["swArr"] = 8665,
3855 ["lAarr"] = 8666,
3856 ["Lleftarrow"] = 8666,
3857 ["rAarr"] = 8667,
3858 ["Rrightarrow"] = 8667,
3859 ["zigrarr"] = 8669,
3860 ["larrb"] = 8676,
3861 ["LeftArrowBar"] = 8676,
3862 ["rarrb"] = 8677,
3863 ["RightArrowBar"] = 8677,
3864 ["duarr"] = 8693,
3865 ["DownArrowUpArrow"] = 8693,
3866 ["loarr"] = 8701,
3867 ["roarr"] = 8702,
3868 ["hoarr"] = 8703,
3869 ["forall"] = 8704,
3870 ["ForAll"] = 8704,
3871 ["comp"] = 8705,
3872 ["complement"] = 8705,
3873 ["part"] = 8706,
3874 ["PartialD"] = 8706,
3875 ["exist"] = 8707,
3876 ["Exists"] = 8707,
3877 ["nexist"] = 8708,
3878 ["NotExists"] = 8708,
3879 ["nexists"] = 8708,
3880 ["empty"] = 8709,
3881 ["emptyset"] = 8709,
3882 ["emptyv"] = 8709,
3883 ["varnothing"] = 8709,
3884 ["nabla"] = 8711,
3885 ["Del"] = 8711,
3886 ["isin"] = 8712,
3887 ["isinv"] = 8712,
3888 ["Element"] = 8712,
3889 ["in"] = 8712,
3890 ["notin"] = 8713,
3891 ["NotElement"] = 8713,
3892 ["notinva"] = 8713,

```



```

3893 ["niv"] = 8715,
3894 ["ReverseElement"] = 8715,
3895 ["ni"] = 8715,
3896 ["SuchThat"] = 8715,
3897 ["notni"] = 8716,
3898 ["notniva"] = 8716,
3899 ["NotReverseElement"] = 8716,
3900 ["prod"] = 8719,
3901 ["Product"] = 8719,
3902 ["coprod"] = 8720,
3903 ["Coproduct"] = 8720,
3904 ["sum"] = 8721,
3905 ["Sum"] = 8721,
3906 ["minus"] = 8722,
3907 ["mnplus"] = 8723,
3908 ["mp"] = 8723,
3909 ["MinusPlus"] = 8723,
3910 ["plusdo"] = 8724,
3911 ["dotplus"] = 8724,
3912 ["setmn"] = 8726,
3913 ["setminus"] = 8726,
3914 ["Backslash"] = 8726,
3915 ["ssetmn"] = 8726,
3916 ["smallsetminus"] = 8726,
3917 ["lowast"] = 8727,
3918 ["compfn"] = 8728,
3919 ["SmallCircle"] = 8728,
3920 ["radic"] = 8730,
3921 ["Sqrt"] = 8730,
3922 ["prop"] = 8733,
3923 ["propto"] = 8733,
3924 ["Proportional"] = 8733,
3925 ["vprop"] = 8733,
3926 ["varpropto"] = 8733,
3927 ["infin"] = 8734,
3928 ["angrt"] = 8735,
3929 ["ang"] = 8736,
3930 ["angle"] = 8736,
3931 ["angmsd"] = 8737,
3932 ["measuredangle"] = 8737,
3933 ["angsph"] = 8738,
3934 ["mid"] = 8739,
3935 ["VerticalBar"] = 8739,
3936 ["smid"] = 8739,
3937 ["shortmid"] = 8739,
3938 ["nmid"] = 8740,
3939 ["NotVerticalBar"] = 8740,

```

```

3940 ["nsmid"] = 8740,
3941 ["nshortmid"] = 8740,
3942 ["par"] = 8741,
3943 ["parallel"] = 8741,
3944 ["DoubleVerticalBar"] = 8741,
3945 ["spar"] = 8741,
3946 ["shortparallel"] = 8741,
3947 ["npar"] = 8742,
3948 ["nparallel"] = 8742,
3949 ["NotDoubleVerticalBar"] = 8742,
3950 ["nspar"] = 8742,
3951 ["nshortparallel"] = 8742,
3952 ["and"] = 8743,
3953 ["wedge"] = 8743,
3954 ["or"] = 8744,
3955 ["vee"] = 8744,
3956 ["cap"] = 8745,
3957 ["cup"] = 8746,
3958 ["int"] = 8747,
3959 ["Integral"] = 8747,
3960 ["Int"] = 8748,
3961 ["tint"] = 8749,
3962 ["iiint"] = 8749,
3963 ["conint"] = 8750,
3964 ["oint"] = 8750,
3965 ["ContourIntegral"] = 8750,
3966 ["Conint"] = 8751,
3967 ["DoubleContourIntegral"] = 8751,
3968 ["Cconint"] = 8752,
3969 ["cwint"] = 8753,
3970 ["cwconint"] = 8754,
3971 ["ClockwiseContourIntegral"] = 8754,
3972 ["awconint"] = 8755,
3973 ["CounterClockwiseContourIntegral"] = 8755,
3974 ["there4"] = 8756,
3975 ["therefore"] = 8756,
3976 ["Therefore"] = 8756,
3977 ["because"] = 8757,
3978 ["because"] = 8757,
3979 ["Because"] = 8757,
3980 ["ratio"] = 8758,
3981 ["Colon"] = 8759,
3982 ["Proportion"] = 8759,
3983 ["minusd"] = 8760,
3984 ["dotminus"] = 8760,
3985 ["mDDot"] = 8762,
3986 ["homtht"] = 8763,

```

```

3987 ["sim"] = 8764,
3988 ["Tilde"] = 8764,
3989 ["thksim"] = 8764,
3990 ["thicksim"] = 8764,
3991 ["bsim"] = 8765,
3992 ["backsim"] = 8765,
3993 ["ac"] = 8766,
3994 ["mstpos"] = 8766,
3995 ["acd"] = 8767,
3996 ["wreath"] = 8768,
3997 ["VerticalTilde"] = 8768,
3998 ["wr"] = 8768,
3999 ["nsim"] = 8769,
4000 ["NotTilde"] = 8769,
4001 ["esim"] = 8770,
4002 ["EqualTilde"] = 8770,
4003 ["eqsim"] = 8770,
4004 ["sime"] = 8771,
4005 ["TildeEqual"] = 8771,
4006 ["simeq"] = 8771,
4007 ["nsime"] = 8772,
4008 ["nsimeq"] = 8772,
4009 ["NotTildeEqual"] = 8772,
4010 ["cong"] = 8773,
4011 ["TildeFullEqual"] = 8773,
4012 ["simne"] = 8774,
4013 ["ncong"] = 8775,
4014 ["NotTildeFullEqual"] = 8775,
4015 ["asymp"] = 8776,
4016 ["ap"] = 8776,
4017 ["TildeTilde"] = 8776,
4018 ["approx"] = 8776,
4019 ["thkap"] = 8776,
4020 ["thickapprox"] = 8776,
4021 ["nap"] = 8777,
4022 ["NotTildeTilde"] = 8777,
4023 ["napprox"] = 8777,
4024 ["ape"] = 8778,
4025 ["approxpeq"] = 8778,
4026 ["apid"] = 8779,
4027 ["bcong"] = 8780,
4028 ["backcong"] = 8780,
4029 ["asympeq"] = 8781,
4030 ["CupCap"] = 8781,
4031 ["bump"] = 8782,
4032 ["HumpDownHump"] = 8782,
4033 ["Bumpeq"] = 8782,

```

```

4034 ["bumpe"] = 8783,
4035 ["HumpEqual"] = 8783,
4036 ["bumpeq"] = 8783,
4037 ["esdot"] = 8784,
4038 ["DotEqual"] = 8784,
4039 ["doteq"] = 8784,
4040 ["eDot"] = 8785,
4041 ["doteqdot"] = 8785,
4042 ["efDot"] = 8786,
4043 ["fallingdotseq"] = 8786,
4044 ["erDot"] = 8787,
4045 ["risingdotseq"] = 8787,
4046 ["colone"] = 8788,
4047 ["coloneq"] = 8788,
4048 ["Assign"] = 8788,
4049 ["ecolon"] = 8789,
4050 ["eqcolon"] = 8789,
4051 ["ecir"] = 8790,
4052 ["eqcirc"] = 8790,
4053 ["cire"] = 8791,
4054 ["circeq"] = 8791,
4055 ["wedgeq"] = 8793,
4056 ["veeeq"] = 8794,
4057 ["trie"] = 8796,
4058 ["triangleq"] = 8796,
4059 ["equest"] = 8799,
4060 ["questeq"] = 8799,
4061 ["ne"] = 8800,
4062 ["NotEqual"] = 8800,
4063 ["equiv"] = 8801,
4064 ["Congruent"] = 8801,
4065 ["nequiv"] = 8802,
4066 ["NotCongruent"] = 8802,
4067 ["le"] = 8804,
4068 ["leq"] = 8804,
4069 ["ge"] = 8805,
4070 ["GreaterEqual"] = 8805,
4071 ["geq"] = 8805,
4072 ["lE"] = 8806,
4073 ["LessFullEqual"] = 8806,
4074 ["leqq"] = 8806,
4075 ["gE"] = 8807,
4076 ["GreaterFullEqual"] = 8807,
4077 ["geqq"] = 8807,
4078 ["lnE"] = 8808,
4079 ["lneqq"] = 8808,
4080 ["gnE"] = 8809,

```

```

4081 ["gneqq"] = 8809,
4082 ["Lt"] = 8810,
4083 ["NestedLessLess"] = 8810,
4084 ["ll"] = 8810,
4085 ["Gt"] = 8811,
4086 ["NestedGreaterGreater"] = 8811,
4087 ["gg"] = 8811,
4088 ["twixt"] = 8812,
4089 ["between"] = 8812,
4090 ["NotCupCap"] = 8813,
4091 ["nlt"] = 8814,
4092 ["NotLess"] = 8814,
4093 ["nless"] = 8814,
4094 ["ngt"] = 8815,
4095 ["NotGreater"] = 8815,
4096 ["ngtr"] = 8815,
4097 ["nle"] = 8816,
4098 ["NotLessEqual"] = 8816,
4099 ["nleq"] = 8816,
4100 ["nge"] = 8817,
4101 ["NotGreaterEqual"] = 8817,
4102 ["ngeq"] = 8817,
4103 ["lsim"] = 8818,
4104 ["LessTilde"] = 8818,
4105 ["lesssim"] = 8818,
4106 ["gsim"] = 8819,
4107 ["gtrsim"] = 8819,
4108 ["GreaterTilde"] = 8819,
4109 ["nlsim"] = 8820,
4110 ["NotLessTilde"] = 8820,
4111 ["ngsim"] = 8821,
4112 ["NotGreaterTilde"] = 8821,
4113 ["lg"] = 8822,
4114 ["lessgtr"] = 8822,
4115 ["LessGreater"] = 8822,
4116 ["gl"] = 8823,
4117 ["gtrless"] = 8823,
4118 ["GreaterLess"] = 8823,
4119 ["ntlg"] = 8824,
4120 ["NotLessGreater"] = 8824,
4121 ["ntgl"] = 8825,
4122 ["NotGreaterLess"] = 8825,
4123 ["pr"] = 8826,
4124 ["Precedes"] = 8826,
4125 ["prec"] = 8826,
4126 ["sc"] = 8827,
4127 ["Succeeds"] = 8827,

```

```

4128 ["succ"] = 8827,
4129 ["prcue"] = 8828,
4130 ["PrecedesSlantEqual"] = 8828,
4131 ["preccurlyeq"] = 8828,
4132 ["sccue"] = 8829,
4133 ["SucceedsSlantEqual"] = 8829,
4134 ["succcurlyeq"] = 8829,
4135 ["prsim"] = 8830,
4136 ["precsim"] = 8830,
4137 ["PrecedesTilde"] = 8830,
4138 ["scsim"] = 8831,
4139 ["succsim"] = 8831,
4140 ["SucceedsTilde"] = 8831,
4141 ["npr"] = 8832,
4142 ["nprec"] = 8832,
4143 ["NotPrecedes"] = 8832,
4144 ["nsc"] = 8833,
4145 ["nsucc"] = 8833,
4146 ["NotSucceeds"] = 8833,
4147 ["sub"] = 8834,
4148 ["subset"] = 8834,
4149 ["sup"] = 8835,
4150 ["supset"] = 8835,
4151 ["Superset"] = 8835,
4152 ["nsub"] = 8836,
4153 ["nsup"] = 8837,
4154 ["sube"] = 8838,
4155 ["SubsetEqual"] = 8838,
4156 ["subseteq"] = 8838,
4157 ["supe"] = 8839,
4158 ["supseteq"] = 8839,
4159 ["SupersetEqual"] = 8839,
4160 ["nsube"] = 8840,
4161 ["nsubseteq"] = 8840,
4162 ["NotSubsetEqual"] = 8840,
4163 ["nsupe"] = 8841,
4164 ["nsupseteq"] = 8841,
4165 ["NotSupersetEqual"] = 8841,
4166 ["subne"] = 8842,
4167 ["subsetneq"] = 8842,
4168 ["supne"] = 8843,
4169 ["supsetneq"] = 8843,
4170 ["cupdot"] = 8845,
4171 ["uplus"] = 8846,
4172 ["UnionPlus"] = 8846,
4173 ["sqsub"] = 8847,
4174 ["SquareSubset"] = 8847,

```

```

4175 ["sqsubset"] = 8847,
4176 ["sqsup"] = 8848,
4177 ["SquareSuperset"] = 8848,
4178 ["sqsupset"] = 8848,
4179 ["sqsube"] = 8849,
4180 ["SquareSubsetEqual"] = 8849,
4181 ["sqsubseteq"] = 8849,
4182 ["sqsupe"] = 8850,
4183 ["SquareSupersetEqual"] = 8850,
4184 ["sqsupseteq"] = 8850,
4185 ["sqcap"] = 8851,
4186 ["SquareIntersection"] = 8851,
4187 ["sqcup"] = 8852,
4188 ["SquareUnion"] = 8852,
4189 ["oplus"] = 8853,
4190 ["CirclePlus"] = 8853,
4191 ["ominus"] = 8854,
4192 ["CircleMinus"] = 8854,
4193 ["otimes"] = 8855,
4194 ["CircleTimes"] = 8855,
4195 ["osol"] = 8856,
4196 ["odot"] = 8857,
4197 ["CircleDot"] = 8857,
4198 ["ocir"] = 8858,
4199 ["circledcirc"] = 8858,
4200 ["oast"] = 8859,
4201 ["circledast"] = 8859,
4202 ["odash"] = 8861,
4203 ["circleddash"] = 8861,
4204 ["plusb"] = 8862,
4205 ["boxplus"] = 8862,
4206 ["minusb"] = 8863,
4207 ["boxminus"] = 8863,
4208 ["timesb"] = 8864,
4209 ["boxtimes"] = 8864,
4210 ["sdotb"] = 8865,
4211 ["dotsquare"] = 8865,
4212 ["vdash"] = 8866,
4213 ["RightTee"] = 8866,
4214 ["dashv"] = 8867,
4215 ["LeftTee"] = 8867,
4216 ["top"] = 8868,
4217 ["DownTee"] = 8868,
4218 ["bottom"] = 8869,
4219 ["bot"] = 8869,
4220 ["perp"] = 8869,
4221 ["UpTee"] = 8869,

```

```

4222 ["models"] = 8871,
4223 ["vDash"] = 8872,
4224 ["DoubleRightTee"] = 8872,
4225 ["Vdash"] = 8873,
4226 ["Vvdash"] = 8874,
4227 ["VDash"] = 8875,
4228 ["nvdash"] = 8876,
4229 ["nvDash"] = 8877,
4230 ["nVdash"] = 8878,
4231 ["nVDash"] = 8879,
4232 ["prurel"] = 8880,
4233 ["vltri"] = 8882,
4234 ["vartriangleleft"] = 8882,
4235 ["LeftTriangle"] = 8882,
4236 ["vrtri"] = 8883,
4237 ["vartriangleright"] = 8883,
4238 ["RightTriangle"] = 8883,
4239 ["ltrie"] = 8884,
4240 ["trianglelefteq"] = 8884,
4241 ["LeftTriangleEqual"] = 8884,
4242 ["rtrie"] = 8885,
4243 ["trianglerighteq"] = 8885,
4244 ["RightTriangleEqual"] = 8885,
4245 ["origof"] = 8886,
4246 ["imof"] = 8887,
4247 ["mumap"] = 8888,
4248 ["multimap"] = 8888,
4249 ["hercon"] = 8889,
4250 ["intcal"] = 8890,
4251 ["intercal"] = 8890,
4252 ["veebar"] = 8891,
4253 ["barvee"] = 8893,
4254 ["angrtvb"] = 8894,
4255 ["ltri"] = 8895,
4256 ["xwedge"] = 8896,
4257 ["Wedge"] = 8896,
4258 ["bigwedge"] = 8896,
4259 ["xvee"] = 8897,
4260 ["Vee"] = 8897,
4261 ["bigvee"] = 8897,
4262 ["xcap"] = 8898,
4263 ["Intersection"] = 8898,
4264 ["bigcap"] = 8898,
4265 ["xcup"] = 8899,
4266 ["Union"] = 8899,
4267 ["bigcup"] = 8899,
4268 ["diam"] = 8900,

```



```

4269 ["diamond"] = 8900,
4270 ["Diamond"] = 8900,
4271 ["sdot"] = 8901,
4272 ["sstarf"] = 8902,
4273 ["Star"] = 8902,
4274 ["divonx"] = 8903,
4275 ["divideontimes"] = 8903,
4276 ["bowtie"] = 8904,
4277 ["ltimes"] = 8905,
4278 ["rtimes"] = 8906,
4279 ["lthree"] = 8907,
4280 ["leftthreetimes"] = 8907,
4281 ["rthree"] = 8908,
4282 ["rightthreetimes"] = 8908,
4283 ["bsime"] = 8909,
4284 ["backsimeq"] = 8909,
4285 ["cuvee"] = 8910,
4286 ["curlyvee"] = 8910,
4287 ["cuwed"] = 8911,
4288 ["curlywedge"] = 8911,
4289 ["Sub"] = 8912,
4290 ["Subset"] = 8912,
4291 ["Sup"] = 8913,
4292 ["Supset"] = 8913,
4293 ["Cap"] = 8914,
4294 ["Cup"] = 8915,
4295 ["fork"] = 8916,
4296 ["pitchfork"] = 8916,
4297 ["epar"] = 8917,
4298 ["ltdot"] = 8918,
4299 ["lessdot"] = 8918,
4300 ["gtdot"] = 8919,
4301 ["gtrdot"] = 8919,
4302 ["Ll"] = 8920,
4303 ["Gg"] = 8921,
4304 ["ggg"] = 8921,
4305 ["leg"] = 8922,
4306 ["LessEqualGreater"] = 8922,
4307 ["lesseqgtr"] = 8922,
4308 ["gel"] = 8923,
4309 ["gtreqless"] = 8923,
4310 ["GreaterEqualLess"] = 8923,
4311 ["cuepr"] = 8926,
4312 ["curlyeqprec"] = 8926,
4313 ["cuesc"] = 8927,
4314 ["curlyeqsucc"] = 8927,
4315 ["nprcue"] = 8928,

```

```

4316 ["NotPrecedesSlantEqual"] = 8928,
4317 ["nsccue"] = 8929,
4318 ["NotSucceedsSlantEqual"] = 8929,
4319 ["nsqsube"] = 8930,
4320 ["NotSquareSubsetEqual"] = 8930,
4321 ["nsqsupe"] = 8931,
4322 ["NotSquareSupersetEqual"] = 8931,
4323 ["lnsim"] = 8934,
4324 ["gnsim"] = 8935,
4325 ["prnsim"] = 8936,
4326 ["precnsim"] = 8936,
4327 ["scnsim"] = 8937,
4328 ["succnsim"] = 8937,
4329 ["nltri"] = 8938,
4330 ["ntriangleleft"] = 8938,
4331 ["NotLeftTriangle"] = 8938,
4332 ["nrtri"] = 8939,
4333 ["ntriangleright"] = 8939,
4334 ["NotRightTriangle"] = 8939,
4335 ["nltrie"] = 8940,
4336 ["ntrianglelefteq"] = 8940,
4337 ["NotLeftTriangleEqual"] = 8940,
4338 ["nrtrie"] = 8941,
4339 ["ntrianglerighteq"] = 8941,
4340 ["NotRightTriangleEqual"] = 8941,
4341 ["vellip"] = 8942,
4342 ["ctdot"] = 8943,
4343 ["utdot"] = 8944,
4344 ["dtdot"] = 8945,
4345 ["disin"] = 8946,
4346 ["isinsv"] = 8947,
4347 ["isins"] = 8948,
4348 ["isindot"] = 8949,
4349 ["notinvc"] = 8950,
4350 ["notinvb"] = 8951,
4351 ["isinE"] = 8953,
4352 ["nisd"] = 8954,
4353 ["xnis"] = 8955,
4354 ["nis"] = 8956,
4355 ["notnivc"] = 8957,
4356 ["notnivb"] = 8958,
4357 ["barwed"] = 8965,
4358 ["barwedge"] = 8965,
4359 ["Barwed"] = 8966,
4360 ["doublebarwedge"] = 8966,
4361 ["lceil"] = 8968,
4362 ["LeftCeiling"] = 8968,

```

```

4363 ["rceil"] = 8969,
4364 ["RightCeiling"] = 8969,
4365 ["lfloor"] = 8970,
4366 ["LeftFloor"] = 8970,
4367 ["rfloor"] = 8971,
4368 ["RightFloor"] = 8971,
4369 ["drcrop"] = 8972,
4370 ["dlcrop"] = 8973,
4371 ["urcrop"] = 8974,
4372 ["ulcrop"] = 8975,
4373 ["bnot"] = 8976,
4374 ["proflin"] = 8978,
4375 ["profsurf"] = 8979,
4376 ["telrec"] = 8981,
4377 ["target"] = 8982,
4378 ["ulcorn"] = 8988,
4379 ["ulcorner"] = 8988,
4380 ["urcorn"] = 8989,
4381 ["urcorner"] = 8989,
4382 ["dlcorn"] = 8990,
4383 ["llcorner"] = 8990,
4384 ["drcorn"] = 8991,
4385 ["lrcorn"] = 8991,
4386 ["frown"] = 8994,
4387 ["sfrown"] = 8994,
4388 ["smile"] = 8995,
4389 ["ssmile"] = 8995,
4390 ["cylcty"] = 9005,
4391 ["profalar"] = 9006,
4392 ["topbot"] = 9014,
4393 ["ovbar"] = 9021,
4394 ["solbar"] = 9023,
4395 ["angzarr"] = 9084,
4396 ["lmoust"] = 9136,
4397 ["lmoustache"] = 9136,
4398 ["rmoust"] = 9137,
4399 ["rmoustache"] = 9137,
4400 ["tbrk"] = 9140,
4401 ["OverBracket"] = 9140,
4402 ["bbrk"] = 9141,
4403 ["UnderBracket"] = 9141,
4404 ["bbrktbrk"] = 9142,
4405 ["OverParenthesis"] = 9180,
4406 ["UnderParenthesis"] = 9181,
4407 ["OverBrace"] = 9182,
4408 ["UnderBrace"] = 9183,
4409 ["trpezium"] = 9186,

```

```

4410 ["elinters"] = 9191,
4411 ["blank"] = 9251,
4412 ["oS"] = 9416,
4413 ["circledS"] = 9416,
4414 ["boxh"] = 9472,
4415 ["HorizontalLine"] = 9472,
4416 ["boxv"] = 9474,
4417 ["boxdr"] = 9484,
4418 ["boxdl"] = 9488,
4419 ["boxur"] = 9492,
4420 ["boxul"] = 9496,
4421 ["boxvr"] = 9500,
4422 ["boxvl"] = 9508,
4423 ["boxhd"] = 9516,
4424 ["boxhu"] = 9524,
4425 ["boxvh"] = 9532,
4426 ["boxH"] = 9552,
4427 ["boxV"] = 9553,
4428 ["boxdR"] = 9554,
4429 ["boxDr"] = 9555,
4430 ["boxDR"] = 9556,
4431 ["boxdL"] = 9557,
4432 ["boxDL"] = 9558,
4433 ["boxDL"] = 9559,
4434 ["boxuR"] = 9560,
4435 ["boxUr"] = 9561,
4436 ["boxUR"] = 9562,
4437 ["boxuL"] = 9563,
4438 ["boxUL"] = 9564,
4439 ["boxUL"] = 9565,
4440 ["boxvR"] = 9566,
4441 ["boxVr"] = 9567,
4442 ["boxVR"] = 9568,
4443 ["boxvL"] = 9569,
4444 ["boxVL"] = 9570,
4445 ["boxVL"] = 9571,
4446 ["boxHd"] = 9572,
4447 ["boxhD"] = 9573,
4448 ["boxHD"] = 9574,
4449 ["boxHu"] = 9575,
4450 ["boxhU"] = 9576,
4451 ["boxHU"] = 9577,
4452 ["boxvH"] = 9578,
4453 ["boxVh"] = 9579,
4454 ["boxVH"] = 9580,
4455 ["uhblk"] = 9600,
4456 ["lhblk"] = 9604,

```

```

4457 ["block"] = 9608,
4458 ["blk14"] = 9617,
4459 ["blk12"] = 9618,
4460 ["blk34"] = 9619,
4461 ["squ"] = 9633,
4462 ["square"] = 9633,
4463 ["Square"] = 9633,
4464 ["squf"] = 9642,
4465 ["squarf"] = 9642,
4466 ["blacksquare"] = 9642,
4467 ["FilledVerySmallSquare"] = 9642,
4468 ["EmptyVerySmallSquare"] = 9643,
4469 ["rect"] = 9645,
4470 ["marker"] = 9646,
4471 ["fltns"] = 9649,
4472 ["xutri"] = 9651,
4473 ["bigtriangleup"] = 9651,
4474 ["utrif"] = 9652,
4475 ["blacktriangle"] = 9652,
4476 ["utri"] = 9653,
4477 ["triangle"] = 9653,
4478 ["rtrif"] = 9656,
4479 ["blacktriangleright"] = 9656,
4480 ["rtri"] = 9657,
4481 ["triangleright"] = 9657,
4482 ["xdtri"] = 9661,
4483 ["bigtriangledown"] = 9661,
4484 ["dtrif"] = 9662,
4485 ["blacktriangledown"] = 9662,
4486 ["dtri"] = 9663,
4487 ["triangledown"] = 9663,
4488 ["ltrif"] = 9666,
4489 ["blacktriangleleft"] = 9666,
4490 ["ltri"] = 9667,
4491 ["triangleleft"] = 9667,
4492 ["loz"] = 9674,
4493 ["lozenge"] = 9674,
4494 ["cir"] = 9675,
4495 ["tridot"] = 9708,
4496 ["xcirc"] = 9711,
4497 ["bigcirc"] = 9711,
4498 ["ultri"] = 9720,
4499 ["urtri"] = 9721,
4500 ["lltri"] = 9722,
4501 ["EmptySmallSquare"] = 9723,
4502 ["FilledSmallSquare"] = 9724,
4503 ["starf"] = 9733,

```

```

4504 ["bigstar"] = 9733,
4505 ["star"] = 9734,
4506 ["phone"] = 9742,
4507 ["female"] = 9792,
4508 ["male"] = 9794,
4509 ["spades"] = 9824,
4510 ["spadesuit"] = 9824,
4511 ["clubs"] = 9827,
4512 ["clubsuit"] = 9827,
4513 ["hearts"] = 9829,
4514 ["heartsuit"] = 9829,
4515 ["diams"] = 9830,
4516 ["diamondsuit"] = 9830,
4517 ["sung"] = 9834,
4518 ["flat"] = 9837,
4519 ["natur"] = 9838,
4520 ["natural"] = 9838,
4521 ["sharp"] = 9839,
4522 ["check"] = 10003,
4523 ["checkmark"] = 10003,
4524 ["cross"] = 10007,
4525 ["malt"] = 10016,
4526 ["maltese"] = 10016,
4527 ["sext"] = 10038,
4528 ["VerticalSeparator"] = 10072,
4529 ["lbbbrk"] = 10098,
4530 ["rbbrk"] = 10099,
4531 ["lobrk"] = 10214,
4532 ["LeftDoubleBracket"] = 10214,
4533 ["robrk"] = 10215,
4534 ["RightDoubleBracket"] = 10215,
4535 ["lang"] = 10216,
4536 ["LeftAngleBracket"] = 10216,
4537 ["langle"] = 10216,
4538 ["rang"] = 10217,
4539 ["RightAngleBracket"] = 10217,
4540 ["rangle"] = 10217,
4541 ["Lang"] = 10218,
4542 ["Rang"] = 10219,
4543 ["loang"] = 10220,
4544 ["roang"] = 10221,
4545 ["xlarr"] = 10229,
4546 ["longleftarrow"] = 10229,
4547 ["LongLeftArrow"] = 10229,
4548 ["xrarr"] = 10230,
4549 ["longrightarrow"] = 10230,
4550 ["LongRightArrow"] = 10230,

```

```

4551 ["xharr"] = 10231,
4552 ["longleftrightarrow"] = 10231,
4553 ["LongLeftRightArrow"] = 10231,
4554 ["xlArr"] = 10232,
4555 ["Longleftarrow"] = 10232,
4556 ["DoubleLongLeftArrow"] = 10232,
4557 ["xrArr"] = 10233,
4558 ["Longrightarrow"] = 10233,
4559 ["DoubleLongRightArrow"] = 10233,
4560 ["xhArr"] = 10234,
4561 ["Longleftrightarrow"] = 10234,
4562 ["DoubleLongLeftRightArrow"] = 10234,
4563 ["xmap"] = 10236,
4564 ["longmapsto"] = 10236,
4565 ["dzigrarr"] = 10239,
4566 ["nvlArr"] = 10498,
4567 ["nvrArr"] = 10499,
4568 ["nvHarr"] = 10500,
4569 ["Map"] = 10501,
4570 ["lbarr"] = 10508,
4571 ["rbarr"] = 10509,
4572 ["bkarow"] = 10509,
4573 ["lBarr"] = 10510,
4574 ["rBarr"] = 10511,
4575 ["dbkarow"] = 10511,
4576 ["RBarr"] = 10512,
4577 ["drbkarow"] = 10512,
4578 ["DDottrahd"] = 10513,
4579 ["UpArrowBar"] = 10514,
4580 ["DownArrowBar"] = 10515,
4581 ["Rarrtl"] = 10518,
4582 ["latail"] = 10521,
4583 ["ratail"] = 10522,
4584 ["lAtail"] = 10523,
4585 ["rAtail"] = 10524,
4586 ["larrfs"] = 10525,
4587 ["rarrfs"] = 10526,
4588 ["larrbfs"] = 10527,
4589 ["rarrbfs"] = 10528,
4590 ["nwarhk"] = 10531,
4591 ["nearhk"] = 10532,
4592 ["searhk"] = 10533,
4593 ["hksearow"] = 10533,
4594 ["swarhk"] = 10534,
4595 ["hkswarow"] = 10534,
4596 ["nwnear"] = 10535,
4597 ["nesear"] = 10536,

```

```

4598 ["toea"] = 10536,
4599 ["seswar"] = 10537,
4600 ["tosa"] = 10537,
4601 ["swnwar"] = 10538,
4602 ["rarrc"] = 10547,
4603 ["cudarrrr"] = 10549,
4604 ["ldca"] = 10550,
4605 ["rdca"] = 10551,
4606 ["cudarrrl"] = 10552,
4607 ["larrpl"] = 10553,
4608 ["curarrm"] = 10556,
4609 ["cularrp"] = 10557,
4610 ["rarrpl"] = 10565,
4611 ["harrcir"] = 10568,
4612 ["Uarroccir"] = 10569,
4613 ["lurdshar"] = 10570,
4614 ["ldrushar"] = 10571,
4615 ["LeftRightVector"] = 10574,
4616 ["RightUpDownVector"] = 10575,
4617 ["DownLeftRightVector"] = 10576,
4618 ["LeftUpDownVector"] = 10577,
4619 ["LeftVectorBar"] = 10578,
4620 ["RightVectorBar"] = 10579,
4621 ["RightUpVectorBar"] = 10580,
4622 ["RightDownVectorBar"] = 10581,
4623 ["DownLeftVectorBar"] = 10582,
4624 ["DownRightVectorBar"] = 10583,
4625 ["LeftUpVectorBar"] = 10584,
4626 ["LeftDownVectorBar"] = 10585,
4627 ["LeftTeeVector"] = 10586,
4628 ["RightTeeVector"] = 10587,
4629 ["RightUpTeeVector"] = 10588,
4630 ["RightDownTeeVector"] = 10589,
4631 ["DownLeftTeeVector"] = 10590,
4632 ["DownRightTeeVector"] = 10591,
4633 ["LeftUpTeeVector"] = 10592,
4634 ["LeftDownTeeVector"] = 10593,
4635 ["lHar"] = 10594,
4636 ["uHar"] = 10595,
4637 ["rHar"] = 10596,
4638 ["dHar"] = 10597,
4639 ["luruhar"] = 10598,
4640 ["ldrdhar"] = 10599,
4641 ["ruluhar"] = 10600,
4642 ["rdldhar"] = 10601,
4643 ["lharul"] = 10602,
4644 ["llhard"] = 10603,

```



```

4645 ["rharul"] = 10604,
4646 ["lrhard"] = 10605,
4647 ["udhar"] = 10606,
4648 ["UpEquilibrium"] = 10606,
4649 ["duhar"] = 10607,
4650 ["ReverseUpEquilibrium"] = 10607,
4651 ["RoundImplies"] = 10608,
4652 ["erarr"] = 10609,
4653 ["simrarr"] = 10610,
4654 ["larrsim"] = 10611,
4655 ["rarrsim"] = 10612,
4656 ["rarrap"] = 10613,
4657 ["ltlarr"] = 10614,
4658 ["gtrarr"] = 10616,
4659 ["subrarr"] = 10617,
4660 ["suplarr"] = 10619,
4661 ["lfisht"] = 10620,
4662 ["rfisht"] = 10621,
4663 ["ufisht"] = 10622,
4664 ["dfisht"] = 10623,
4665 ["lopar"] = 10629,
4666 ["ropar"] = 10630,
4667 ["lbrke"] = 10635,
4668 ["rbrke"] = 10636,
4669 ["lbrkslu"] = 10637,
4670 ["rbrksld"] = 10638,
4671 ["lbrksld"] = 10639,
4672 ["rbrkslu"] = 10640,
4673 ["langd"] = 10641,
4674 ["rangd"] = 10642,
4675 ["lparlt"] = 10643,
4676 ["rpargt"] = 10644,
4677 ["gtlPar"] = 10645,
4678 ["ltrPar"] = 10646,
4679 ["vzigzag"] = 10650,
4680 ["vangrt"] = 10652,
4681 ["angrtvbd"] = 10653,
4682 ["ange"] = 10660,
4683 ["range"] = 10661,
4684 ["dwangle"] = 10662,
4685 ["uwangle"] = 10663,
4686 ["angmsdaa"] = 10664,
4687 ["angmsdab"] = 10665,
4688 ["angmsdac"] = 10666,
4689 ["angmsdad"] = 10667,
4690 ["angmsdae"] = 10668,
4691 ["angmsdaf"] = 10669,

```

```

4692 ["angmsdag"] = 10670,
4693 ["angmsdah"] = 10671,
4694 ["bemptyv"] = 10672,
4695 ["demptyv"] = 10673,
4696 ["cemptyv"] = 10674,
4697 ["raemptyv"] = 10675,
4698 ["laemptyv"] = 10676,
4699 ["ohbar"] = 10677,
4700 ["omid"] = 10678,
4701 ["opar"] = 10679,
4702 ["operp"] = 10681,
4703 ["olcross"] = 10683,
4704 ["odsold"] = 10684,
4705 ["olcir"] = 10686,
4706 ["ofcir"] = 10687,
4707 ["olt"] = 10688,
4708 ["ogt"] = 10689,
4709 ["cirscir"] = 10690,
4710 ["cirE"] = 10691,
4711 ["solb"] = 10692,
4712 ["bsolb"] = 10693,
4713 ["boxbox"] = 10697,
4714 ["trish"] = 10701,
4715 ["rtriltri"] = 10702,
4716 ["LeftTriangleBar"] = 10703,
4717 ["RightTriangleBar"] = 10704,
4718 ["race"] = 10714,
4719 ["iinfin"] = 10716,
4720 ["infintie"] = 10717,
4721 ["nvinfin"] = 10718,
4722 ["eparsl"] = 10723,
4723 ["smeparsl"] = 10724,
4724 ["eqvparsl"] = 10725,
4725 ["lozf"] = 10731,
4726 ["blacklozenge"] = 10731,
4727 ["RuleDelayed"] = 10740,
4728 ["dsol"] = 10742,
4729 ["xodot"] = 10752,
4730 ["bigodot"] = 10752,
4731 ["xoplus"] = 10753,
4732 ["bigoplus"] = 10753,
4733 ["xotime"] = 10754,
4734 ["bigotimes"] = 10754,
4735 ["xuplus"] = 10756,
4736 ["biguplus"] = 10756,
4737 ["xsqcup"] = 10758,
4738 ["bigsqcup"] = 10758,

```

```

4739 ["qint"] = 10764,
4740 ["iiiint"] = 10764,
4741 ["fpartint"] = 10765,
4742 ["cirfnint"] = 10768,
4743 ["awint"] = 10769,
4744 ["rppolint"] = 10770,
4745 ["scpolint"] = 10771,
4746 ["npolint"] = 10772,
4747 ["pointint"] = 10773,
4748 ["quatint"] = 10774,
4749 ["intlarhk"] = 10775,
4750 ["pluscir"] = 10786,
4751 ["plusacir"] = 10787,
4752 ["simplus"] = 10788,
4753 ["plusdu"] = 10789,
4754 ["plussim"] = 10790,
4755 ["plustwo"] = 10791,
4756 ["mcomma"] = 10793,
4757 ["minusdu"] = 10794,
4758 ["loplus"] = 10797,
4759 ["roplus"] = 10798,
4760 ["Cross"] = 10799,
4761 ["timesd"] = 10800,
4762 ["timesbar"] = 10801,
4763 ["smashp"] = 10803,
4764 ["lotimes"] = 10804,
4765 ["rotimes"] = 10805,
4766 ["otimesas"] = 10806,
4767 ["Otimes"] = 10807,
4768 ["odiv"] = 10808,
4769 ["triplus"] = 10809,
4770 ["triminus"] = 10810,
4771 ["tritime"] = 10811,
4772 ["iproduct"] = 10812,
4773 ["intprod"] = 10812,
4774 ["amalg"] = 10815,
4775 ["capdot"] = 10816,
4776 ["ncup"] = 10818,
4777 ["ncap"] = 10819,
4778 ["capand"] = 10820,
4779 ["cupor"] = 10821,
4780 ["cupcap"] = 10822,
4781 ["capcup"] = 10823,
4782 ["cupbrcap"] = 10824,
4783 ["capbrcup"] = 10825,
4784 ["cupcup"] = 10826,
4785 ["capcap"] = 10827,

```

4786 ["ccups"] = 10828,  
 4787 ["ccaps"] = 10829,  
 4788 ["ccupssm"] = 10832,  
 4789 ["And"] = 10835,  
 4790 ["Or"] = 10836,  
 4791 ["andand"] = 10837,  
 4792 ["oror"] = 10838,  
 4793 ["orslope"] = 10839,  
 4794 ["andslope"] = 10840,  
 4795 ["andv"] = 10842,  
 4796 ["orv"] = 10843,  
 4797 ["andd"] = 10844,  
 4798 ["ord"] = 10845,  
 4799 ["wedbar"] = 10847,  
 4800 ["sdote"] = 10854,  
 4801 ["simdot"] = 10858,  
 4802 ["congdote"] = 10861,  
 4803 ["easter"] = 10862,  
 4804 ["apacir"] = 10863,  
 4805 ["apE"] = 10864,  
 4806 ["eplus"] = 10865,  
 4807 ["pluse"] = 10866,  
 4808 ["Esim"] = 10867,  
 4809 ["Colone"] = 10868,  
 4810 ["Equal"] = 10869,  
 4811 ["eDDot"] = 10871,  
 4812 ["ddotseq"] = 10871,  
 4813 ["equivDD"] = 10872,  
 4814 ["ltcir"] = 10873,  
 4815 ["gtcir"] = 10874,  
 4816 ["ltquest"] = 10875,  
 4817 ["gtquest"] = 10876,  
 4818 ["les"] = 10877,  
 4819 ["LessSlantEqual"] = 10877,  
 4820 ["leqslant"] = 10877,  
 4821 ["ges"] = 10878,  
 4822 ["GreaterSlantEqual"] = 10878,  
 4823 ["geqslant"] = 10878,  
 4824 ["lesdot"] = 10879,  
 4825 ["gesdot"] = 10880,  
 4826 ["lesdoto"] = 10881,  
 4827 ["gesdoto"] = 10882,  
 4828 ["lesdotor"] = 10883,  
 4829 ["gesdoto1"] = 10884,  
 4830 ["lap"] = 10885,  
 4831 ["lessapprox"] = 10885,  
 4832 ["gap"] = 10886,

```

4833 ["gtrapprox"] = 10886,
4834 ["lne"] = 10887,
4835 ["lneq"] = 10887,
4836 ["gne"] = 10888,
4837 ["gneq"] = 10888,
4838 ["lnap"] = 10889,
4839 ["lnapprox"] = 10889,
4840 ["gnap"] = 10890,
4841 ["gnapprox"] = 10890,
4842 ["lEg"] = 10891,
4843 ["lesseqqgtr"] = 10891,
4844 ["gEl"] = 10892,
4845 ["gtreqqless"] = 10892,
4846 ["lsime"] = 10893,
4847 ["gsime"] = 10894,
4848 ["lsimg"] = 10895,
4849 ["gsiml"] = 10896,
4850 ["lgE"] = 10897,
4851 ["glE"] = 10898,
4852 ["lesges"] = 10899,
4853 ["gesles"] = 10900,
4854 ["els"] = 10901,
4855 ["eqslantless"] = 10901,
4856 ["egs"] = 10902,
4857 ["eqslantgtr"] = 10902,
4858 ["elsdot"] = 10903,
4859 ["egsdot"] = 10904,
4860 ["el"] = 10905,
4861 ["eg"] = 10906,
4862 ["siml"] = 10909,
4863 ["simg"] = 10910,
4864 ["simlE"] = 10911,
4865 ["simgE"] = 10912,
4866 ["LessLess"] = 10913,
4867 ["GreaterGreater"] = 10914,
4868 ["glj"] = 10916,
4869 ["gla"] = 10917,
4870 ["ltcc"] = 10918,
4871 ["gtcc"] = 10919,
4872 ["lescc"] = 10920,
4873 ["gescc"] = 10921,
4874 ["smt"] = 10922,
4875 ["lat"] = 10923,
4876 ["smte"] = 10924,
4877 ["late"] = 10925,
4878 ["bumpE"] = 10926,
4879 ["pre"] = 10927,

```

```

4880 ["preceq"] = 10927,
4881 ["PrecedesEqual"] = 10927,
4882 ["sce"] = 10928,
4883 ["succeq"] = 10928,
4884 ["SucceedsEqual"] = 10928,
4885 ["prE"] = 10931,
4886 ["scE"] = 10932,
4887 ["prnE"] = 10933,
4888 ["precneqq"] = 10933,
4889 ["scnE"] = 10934,
4890 ["succneqq"] = 10934,
4891 ["prap"] = 10935,
4892 ["precapprox"] = 10935,
4893 ["scap"] = 10936,
4894 ["succapprox"] = 10936,
4895 ["prnap"] = 10937,
4896 ["precnapprox"] = 10937,
4897 ["scnap"] = 10938,
4898 ["succnapprox"] = 10938,
4899 ["Pr"] = 10939,
4900 ["Sc"] = 10940,
4901 ["subdot"] = 10941,
4902 ["supdot"] = 10942,
4903 ["subplus"] = 10943,
4904 ["supplus"] = 10944,
4905 ["submult"] = 10945,
4906 ["supmult"] = 10946,
4907 ["subedot"] = 10947,
4908 ["supedot"] = 10948,
4909 ["subE"] = 10949,
4910 ["subseteqq"] = 10949,
4911 ["supE"] = 10950,
4912 ["supseteqq"] = 10950,
4913 ["subsim"] = 10951,
4914 ["supsim"] = 10952,
4915 ["subnE"] = 10955,
4916 ["subsetneqq"] = 10955,
4917 ["supnE"] = 10956,
4918 ["supsetneqq"] = 10956,
4919 ["csub"] = 10959,
4920 ["csup"] = 10960,
4921 ["csube"] = 10961,
4922 ["csupe"] = 10962,
4923 ["subsup"] = 10963,
4924 ["supsub"] = 10964,
4925 ["subsub"] = 10965,
4926 ["supsup"] = 10966,

```

```

4927 ["suphsub"] = 10967,
4928 ["supdsub"] = 10968,
4929 ["forkv"] = 10969,
4930 ["topfork"] = 10970,
4931 ["mlcp"] = 10971,
4932 ["Dashv"] = 10980,
4933 ["DoubleLeftTee"] = 10980,
4934 ["Vdashl"] = 10982,
4935 ["Barv"] = 10983,
4936 ["vBar"] = 10984,
4937 ["vBarv"] = 10985,
4938 ["Vbar"] = 10987,
4939 ["Not"] = 10988,
4940 ["bNot"] = 10989,
4941 ["rnmid"] = 10990,
4942 ["cirmid"] = 10991,
4943 ["midcir"] = 10992,
4944 ["topcir"] = 10993,
4945 ["nhpar"] = 10994,
4946 ["parsim"] = 10995,
4947 ["parsl"] = 11005,
4948 ["fflig"] = 64256,
4949 ["filig"] = 64257,
4950 ["fllig"] = 64258,
4951 ["ffilig"] = 64259,
4952 ["ffllig"] = 64260,
4953 ["Ascr"] = 119964,
4954 ["Cscr"] = 119966,
4955 ["Dscr"] = 119967,
4956 ["Gscr"] = 119970,
4957 ["Jscr"] = 119973,
4958 ["Kscr"] = 119974,
4959 ["Nscr"] = 119977,
4960 ["Oscr"] = 119978,
4961 ["Pscr"] = 119979,
4962 ["Qscr"] = 119980,
4963 ["Sscr"] = 119982,
4964 ["Tscr"] = 119983,
4965 ["Uscr"] = 119984,
4966 ["Vscr"] = 119985,
4967 ["Wscr"] = 119986,
4968 ["Xscr"] = 119987,
4969 ["Yscr"] = 119988,
4970 ["Zscr"] = 119989,
4971 ["ascr"] = 119990,
4972 ["bscr"] = 119991,
4973 ["cscr"] = 119992,

```

```

4974 ["dscr"] = 119993,
4975 ["fscr"] = 119995,
4976 ["hscr"] = 119997,
4977 ["iscr"] = 119998,
4978 ["jscr"] = 119999,
4979 ["kscr"] = 120000,
4980 ["lscr"] = 120001,
4981 ["mscr"] = 120002,
4982 ["nscr"] = 120003,
4983 ["pscr"] = 120005,
4984 ["qscr"] = 120006,
4985 ["rscr"] = 120007,
4986 ["sscr"] = 120008,
4987 ["tscr"] = 120009,
4988 ["uscr"] = 120010,
4989 ["vscr"] = 120011,
4990 ["wscr"] = 120012,
4991 ["xscr"] = 120013,
4992 ["yscr"] = 120014,
4993 ["zscr"] = 120015,
4994 ["Afr"] = 120068,
4995 ["Bfr"] = 120069,
4996 ["Dfr"] = 120071,
4997 ["Efr"] = 120072,
4998 ["Ffr"] = 120073,
4999 ["Gfr"] = 120074,
5000 ["Jfr"] = 120077,
5001 ["Kfr"] = 120078,
5002 ["Lfr"] = 120079,
5003 ["Mfr"] = 120080,
5004 ["Nfr"] = 120081,
5005 ["Ofr"] = 120082,
5006 ["Pfr"] = 120083,
5007 ["Qfr"] = 120084,
5008 ["Sfr"] = 120086,
5009 ["Tfr"] = 120087,
5010 ["Ufr"] = 120088,
5011 ["Vfr"] = 120089,
5012 ["Wfr"] = 120090,
5013 ["Xfr"] = 120091,
5014 ["Yfr"] = 120092,
5015 ["afr"] = 120094,
5016 ["bfr"] = 120095,
5017 ["cfr"] = 120096,
5018 ["dfr"] = 120097,
5019 ["efr"] = 120098,
5020 ["ffr"] = 120099,

```



```

5021 ["gfr"] = 120100,
5022 ["hfr"] = 120101,
5023 ["ifr"] = 120102,
5024 ["jfr"] = 120103,
5025 ["kfr"] = 120104,
5026 ["lfr"] = 120105,
5027 ["mfr"] = 120106,
5028 ["nfr"] = 120107,
5029 ["ofr"] = 120108,
5030 ["pfr"] = 120109,
5031 ["qfr"] = 120110,
5032 ["rfr"] = 120111,
5033 ["sfr"] = 120112,
5034 ["tfr"] = 120113,
5035 ["ufr"] = 120114,
5036 ["vfr"] = 120115,
5037 ["wfr"] = 120116,
5038 ["xfr"] = 120117,
5039 ["yfr"] = 120118,
5040 ["zfr"] = 120119,
5041 ["Aopf"] = 120120,
5042 ["Bopf"] = 120121,
5043 ["Dopf"] = 120123,
5044 ["Eopf"] = 120124,
5045 ["Fopf"] = 120125,
5046 ["Gopf"] = 120126,
5047 ["Iopf"] = 120128,
5048 ["Jopf"] = 120129,
5049 ["Kopf"] = 120130,
5050 ["Lopf"] = 120131,
5051 ["Mopf"] = 120132,
5052 ["Oopf"] = 120134,
5053 ["Sopf"] = 120138,
5054 ["Topf"] = 120139,
5055 ["Uopf"] = 120140,
5056 ["Vopf"] = 120141,
5057 ["Wopf"] = 120142,
5058 ["Xopf"] = 120143,
5059 ["Yopf"] = 120144,
5060 ["aopf"] = 120146,
5061 ["bopf"] = 120147,
5062 ["copf"] = 120148,
5063 ["dopf"] = 120149,
5064 ["eopf"] = 120150,
5065 ["fopf"] = 120151,
5066 ["gopf"] = 120152,
5067 ["hopf"] = 120153,

```

```

5068 ["iopf"] = 120154,
5069 ["jopf"] = 120155,
5070 ["kopf"] = 120156,
5071 ["lopf"] = 120157,
5072 ["mopf"] = 120158,
5073 ["nopf"] = 120159,
5074 ["oopf"] = 120160,
5075 ["popf"] = 120161,
5076 ["qopf"] = 120162,
5077 ["ropf"] = 120163,
5078 ["sopf"] = 120164,
5079 ["topf"] = 120165,
5080 ["uopf"] = 120166,
5081 ["vopf"] = 120167,
5082 ["wopf"] = 120168,
5083 ["xopf"] = 120169,
5084 ["yopf"] = 120170,
5085 ["zopf"] = 120171,
5086 }

```

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

```

5087 function entities.dec_entity(s)
5088     return unicode.utf8.char(tonumber(s))
5089 end

```

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

```

5090 function entities.hex_entity(s)
5091     return unicode.utf8.char(tonumber("0x"..s))
5092 end

```

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

```

5093 function entities.char_entity(s)
5094     local n = character_entities[s]
5095     if n == nil then
5096         return "&" .. s .. ";"
5097     end
5098     return unicode.utf8.char(n)
5099 end

```

### 3.1.3 Plain T<sub>E</sub>X Writer

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`,

`lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark Lua module.

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

```
5100 M.writer = {}
```

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

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `writer->member`. All member variables are immutable unless explicitly stated otherwise.

```
5101 function M.writer.new(options)
5102   local self = {}
```

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

```
5103   self.options = options
```

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

```
5104   local slice_specifiers = {}
5105   for specifier in options.slice:gmatch("[^%s]+") do
5106     table.insert(slice_specifiers, specifier)
5107   end
5108
5109   if #slice_specifiers == 2 then
5110     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5111     local slice_begin_type = self.slice_begin:sub(1, 1)
5112     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5113       self.slice_begin = "^" .. self.slice_begin
5114     end
5115     local slice_end_type = self.slice_end:sub(1, 1)
5116     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5117       self.slice_end = "$" .. self.slice_end
5118     end
5119   elseif #slice_specifiers == 1 then
5120     self.slice_begin = "^" .. slice_specifiers[1]
5121     self.slice_end = "$" .. slice_specifiers[1]
5122   end
5123
5124   self.slice_begin_type = self.slice_begin:sub(1, 1)
5125   self.slice_begin_identifier = self.slice_begin:sub(2) or ""
```

```

5126 self.slice_end_type = self.slice_end:sub(1, 1)
5127 self.slice_end_identifier = self.slice_end:sub(2) or ""
5128
5129 if self.slice_begin == "^" and self.slice_end ~= "^" then
5130     self.is_writing = true
5131 else
5132     self.is_writing = false
5133 end

```

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

```

5134 self.suffix = ".tex"

```

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

```

5135 self.space = " "

```

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

```

5136 self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

5137 function self.plain(s)
5138     return s
5139 end

```

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

```

5140 function self.paragraph(s)
5141     if not self.is_writing then return "" end
5142     return s
5143 end

```

Define **writer->pack** as a function that will take the filename **name** of the output file prepared by the reader and transform it to the output format.

```

5144 function self.pack(name)
5145     return [[\input{}} .. name .. [{}\relax]]
5146 end

```

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

```

5147 function self.interblocksep()
5148     if not self.is_writing then return "" end
5149     return "\\markdownRendererInterblockSeparator\n{}"
5150 end

```

Define **writer->hard\_line\_break** as the output format of a forced line break.

```

5151 self.hard_line_break = "\\markdownRendererHardLineBreak\n{}"

```

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

```

5152 self.ellipsis = "\\markdownRendererEllipsis{}"

```

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

```
5153 function self.thematic_break()
5154     if not self.is_writing then return "" end
5155     return "\\markdownRendererThematicBreak{}"
5156 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.

```
5157 self.escaped_uri_chars = {
5158     ["{"] = "\\markdownRendererLeftBrace{}",
5159     ["}"] = "\\markdownRendererRightBrace{}",
5160     ["\\"] = "\\markdownRendererBackslash{}",
5161 }
5162 self.escaped_minimal_strings = {
5163     ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
5164     ["☒"] = "\\markdownRendererTickedBox{}",
5165     ["◻"] = "\\markdownRendererHalfTickedBox{}",
5166     ["□"] = "\\markdownRendererUntickedBox{}",
5167     [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{}",
5168 }
```

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

```
5169 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5170 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

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

```
5171 self.escaped_chars = {
5172     ["{"] = "\\markdownRendererLeftBrace{}",
5173     ["}"] = "\\markdownRendererRightBrace{}",
5174     ["%"] = "\\markdownRendererPercentSign{}",
5175     ["\\"] = "\\markdownRendererBackslash{}",
5176     ["#"] = "\\markdownRendererHash{}",
5177     ["$"] = "\\markdownRendererDollarSign{}",
5178     ["&"] = "\\markdownRendererAmpersand{}",
5179     ["_"] = "\\markdownRendererUnderscore{}",
5180     ["^"] = "\\markdownRendererCircumflex{}",
5181     ["~"] = "\\markdownRendererTilde{}",
5182     ["|"] = "\\markdownRendererPipe{}",
5183     [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{}",
5184 }
```

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

```

5185 local escape_typographic_text = util.escaper(
5186     self.escaped_chars, self.escaped_strings)
5187 local escape_programmatic_text = util.escaper(
5188     self.escaped_uri_chars, self.escaped_minimal_strings)
5189 local escape_minimal = util.escaper(
5190     {}, 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.

```

5191 self.escape = escape_typographic_text
5192 self.math = escape_minimal
5193 if options.hybrid then
5194     self.identifier = escape_minimal
5195     self.string = escape_minimal
5196     self.uri = escape_minimal
5197 else
5198     self.identifier = escape_programmatic_text
5199     self.string = escape_typographic_text
5200     self.uri = escape_programmatic_text
5201 end

```

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

```

5202 function self.code(s, attributes)
5203     local buf = {}
5204     if attributes ~= nil then
5205         table.insert(buf,
5206             "\\markdownRenderCodeSpanAttributeContextBegin\n")
5207         table.insert(buf, self.attributes(attributes))
5208     end
5209     table.insert(buf,
5210         {"\\markdownRenderCodeSpan{", self.escape(s), "}"})
5211     if attributes ~= nil then
5212         table.insert(buf,
5213             "\n\\markdownRenderCodeSpanAttributeContextEnd ")
5214     end
5215     return buf
5216 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.

```

5217 function self.link(lab, src, tit, attributes)
5218     local buf = {}
5219     if attributes ~= nil then
5220         table.insert(buf,
5221             "\\markdownRendererLinkAttributeContextBegin\n")
5222         table.insert(buf, self.attributes(attributes))
5223     end
5224     table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ",
5225         {" ,self.escape(src),"} ",
5226         {" ,self.uri(src),"} ",
5227         {" ,self.string(tit or ""),"} "})
5228     if attributes ~= nil then
5229         table.insert(buf,
5230             "\n\\markdownRendererLinkAttributeContextEnd ")
5231     end
5232     return buf
5233 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.

```

5234 function self.image(lab, src, tit, attributes)
5235     local buf = {}
5236     if attributes ~= nil then
5237         table.insert(buf,
5238             "\\markdownRendererImageAttributeContextBegin\n")
5239         table.insert(buf, self.attributes(attributes))
5240     end
5241     table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
5242         {" ,self.string(src),"} ",
5243         {" ,self.uri(src),"} ",
5244         {" ,self.string(tit or ""),"} "})
5245     if attributes ~= nil then
5246         table.insert(buf,
5247             "\n\\markdownRendererImageAttributeContextEnd ")
5248     end
5249     return buf
5250 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.

```

5251 function self.bulletlist(items,tight)
5252     if not self.is_writing then return "" end

```

```

5253     local buffer = {}
5254     for _,item in ipairs(items) do
5255         buffer[#buffer + 1] = self.bulletitem(item)
5256     end
5257     local contents = util.intersperse(buffer,"\n")
5258     if tight and options.tightLists then
5259         return {"\\markdownRendererUlBeginTight\n",contents,
5260             "\n\\markdownRendererUlEndTight "}
5261     else
5262         return {"\\markdownRendererUlBegin\n",contents,
5263             "\n\\markdownRendererUlEnd "}
5264     end
5265 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.

```

5266 function self.bulletitem(s)
5267     return {"\\markdownRendererUlItem ",s,
5268         "\\markdownRendererUlItemEnd "}
5269 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.

```

5270 function self.orderedlist(items,tight,startnum)
5271     if not self.is_writing then return "" end
5272     local buffer = {}
5273     local num = startnum
5274     for _,item in ipairs(items) do
5275         buffer[#buffer + 1] = self.ordereditem(item,num)
5276         if num ~= nil then
5277             num = num + 1
5278         end
5279     end
5280     local contents = util.intersperse(buffer,"\n")
5281     if tight and options.tightLists then
5282         return {"\\markdownRendererOlBeginTight\n",contents,
5283             "\n\\markdownRendererOlEndTight "}
5284     else
5285         return {"\\markdownRendererOlBegin\n",contents,
5286             "\n\\markdownRendererOlEnd "}
5287     end
5288 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.



```

5289 function self.ordereditem(s,num)
5290     if num ~= nil then
5291         return {"\\markdownRendererOliItemWithNumber{" ,num,"} ",s,
5292             "\\markdownRendererOliItemEnd "}
5293     else
5294         return {"\\markdownRendererOliItem " ,s,
5295             "\\markdownRendererOliItemEnd "}
5296     end
5297 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.

```

5298 function self.inline_html_comment(contents)
5299     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5300 end

```

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

```

5301 function self.block_html_comment(contents)
5302     if not self.is_writing then return "" end
5303     return {"\\markdownRendererBlockHtmlCommentBegin\\n",contents,
5304         "\\n\\markdownRendererBlockHtmlCommentEnd "}
5305 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.

```

5306 function self.inline_html_tag(contents)
5307     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5308 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.

```

5309 function self.block_html_element(s)
5310     if not self.is_writing then return "" end
5311     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5312     return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5313 end

```

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

```

5314 function self.emphasis(s)
5315     return {"\\markdownRendererEmphasis{" ,s,"}"}
5316 end

```

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

```
5317 function self.checkbox(f)
5318   if f == 1.0 then
5319     return "☒ "
5320   elseif f == 0.0 then
5321     return "☐ "
5322   else
5323     return "◻ "
5324   end
5325 end
```

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

```
5326 function self.strong(s)
5327   return {"\\markdownRendererStrongEmphasis{" ,s,""} }
5328 end
```

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

```
5329 function self.blockquote(s)
5330   if #util.rope_to_string(s) == 0 then return "" end
5331   return {"\\markdownRendererBlockQuoteBegin\\n",s,
5332     "\\n\\markdownRendererBlockQuoteEnd "}
5333 end
```

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

```
5334 function self.verbatim(s)
5335   if not self.is_writing then return "" end
5336   s = s:gsub("\\n$", "")
5337   local name = util.cache_verbatim(options.cacheDir, s)
5338   return {"\\markdownRendererInputVerbatim{" ,name,""} }
5339 end
```

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

```
5340 function self.document(d)
5341   local buf = {"\\markdownRendererDocumentBegin\\n", d}
5342
5343   -- pop all attributes
5344   table.insert(buf, self.pop_attributes())
5345
5346   table.insert(buf, "\\markdownRendererDocumentEnd")
5347
5348   return buf
5349 end
```

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

```

5350  function self.attributes(attr)
5351      local buf = {}
5352
5353      table.sort(attr)
5354      local seen = {}
5355      local key, value
5356      for i = 1, #attr do
5357          if seen[attr[i]] ~= nil then
5358              goto continue -- prevent duplicate attributes
5359          else
5360              seen[attr[i]] = true
5361          end
5362          if attr[i]:sub(1, 1) == "#" then
5363              table.insert(buf, {"\\markdownRenderAttributeIdentifier{",
5364                              attr[i]:sub(2), "}"}))
5365          elseif attr[i]:sub(1, 1) == "." then
5366              table.insert(buf, {"\\markdownRenderAttributeClassName{",
5367                              attr[i]:sub(2), "}"}))
5368          else
5369              key, value = attr[i]:match("(^[^= ]+)%s*=%s*(.*)")
5370              table.insert(buf, {"\\markdownRenderAttributeKeyValue{",
5371                              key, "}{" , value, "}"}))
5372          end
5373          ::continue::
5374      end
5375
5376      return buf
5377  end

```

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

```

5378  self.active_attributes = {}

```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```

5379  local function apply_attributes()
5380      local buf = {}
5381      for i = 1, #self.active_attributes do
5382          local start_output = self.active_attributes[i][3]
5383          if start_output ~= nil then
5384              table.insert(buf, start_output)
5385          end
5386      end
5387      return buf

```

```

5388 end
5389
5390 local function tear_down_attributes()
5391     local buf = {}
5392     for i = #self.active_attributes, 1, -1 do
5393         local end_output = self.active_attributes[i][4]
5394         if end_output ~= nil then
5395             table.insert(buf, end_output)
5396         end
5397     end
5398     return buf
5399 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.

```

5400 function self.push_attributes(attribute_type, attributes,
5401                             start_output, end_output)
5402     -- index attributes in a hash table for easy lookup
5403     attributes = attributes or {}
5404     for i = 1, #attributes do
5405         attributes[attributes[i]] = true
5406     end
5407
5408     local buf = {}
5409     -- handle slicing
5410     if attributes["#" .. self.slice_end_identifier] ~= nil and
5411        self.slice_end_type == "^" then
5412         if self.is_writing then
5413             table.insert(buf, tear_down_attributes())
5414         end
5415         self.is_writing = false
5416     end
5417     if attributes["#" .. self.slice_begin_identifier] ~= nil and
5418        self.slice_begin_type == "^" then
5419         self.is_writing = true
5420         table.insert(buf, apply_attributes())
5421         self.is_writing = true
5422     end
5423     if self.is_writing and start_output ~= nil then
5424         table.insert(buf, start_output)
5425     end
5426     table.insert(self.active_attributes,
5427                 {attribute_type, attributes,
5428                  start_output, end_output})

```

```

5429     return buf
5430 end
5431

```

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`).

```

5432 function self.pop_attributes(attribute_type)
5433     local buf = {}
5434     -- pop attributes until we find attributes of correct type
5435     -- or until no attributes remain
5436     local current_attribute_type = false
5437     while current_attribute_type ~= attribute_type and
5438         #self.active_attributes > 0 do
5439         local attributes, _, end_output
5440         current_attribute_type, attributes, _, end_output = table.unpack(
5441             self.active_attributes[#self.active_attributes])
5442         if self.is_writing and end_output ~= nil then
5443             table.insert(buf, end_output)
5444         end
5445         table.remove(self.active_attributes, #self.active_attributes)
5446         -- handle slicing
5447         if attributes["#" .. self.slice_end_identifier] ~= nil
5448             and self.slice_end_type == "$" then
5449             if self.is_writing then
5450                 table.insert(buf, tear_down_attributes())
5451             end
5452             self.is_writing = false
5453         end
5454         if attributes["#" .. self.slice_begin_identifier] ~= nil and
5455             self.slice_begin_type == "$" then
5456             self.is_writing = true
5457             table.insert(buf, apply_attributes())
5458         end
5459     end
5460     return buf
5461 end

```

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

```

5462     local current_heading_level = 0
5463     function self.heading(s, level, attributes)
5464         local buf = {}
5465

```

```

5466 -- push empty attributes for implied sections
5467 while current_heading_level < level - 1 do
5468     table.insert(buf,
5469         self.push_attributes("heading",
5470             nil,
5471             "\\markdownRendererSectionBegin\n",
5472             "\n\\markdownRendererSectionEnd "))
5473     current_heading_level = current_heading_level + 1
5474 end
5475
5476 -- pop attributes for sections that have ended
5477 while current_heading_level >= level do
5478     table.insert(buf, self.pop_attributes("heading"))
5479     current_heading_level = current_heading_level - 1
5480 end
5481
5482 -- push attributes for the new section
5483 local start_output = {}
5484 local end_output = {}
5485 table.insert(start_output, "\\markdownRendererSectionBegin\n")
5486 if options.headerAttributes and attributes ~= nil and #attributes > 0 then
5487     table.insert(start_output,
5488         "\\markdownRendererHeaderAttributeContextBegin\n")
5489     table.insert(start_output, self.attributes(attributes))
5490     table.insert(end_output,
5491         "\n\\markdownRendererHeaderAttributeContextEnd ")
5492 end
5493 table.insert(end_output, "\n\\markdownRendererSectionEnd ")
5494
5495 table.insert(buf, self.push_attributes("heading",
5496     attributes,
5497     start_output,
5498     end_output))
5499 current_heading_level = current_heading_level + 1
5500 assert(current_heading_level == level)
5501
5502 -- produce the renderer
5503 local cmd
5504 level = level + options.shiftHeadings
5505 if level <= 1 then
5506     cmd = "\\markdownRendererHeadingOne"
5507 elseif level == 2 then
5508     cmd = "\\markdownRendererHeadingTwo"
5509 elseif level == 3 then
5510     cmd = "\\markdownRendererHeadingThree"
5511 elseif level == 4 then
5512     cmd = "\\markdownRendererHeadingFour"

```

```

5513     elseif level == 5 then
5514         cmd = "\\markdownRendererHeadingFive"
5515     elseif level >= 6 then
5516         cmd = "\\markdownRendererHeadingSix"
5517     else
5518         cmd = ""
5519     end
5520     if self.is_writing then
5521         table.insert(buf, {cmd, "{", s, "}"})
5522     end
5523
5524     return buf
5525 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.

```

5526 function self.get_state()
5527     return {
5528         is_writing=self.is_writing,
5529         active_attributes={table.unpack(self.active_attributes)},
5530     }
5531 end

```

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

```

5532 function self.set_state(s)
5533     local previous_state = self.get_state()
5534     for key, value in pairs(s) do
5535         self[key] = value
5536     end
5537     return previous_state
5538 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`.

```

5539 function self.defer_call(f)
5540     local previous_state = self.get_state()
5541     return function(...)
5542         local state = self.set_state(previous_state)
5543         local return_value = f(...)
5544         self.set_state(state)
5545         return return_value
5546     end
5547 end
5548
5549 return self
5550 end

```

### 3.1.4 Parsers

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

```
5551 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
5552 parsers.percent = P("%")
5553 parsers.at = P("@")
5554 parsers.comma = P(",")
5555 parsers.asterisk = P("*")
5556 parsers.dash = P("-")
5557 parsers.plus = P("+")
5558 parsers.underscore = P("_")
5559 parsers.period = P(".")
5560 parsers.hash = P("#")
5561 parsers.dollar = P("$")
5562 parsers.ampersand = P("&")
5563 parsers.backtick = P("`")
5564 parsers.less = P("<")
5565 parsers.more = P(">")
5566 parsers.space = P(" ")
5567 parsers.squote = P("'")
5568 parsers.dquote = P('"')
5569 parsers.lparent = P("(")
5570 parsers.rparent = P(")")
5571 parsers.lbracket = P("[")
5572 parsers.rbracket = P("]")
5573 parsers.lbrace = P("{")
5574 parsers.rbrace = P("}")
5575 parsers.circumflex = P("^")
5576 parsers.slash = P("/")
5577 parsers.equal = P("=")
5578 parsers.colon = P(":")
5579 parsers.semicolon = P(";")
5580 parsers.exclamation = P("!")
5581 parsers.pipe = P("|")
5582 parsers.tilde = P("~")
5583 parsers.backslash = P("\\")
5584 parsers.tab = P("\t")
5585 parsers.newline = P("\n")
5586 parsers.tightblocksep = P("\001")
5587
5588 parsers.digit = R("09")
5589 parsers.hexdigit = R("09", "af", "AF")
5590 parsers.letter = R("AZ", "az")
```



```

5591 parsers.alphanumeric      = R("AZ","az","09")
5592 parsers.keyword           = parsers.letter
5593                             * parsers.alphanumeric^0
5594 parsers.internal_punctuation = S(":,;,.?")
5595
5596 parsers.doubleasterisks     = P("**")
5597 parsers.doubleunderscores   = P("__")
5598 parsers.doubletildes        = P("~~")
5599 parsers.fourspace          = P("    ")
5600
5601 parsers.any                 = P(1)
5602 parsers.succeed             = P(true)
5603 parsers.fail                = P(false)
5604
5605 parsers.escapable           = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
5606 parsers.anyescaped         = parsers.backslash / "\"" * parsers.escapable
5607                             + parsers.any
5608
5609 parsers.spacechar           = S("\t ")
5610 parsers.spacing             = S(" \n\r\t")
5611 parsers.nonspacechar        = parsers.any - parsers.spacing
5612 parsers.optionalspace       = parsers.spacechar^0
5613
5614 parsers.normalchar          = parsers.any - (V("SpecialChar")
5615                                             + parsers.spacing
5616                                             + parsers.tightblocksep)
5617 parsers.eof                 = -parsers.any
5618 parsers.nonindentspace      = parsers.space^3 * - parsers.spacechar
5619 parsers.indent              = parsers.space^3 * parsers.tab
5620                             + parsers.fourspace / "\""
5621 parsers.linechar            = P(1 - parsers.newline)
5622
5623 parsers.blankline           = parsers.optionalspace
5624                             * parsers.newline / "\n"
5625 parsers.blanklines          = parsers.blankline^0
5626 parsers.skipblanklines      = (parsers.optionalspace * parsers.newline)^0
5627 parsers.indentedline        = parsers.indent / "\""
5628                             * C(parsers.linechar^1 * parsers.newline^-
5629                                1)
5629 parsers.optionallyindentedline = parsers.indent^-1 / "\""
5630                             * C(parsers.linechar^1 * parsers.newline^-
5631                                1)
5631 parsers.sp                  = parsers.spacing^0
5632 parsers.spnl                = parsers.optionalspace
5633                             * (parsers.newline * parsers.optionalspace)^-
5634                             1
5634 parsers.line                = parsers.linechar^0 * parsers.newline

```

```
5635 parsers.nonemptyline = parsers.line - parsers.blankline
```

The `parsers.commented_line1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 6.

```
5636 parsers.commented_line_letter = parsers.linechar
5637                               + parsers.newline
5638                               - parsers.backslash
5639                               - parsers.percent
5640 parsers.commented_line = Cg(Cc(""), "backslashes")
5641 * ((#(parsers.commented_line_letter
5642     - parsers.newline)
5643    * Cb("backslashes")
5644    * Cs(parsers.commented_line_letter
5645        - parsers.newline)1 -- initial
5646    * Cg(Cc(""), "backslashes"))
5647 + #(parsers.backslash * parsers.backslash
5648    * Cg((parsers.backslash -- even backslash
5649        * parsers.backslash)1, "backslashes")
5650 + (parsers.backslash
5651    * (#parsers.percent
5652      * Cb("backslashes")
5653      / function(backslashes)
5654        return string.rep("\\", #backslashes / 2)
5655      end
5656      * C(parsers.percent)
5657      + #parsers.commented_line_letter
5658      * Cb("backslashes")
5659      * Cc("\\")
5660      * C(parsers.commented_line_letter))
5661    * Cg(Cc(""), "backslashes"))0
5662 * (#parsers.percent
5663    * Cb("backslashes")
5664    / function(backslashes)
5665      return string.rep("\\", #backslashes / 2)
5666    end
5667 * ((parsers.percent -- comment
5668     * parsers.line
5669     * #parsers.blankline) -- blank line
5670    / "\n"
5671    + parsers.percent -- comment
5672    * parsers.line
5673    * parsers.optionalspace) -- leading tabs and space
5674 + #(parsers.newline)
5675    * Cb("backslashes")
5676    * C(parsers.newline))
5677
5678 parsers.chunk = parsers.line * (parsers.optionallyindentedline
```



```

5679                                     - parsers.blankline)^0
5680
5681 parsers.attribute_key_char      = parsers.alphanumeric + S("_-")
5682 parsers.attribute_key          = (parsers.attribute_key_char
5683                                - parsers.dash - parsers.digit)
5684                                * parsers.attribute_key_char^0
5685 parsers.attribute_value         = ( (parsers.dquote / "\"")
5686                                * (parsers.anyescaped - parsers.dquote)^0
5687                                * (parsers.dquote / "\""))
5688                                + ( parsers.anyescaped - parsers.dquote - parsers.rbra
5689                                - parsers.space)^0
5690
5691 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
5692                  + C((parsers.hash + parsers.period)
5693                     * parsers.attribute_key)
5694                  + Cs( parsers.attribute_key
5695                     * parsers.optionalspace * parsers.equal * parsers.optionalspace
5696                     * parsers.attribute_value)
5697 parsers.attributes = parsers.lbrace
5698                  * parsers.optionalspace
5699                  * parsers.attribute
5700                  * (parsers.spacechar^1
5701                  * parsers.attribute)^0
5702                  * parsers.optionalspace
5703                  * parsers.rbrace
5704
5705
5706 parsers.raw_attribute = parsers.lbrace
5707                  * parsers.optionalspace
5708                  * parsers.equal
5709                  * C(parsers.attribute_key)
5710                  * parsers.optionalspace
5711                  * parsers.rbrace
5712
5713 -- block followed by 0 or more optionally
5714 -- indented blocks with first line indented.
5715 parsers.indented_blocks = function(bl)
5716   return Cs( bl
5717             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
5718             * (parsers.blankline^1 + parsers.eof) )
5719 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

5720 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
5721
5722 parsers.bullet = ( parsers.bulletchar * #parsers.spacing

```

```

5723                                     * (parsers.tab + parsers.space~-
3)
5724         + parsers.space * parsers.bulletchar * #parsers.spacing
5725                                     * (parsers.tab + parsers.space~-2)
5726         + parsers.space * parsers.space * parsers.bulletchar
5727                                     * #parsers.spacing
5728                                     * (parsers.tab + parsers.space~-1)
5729         + parsers.space * parsers.space * parsers.space
5730                                     * parsers.bulletchar * #parsers.spacing
5731     )
5732
5733 local function tickbox(interior)
5734     return parsers.optionalspace * parsers.lbracket
5735         * interior * parsers.rbracket * parsers.spacechar~1
5736 end
5737
5738 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
5739 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
5740 parsers.unticked_box = tickbox(parsers.spacechar~1) * Cc(0.0)
5741

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```

5742 parsers.openticks    = Cg(parsers.backtick~1, "ticks")
5743
5744 local function captures_equal_length(_,i,a,b)
5745     return #a == #b and i
5746 end
5747
5748 parsers.closeticks    = parsers.space~-1
5749                       * Cmt(C(parsers.backtick~1)
5750                           * Cb("ticks"), captures_equal_length)
5751
5752 parsers.intickschar    = (parsers.any - S("\n\r`"))
5753                       + (parsers.newline * -parsers.blankline)
5754                       + (parsers.space - parsers.closeticks)
5755                       + (parsers.backtick~1 - parsers.closeticks)
5756
5757 parsers.inticks        = parsers.openticks * parsers.space~-1
5758                       * C(parsers.intickschar~0) * parsers.closeticks

```

### 3.1.4.4 Parsers Used for Markdown Tags and Links

```

5759 parsers.leader        = parsers.space~-3
5760
5761 -- content in balanced brackets, parentheses, or quotes:
5762 parsers.bracketed     = P{ parsers.lbracket
5763                       * (( parsers.backslash / '"' * parsers.rbracket

```

```

5764             + parsers.any - (parsers.lbracket
5765                               + parsers.rbracket
5766                               + parsers.blankline^2)
5767             ) + V(1))^0
5768         * parsers.rbracket }
5769
5770 parsers.inparens = P{ parsers.lparent
5771                       * ((parsers.anyescaped - (parsers.lparent
5772                                                     + parsers.rparent
5773                                                     + parsers.blankline^2)
5774                       ) + V(1))^0
5775                       * parsers.rparent }
5776
5777 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
5778                       * ((parsers.anyescaped - (parsers.squote
5779                                                     + parsers.blankline^2)
5780                       ) + V(1))^0
5781                       * parsers.squote }
5782
5783 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
5784                       * ((parsers.anyescaped - (parsers.dquote
5785                                                     + parsers.blankline^2)
5786                       ) + V(1))^0
5787                       * parsers.dquote }
5788
5789 -- bracketed tag for markdown links, allowing nested brackets:
5790 parsers.tag      = parsers.lbracket
5791                   * Cs((parsers.alphanumeric^1
5792                         + parsers.bracketed
5793                         + parsers.inticks
5794                         + ( parsers.backslash / "\"" * parsers.rbracket
5795                         + parsers.any
5796                         - (parsers.rbracket + parsers.blankline^2))))^0)
5797                   * parsers.rbracket
5798
5799 -- url for markdown links, allowing nested brackets:
5800 parsers.url      = parsers.less * Cs((parsers.anyescaped
5801                                       - parsers.more)^0)
5802                   * parsers.more
5803                   + Cs((parsers.inparens + (parsers.anyescaped
5804                                       - parsers.spacing
5805                                       - parsers.rparent))^1)
5806
5807 -- quoted text, possibly with nested quotes:
5808 parsers.title_s  = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5809                                       + parsers.squoted)^0)
5810                   * parsers.squote

```

```

5811
5812 parsers.title_d      = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5813                                     + parsers.dquoted)^0)
5814                                     * parsers.dquote
5815
5816 parsers.title_p      = parsers.lparent
5817                       * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5818                       * parsers.rparent
5819
5820 parsers.title        = parsers.title_d + parsers.title_s + parsers.title_p
5821
5822 parsers.optionaltitle
5823                       = parsers.spnl * parsers.title * parsers.spacechar^0
5824                       + Cc("")
5825
5826 parsers.indirect_link
5827                       = parsers.tag
5828                       * ( C(parsers.spnl) * parsers.tag
5829                           + Cc(nil) * Cc(nil)  -- always produce exactly two captures
5830                           )
5831
5832 parsers.indirect_image
5833                       = parsers.exclamation * parsers.indirect_link

```

### 3.1.4.5 Parsers Used for HTML

```

5834 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5835 parsers.keyword_exact = function(s)
5836   local parser = P(0)
5837   for i=1,#s do
5838     local c = s:sub(i,i)
5839     local m = c .. upper(c)
5840     parser = parser * S(m)
5841   end
5842   return parser
5843 end
5844
5845 parsers.block_keyword =
5846   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5847   parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5848   parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5849   parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5850   parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5851   parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5852   parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5853   parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5854   parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +

```

```

5855     parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5856     parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5857     parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5858     parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5859     parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5860     parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5861     parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5862     parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5863     parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5864
5865 -- There is no reason to support bad html, so we expect quoted attributes
5866 parsers.htmlattributevalue
5867         = parsers.squote * (parsers.any - (parsers.blankline
5868                                           + parsers.squote))^0
5869         * parsers.squote
5870         + parsers.dquote * (parsers.any - (parsers.blankline
5871                                           + parsers.dquote))^0
5872         * parsers.dquote
5873
5874 parsers.htmlattribute    = parsers.spacing^1
5875                         * (parsers.alphanumeric + S("_-"))^1
5876                         * parsers.sp * parsers.equal * parsers.sp
5877                         * parsers.htmlattributevalue
5878
5879 parsers.htmlcomment     = P("<!--")
5880                         * parsers.optionalspace
5881                         * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5882                         * parsers.optionalspace
5883                         * P("-->")
5884
5885 parsers.htmlinstruction  = P("<?") * (parsers.any - P("?>"))^0 * P(">")
5886
5887 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5888                   * parsers.sp * parsers.more
5889
5890 parsers.openelt_exact = function(s)
5891   return parsers.less * parsers.sp * parsers.keyword_exact(s)
5892         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5893 end
5894
5895 parsers.openelt_block = parsers.sp * parsers.block_keyword
5896                   * parsers.htmlattribute^0 * parsers.sp * parsers.more
5897
5898 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5899                   * parsers.keyword * parsers.sp * parsers.more
5900
5901 parsers.closeelt_exact = function(s)

```



```

5902 return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5903      * parsers.sp * parsers.more
5904 end
5905
5906 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5907                      * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5908                      * parsers.more
5909
5910 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
5911                      * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5912                      * parsers.more
5913
5914 parsers.displaytext = (parsers.any - parsers.less)^1
5915
5916 -- return content between two matched HTML tags
5917 parsers.in_matched = function(s)
5918   return { parsers.openelt_exact(s)
5919           * (V(1) + parsers.displaytext
5920             + (parsers.less - parsers.closeelt_exact(s)))^0
5921           * parsers.closeelt_exact(s) }
5922 end
5923
5924 local function parse_matched_tags(s,pos)
5925   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5926   return lpeg.match(parsers.in_matched(t),s,pos-1)
5927 end
5928
5929 parsers.in_matched_block_tags = parsers.less
5930                               * Cmt(#parsers.openelt_block, parse_matched_tags)
5931

```

#### 3.1.4.6 Parsers Used for HTML Entities

```

5932 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5933                  * C(parsers.hexdigit^1) * parsers.semicolon
5934 parsers.decentity = parsers.ampersand * parsers.hash
5935                  * C(parsers.digit^1) * parsers.semicolon
5936 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5937                  * parsers.semicolon

```

#### 3.1.4.7 Helpers for Link Reference Definitions

```

5938 -- parse a reference definition: [foo]: /bar "title"
5939 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5940                               * parsers.spacechar^0 * parsers.url
5941                               * parsers.optionaltitle

```

#### 3.1.4.8 Inline Elements

```
5942 parsers.Inline          = V("Inline")
5943 parsers.IndentedInline = V("IndentedInline")
5944
5945 -- parse many p between starter and ender
5946 parsers.between = function(p, starter, ender)
5947   local ender2 = B(parsers.nonspacechar) * ender
5948   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5949 end
5950
5951 parsers.urlchar          = parsers.anyescaped
5952                          - parsers.newline
5953                          - parsers.more
5954
5955 parsers.auto_link_url = parsers.less
5956                      * C( parsers.alphanumeric^1 * P("://")
5957                          * parsers.urlchar^1)
5958                      * parsers.more
5959
5960 parsers.auto_link_email
5961                      = parsers.less
5962                      * C((parsers.alphanumeric + S("-._+"))^1
5963                          * P("@") * parsers.urlchar^1)
5964                      * parsers.more
5965
5966 parsers.auto_link_relative_reference
5967                      = parsers.less
5968                      * C(parsers.urlchar^1)
5969                      * parsers.more
5970
```

#### 3.1.4.9 Block Elements

```
5971 parsers.lineof = function(c)
5972   return (parsers.leader * (P(c) * parsers.optionalspace)^3
5973         * (parsers.newline * parsers.blankline^1
5974           + parsers.newline^-1 * parsers.eof))
5975 end
```

#### 3.1.4.10 Headings

```
5976 -- parse Atx heading start and return level
5977 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
5978                      * -parsers.hash / length
5979
5980 -- parse setext header ending and return level
5981 parsers.heading_level = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5982
```

```

5983 local function strip_atx_end(s)
5984     return s:gsub("#%s*\n$", "")
5985 end

```

### 3.1.5 Markdown Reader

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

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

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

```

5986 M.reader = {}
5987 function M.reader.new(writer, options)
5988     local self = {}

```

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

```

5989     self.writer = writer
5990     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.

```

5991     self.parsers = {}
5992     (function(parsers)
5993         setmetatable(self.parsers, {
5994             __index = function (_, key)
5995                 return parsers[key]
5996             end
5997         })
5998     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.

```

5999     local parsers = self.parsers

```

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

```

6000     function self.normalize_tag(tag)

```

```

6001     tag = util.rope_to_string(tag)
6002     tag = tag:gsub("[ \n\r\t]+", " ")
6003     tag = tag:gsub("^ ", ""):gsub(" $", "")
6004     tag = uni_case.casefold(tag, true, false)
6005     return tag
6006 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.

```

6007 local function iterlines(s, f)
6008     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
6009     return util.rope_to_string(rope)
6010 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.

```

6011 if options.preserveTabs then
6012     self.expandtabs = function(s) return s end
6013 else
6014     self.expandtabs = function(s)
6015         if s:find("\t") then
6016             return iterlines(s, util.expand_tabs_in_line)
6017         else
6018             return s
6019         end
6020     end
6021 end

```

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

```

6022 self.parser_functions = {}
6023 self.create_parser = function(name, grammar, toplevel)
6024     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.

```

6025         if toplevel and options.stripIndent then
6026             local min_prefix_length, min_prefix = nil, ''
6027             str = iterlines(str, function(line)

```

```

6028         if lpeg.match(parsers.nonemptyline, line) == nil then
6029             return line
6030         end
6031         line = util.expand_tabs_in_line(line)
6032         local prefix = lpeg.match(C(parsers.optionalspace), line)
6033         local prefix_length = #prefix
6034         local is_shorter = min_prefix_length == nil
6035         is_shorter = is_shorter or prefix_length < min_prefix_length
6036         if is_shorter then
6037             min_prefix_length, min_prefix = prefix_length, prefix
6038         end
6039         return line
6040     end)
6041     str = str:gsub('^' .. min_prefix, '')
6042 end

```

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

```

6043     if toplevel and (options.texComments or options.hybrid) then
6044         str = lpeg.match(Ct(parsers.commented_line^1), str)
6045         str = util.rope_to_string(str)
6046     end
6047     local res = lpeg.match(grammar(), str)
6048     if res == nil then
6049         error(format("%s failed on:\n%s", name, str:sub(1,20)))
6050     else
6051         return res
6052     end
6053 end
6054 end
6055
6056 self.create_parser("parse_blocks",
6057     function()
6058         return parsers.blocks
6059     end, true)
6060
6061 self.create_parser("parse_blocks_nested",
6062     function()
6063         return parsers.blocks_nested
6064     end, false)
6065
6066 self.create_parser("parse_inlines",
6067     function()
6068         return parsers.inlines
6069     end, false)
6070

```

```

6071 self.create_parser("parse_inlines_no_link",
6072                     function()
6073                         return parsers.inlines_no_link
6074                     end, false)
6075
6076 self.create_parser("parse_inlines_no_inline_note",
6077                     function()
6078                         return parsers.inlines_no_inline_note
6079                     end, false)
6080
6081 self.create_parser("parse_inlines_no_html",
6082                     function()
6083                         return parsers.inlines_no_html
6084                     end, false)
6085
6086 self.create_parser("parse_inlines_nbsp",
6087                     function()
6088                         return parsers.inlines_nbsp
6089                     end, false)

```

### 3.1.5.3 Parsers Used for Markdown Lists (local)

```

6090 if options.hashEnumerators then
6091     parsers.dig = parsers.digit + parsers.hash
6092 else
6093     parsers.dig = parsers.digit
6094 end
6095
6096 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
6097                     + C(parsers.dig^2 * parsers.period) * #parsers.spacing
6098                       * (parsers.tab + parsers.space^1)
6099                     + C(parsers.dig * parsers.period) * #parsers.spacing
6100                       * (parsers.tab + parsers.space^2)
6101                     + parsers.space * C(parsers.dig^2 * parsers.period)
6102                       * #parsers.spacing
6103                     + parsers.space * C(parsers.dig * parsers.period)
6104                       * #parsers.spacing
6105                       * (parsers.tab + parsers.space^1)
6106                     + parsers.space * parsers.space * C(parsers.dig^1
6107                       * parsers.period) * #parsers.spacing

```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```

6108 -- strip off leading > and indents, and run through blocks
6109 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^1)
6110                             1) / ""
6111                             * parsers.linechar^0 * parsers.newline)^1
6112                             * (-V("BlockquoteExceptions") * parsers.linechar^1

```

```

6112             * parsers.newline)^0
6113
6114     if not options.breakableBlockquotes then
6115         parsers.blockquote_body = parsers.blockquote_body
6116             * (parsers.blankline^0 / "")
6117     end

```

### 3.1.5.5 Helpers for Links and Link Reference Definitions (local)

```

6118     -- List of references defined in the document
6119     local references
6120

```

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.

```

6121     function self.register_link(tag, url, title,
6122                                attributes)
6123         tag = self.normalize_tag(tag)
6124         references[tag] = {
6125             url = url,
6126             title = title,
6127             attributes = attributes,
6128         }
6129         return ""
6130     end
6131

```

The `reader->lookup_reference` method looks up a reference with link label `tag`. When the reference exists the method returns a link. The attributes of a link are produced by merging the attributes of the link reference and the optional `attributes`. Otherwise, the method returns a two-tuple of `nil` and fallback text constructed from the link text `label` and the optional spaces `sps` between the link text and the link label.

```

6132     function self.lookup_reference(label, sps, tag,
6133                                attributes)
6134         local tagpart
6135         if not tag then
6136             tag = label
6137             tagpart = ""
6138         elseif tag == "" then
6139             tag = label
6140             tagpart = "[]"
6141         else
6142             tagpart = {
6143                 "[",
6144                 self.parser_functions.parse_inlines(tag),

```

```

6145         "]"
6146     }
6147 end
6148 if sps then
6149     tagpart = {sps, tagpart}
6150 end
6151 tag = self.normalize_tag(tag)
6152 local r = references[tag]
6153 if r then
6154     local merged_attributes = {}
6155     for _, attribute in ipairs(r.attributes or {}) do
6156         table.insert(merged_attributes, attribute)
6157     end
6158     for _, attribute in ipairs(attributes or {}) do
6159         table.insert(merged_attributes, attribute)
6160     end
6161     if #merged_attributes == 0 then
6162         merged_attributes = nil
6163     end
6164     return {
6165         url = r.url,
6166         title = r.title,
6167         attributes = merged_attributes,
6168     }
6169 else
6170     return nil, {
6171         "[",
6172         self.parser_functions.parse_inlines(label),
6173         "]",
6174         tagpart
6175     }
6176 end
6177 end
6178
6179 -- lookup link reference and return a link, if the reference is found,
6180 -- or a bracketed label otherwise.
6181 local function indirect_link(label, sps, tag)
6182     return writer.defer_call(function()
6183         local r, fallback = self.lookup_reference(label, sps, tag)
6184         if r then
6185             return writer.link(
6186                 self.parser_functions.parse_inlines_no_link(label),
6187                 r.url, r.title)
6188         else
6189             return fallback
6190         end
6191     end)

```



```

6192 end
6193
6194 -- lookup image reference and return an image, if the reference is found,
6195 -- or a bracketed label otherwise.
6196 local function indirect_image(label, sps, tag)
6197     return writer.defer_call(function()
6198         local r,fallback = self.lookup_reference(label, sps, tag)
6199         if r then
6200             return writer.image(writer.string(label), r.url, r.title)
6201         else
6202             return {"!", fallback}
6203         end
6204     end)
6205 end
6206
6207 parsers.direct_link_tail = parsers.spnl
6208                         * parsers.lparent
6209                         * (parsers.url + Cc("")) -- link can be empty [foo]()
6210                         * parsers.optionaltitle
6211                         * parsers.rparent
6212
6213 parsers.direct_link = (parsers.tag / self.parser_functions.parse_inlines_no_link)
6214                     * parsers.direct_link_tail
6215
6216 parsers.direct_image = parsers.exclamation
6217                     * (parsers.tag / self.parser_functions.parse_inlines)
6218                     * parsers.direct_link_tail

```

### 3.1.5.6 Inline Elements (local)

```

6219 parsers.Str          = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
6220                     / writer.string
6221
6222 parsers.Symbol       = (V("SpecialChar") - parsers.tightblocksep)
6223                     / writer.string
6224
6225 parsers.Ellipsis     = P("...") / writer.ellipsis
6226
6227 parsers.Smart        = parsers.Ellipsis
6228
6229 parsers.Code         = parsers.inticks / writer.code
6230
6231 if options.blankBeforeBlockquote then
6232     parsers.bqstart = parsers.fail
6233 else
6234     parsers.bqstart = parsers.more
6235 end

```

```

6236
6237 if options.blankBeforeHeading then
6238     parsers.headerstart = parsers.fail
6239 else
6240     parsers.headerstart = parsers.hash
6241                         + (parsers.line * (parsers.equal^1 + parsers.dash^1)
6242                         * parsers.optionalspace * parsers.newline)
6243 end
6244
6245 parsers.EndlineExceptions
6246     = parsers.blankline -- paragraph break
6247     + parsers.tightblocksep -- nested list
6248     + parsers.eof        -- end of document
6249     + parsers.bqstart
6250     + parsers.headerstart
6251
6252 parsers.Endline = parsers.newline
6253                 * -V("EndlineExceptions")
6254                 * parsers.spacechar^0
6255                 / (options.hardLineBreaks and writer.hard_line_break
6256                   or writer.space)
6257
6258 parsers.OptionalIndent
6259     = parsers.spacechar^1 / writer.space
6260
6261 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6262               + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6263               + parsers.spacechar^1 * parsers.Endline
6264               * parsers.optionalspace
6265               / (options.hardLineBreaks
6266                 and writer.hard_line_break
6267                 or writer.space)
6268               + parsers.spacechar^1 * parsers.optionalspace
6269               / writer.space
6270
6271 parsers.NonbreakingEndline
6272     = parsers.newline
6273     * -V("EndlineExceptions")
6274     * parsers.spacechar^0
6275     / (options.hardLineBreaks and writer.hard_line_break
6276       or writer.nbsp)
6277
6278 parsers.NonbreakingSpace
6279     = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6280     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6281     + parsers.spacechar^1 * parsers.Endline
6282     * parsers.optionalspace

```

```

6283                                     / (options.hardLineBreaks
6284                                     and writer.hard_line_break
6285                                     or writer.nbsp)
6286         + parsers.spacechar~1 * parsers.optionalspace
6287                                     / writer.nbsp
6288
6289     if options.underscores then
6290         parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
6291                                           parsers.doubleasterisks)
6292                           + parsers.between(parsers.Inline, parsers.doubleunderscores,
6293                                           parsers.doubleunderscores)
6294                           ) / writer.strong
6295
6296         parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
6297                                           parsers.asterisk)
6298                           + parsers.between(parsers.Inline, parsers.underscore,
6299                                           parsers.underscore)
6300                           ) / writer.emphasis
6301     else
6302         parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
6303                                           parsers.doubleasterisks)
6304                           ) / writer.strong
6305
6306         parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
6307                                           parsers.asterisk)
6308                           ) / writer.emphasis
6309     end
6310

```

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.

```

6311 function self.auto_link_url(url, attributes)
6312     return writer.link(writer.escape(url),
6313                       url, nil, attributes)
6314 end
6315

```

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.

```

6316 function self.auto_link_email(email, attributes)
6317     return writer.link(writer.escape(email),
6318                       "mailto:".email,
6319                       nil, attributes)
6320 end
6321

```

```

6322 parsers.AutoLinkUrl = parsers.auto_link_url
6323                       / self.auto_link_url
6324
6325 parsers.AutoLinkEmail
6326                       = parsers.auto_link_email
6327                       / self.auto_link_email
6328
6329 parsers.AutoLinkRelativeReference
6330                       = parsers.auto_link_relative_reference
6331                       / self.auto_link_url
6332
6333 parsers.DirectLink    = parsers.direct_link
6334                       / writer.link
6335
6336 parsers.IndirectLink  = parsers.indirect_link
6337                       / indirect_link
6338
6339 -- parse a link or image (direct or indirect)
6340 parsers.Link          = parsers.DirectLink + parsers.IndirectLink
6341
6342 parsers.DirectImage   = parsers.direct_image
6343                       / writer.image
6344
6345 parsers.IndirectImage = parsers.indirect_image
6346                       / indirect_image
6347
6348 parsers.Image         = parsers.DirectImage + parsers.IndirectImage
6349
6350 -- avoid parsing long strings of * or _ as emph/strong
6351 parsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
6352                       / writer.string
6353
6354 parsers.EscapedChar   = parsers.backslash * C(parsers.escapable) / writer.string
6355
6356 parsers.InlineHtml    = parsers.emptyelt_any / writer.inline_html_tag
6357                       + (parsers.htmlcomment / self.parser_functions.parse_inlines_
6358                          / writer.inline_html_comment
6359                          + parsers.htmlinstruction
6360                          + parsers.openelt_any / writer.inline_html_tag
6361                          + parsers.closeelt_any / writer.inline_html_tag
6362
6363 parsers.HtmlEntity     = parsers.hexentity / entities.hex_entity / writer.string
6364                       + parsers.decentity / entities.dec_entity / writer.string
6365                       + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.7 Block Elements (local)

```

6366 parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_ne
6367 / writer.block_html_comment
6368 + parsers.emptyelt_block / writer.block_html_element
6369 + parsers.openelt_exact("hr") / writer.block_html_element
6370 + parsers.in_matched_block_tags / writer.block_html_element
6371 + parsers.htmlinstruction
6372
6373 parsers.Verbatim = Cs( (parsers.blanklines
6374 * ((parsers.indentedline - parsers.blankline))^1)^1
6375 ) / self.expandtabs / writer.verbatim
6376
6377 parsers.BlockquoteExceptions = parsers.leader * parsers.more
6378 + parsers.blankline
6379
6380 parsers.Blockquote = Cs(parsers.blockquote_body^1)
6381 / self.parser_functions.parse_blocks_nested
6382 / writer.blockquote
6383
6384 parsers.ThematicBreak = ( parsers.lineof(parsers.asterisk)
6385 + parsers.lineof(parsers.dash)
6386 + parsers.lineof(parsers.underscore)
6387 ) / writer.thematic_break
6388
6389 parsers.Reference = parsers.define_reference_parser
6390 * parsers.blankline^1
6391 / self.register_link
6392
6393 parsers.Paragraph = parsers.nonindentspace * Ct(parsers.Inline^1)
6394 * ( parsers.newline
6395 * ( parsers.blankline^1
6396 + #V("EndlineExceptions")
6397 )
6398 + parsers.eof)
6399 / writer.paragraph
6400
6401 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
6402 / writer.plain

```

### 3.1.5.8 Lists (local)

```

6403 parsers.starter = parsers.bullet + parsers.enumerator
6404
6405 if options.taskLists then
6406   parsers.tickbox = ( parsers.ticked_box
6407 + parsers.halfticked_box
6408 + parsers.unticked_box
6409 ) / writer.tickbox

```

```

6410 else
6411     parsers.tickbox = parsers.fail
6412 end
6413
6414 -- we use \001 as a separator between a tight list item and a
6415 -- nested list under it.
6416 parsers.NestedList = Cs((parsers.optionallyindentedline
6417     - parsers.starter)^1)
6418     / function(a) return "\001"..a end
6419
6420 parsers.ListBlockLine = parsers.optionallyindentedline
6421     - parsers.blankline - (parsers.indent^-
1
6422     * parsers.starter)
6423
6424 parsers.ListBlock = parsers.line * parsers.ListBlockLine^0
6425
6426 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6427     * parsers.ListBlock
6428
6429 parsers.TightListItem = function(starter)
6430     return -parsers.ThematicBreak
6431     * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Ne
1)
6432     / self.parser_functions.parse_blocks_nested)
6433     * -(parsers.blanklines * parsers.indent)
6434 end
6435
6436 parsers.LooseListItem = function(starter)
6437     return -parsers.ThematicBreak
6438     * Cs( starter / "" * parsers.tickbox^-1 * parsers.ListBlock * Cc("\n")
6439     * (parsers.NestedList + parsers.ListContinuationBlock^0)
6440     * (parsers.blanklines / "\n\n")
6441     ) / self.parser_functions.parse_blocks_nested
6442 end
6443
6444 parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
6445     * parsers.skipblanklines * -parsers.bullet
6446     + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
6447     * parsers.skipblanklines )
6448     / writer.bulletlist
6449
6450 local function ordered_list(items,tight,startnum)
6451     if options.startNumber then
6452         startnum = tonumber(startnum) or 1 -- fallback for '#'
6453         if startnum ~= nil then
6454             startnum = math.floor(startnum)

```

```

6455     end
6456   else
6457     startnum = nil
6458   end
6459   return writer.orderedlist(items,tight,startnum)
6460 end
6461
6462 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
6463   ( Ct(parsers.TightListItem(Cb("listtype")))
6464     * parsers.TightListItem(parsers.enumerator)^0)
6465   * Cc(true) * parsers.skipblanklines * -parsers.enumerator
6466   + Ct(parsers.LooseListItem(Cb("listtype")))
6467     * parsers.LooseListItem(parsers.enumerator)^0)
6468   * Cc(false) * parsers.skipblanklines
6469   ) * Cb("listtype") / ordered_list

```

### 3.1.5.9 Blank (local)

```

6470 parsers.Blank      = parsers.blankline / ""
6471                   + V("Reference")
6472                   + (parsers.tightblocksep / "\n")

```

### 3.1.5.10 Headings (local)

```

6473 -- parse atx header
6474 parsers.AtxHeading = Cg(parsers.heading_start, "level")
6475                   * parsers.optionalspace
6476                   * (C(parsers.line)
6477                     / strip_atx_end
6478                     / self.parser_functions.parse_inlines)
6479                   * Cb("level")
6480                   / writer.heading
6481
6482 parsers.SetextHeading = #(parsers.line * S("="))
6483                   * Ct(parsers.linechar^1
6484                       / self.parser_functions.parse_inlines)
6485                   * parsers.newline
6486                   * parsers.heading_level
6487                   * parsers.optionalspace
6488                   * parsers.newline
6489                   / writer.heading
6490
6491 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

**3.1.5.11 Syntax Specification** Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions`

and returns a conversion function that takes a markdown string and turns it into a plain TeX output.

```
6492 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.

```
6493   local walkable_syntax = (function(global_walkable_syntax)
6494     local local_walkable_syntax = {}
6495     for lhs, rule in pairs(global_walkable_syntax) do
6496       local_walkable_syntax[lhs] = util.table_copy(rule)
6497     end
6498     return local_walkable_syntax
6499   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.

```
6500   local current_extension_name = nil
6501   self.insert_pattern = function(selector, pattern, pattern_name)
6502     assert(pattern_name == nil or type(pattern_name) == "string")
6503     local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
6504     assert(lhs ~= nil,
6505       [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
6506       .. selector .. ["]])
6507     assert(walkable_syntax[lhs] ~= nil,
6508       [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
6509     assert(pos == "before" or pos == "after" or pos == "instead of",
6510       [[Expected positional specifier "before", "after", or "instead of", not "]]
6511       .. pos .. ["]])
6512     local rule = walkable_syntax[lhs]
6513     local index = nil
6514     for current_index, current_rhs in ipairs(rule) do
6515       if type(current_rhs) == "string" and current_rhs == rhs then
6516         index = current_index
6517         if pos == "after" then
6518           index = index + 1
6519         end
6520         break
6521       end
6522     end
6523     assert(index ~= nil,
6524       [[Rule ]] .. lhs .. [[ -> ]] .. rhs
6525       .. [[ does not exist in markdown grammar]])
6526     local accountable_pattern
```



```

6527     if current_extension_name then
6528         accountable_pattern = { pattern, current_extension_name, pattern_name }
6529     else
6530         assert(type(pattern) == "string",
6531             [[reader->insert_pattern() was called outside an extension with ]]
6532             .. [[a PEG pattern instead of a rule name]])
6533         accountable_pattern = pattern
6534     end
6535     if pos == "instead of" then
6536         rule[index] = accountable_pattern
6537     else
6538         table.insert(rule, index, accountable_pattern)
6539     end
6540 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.

```

6541     local syntax =
6542     { "Blocks",
6543
6544         Blocks = V("InitializeState")
6545             * ( V("ExpectedJekyllData")
6546                 * (V("Blank")^0 / writer.interblocksep))^~
6547             * V("Blank")^0
6548             * V("Block")^-1
6549             * ( V("Blank")^0 / writer.interblocksep
6550                 * V("Block"))^0
6551             * V("Blank")^0 * parsers.eof,
6552
6553         ExpectedJekyllData = parsers.fail,
6554
6555         Blank = parsers.Blank,
6556         Reference = parsers.Reference,
6557
6558         Blockquote = parsers.Blockquote,
6559         Verbatim = parsers.Verbatim,
6560         ThematicBreak = parsers.ThematicBreak,
6561         BulletList = parsers.BulletList,
6562         OrderedList = parsers.OrderedList,
6563         Heading = parsers.Heading,
6564         DisplayHtml = parsers.DisplayHtml,
6565         Paragraph = parsers.Paragraph,
6566         Plain = parsers.Plain,
6567
6568         EndlineExceptions = parsers.EndlineExceptions,
6569         BlockquoteExceptions = parsers.BlockquoteExceptions,

```

```

6570
6571      Str                = parsers.Str,
6572      Space              = parsers.Space,
6573      OptionalIndent    = parsers.OptionalIndent,
6574      Endline            = parsers.Endline,
6575      UlOrStarLine       = parsers.UlOrStarLine,
6576      Strong             = parsers.Strong,
6577      Emph               = parsers.Emph,
6578      Link               = parsers.Link,
6579      Image              = parsers.Image,
6580      Code               = parsers.Code,
6581      AutoLinkUrl        = parsers.AutoLinkUrl,
6582      AutoLinkEmail      = parsers.AutoLinkEmail,
6583      AutoLinkRelativeReference
6584                                = parsers.AutoLinkRelativeReference,
6585      InlineHtml         = parsers.InlineHtml,
6586      HtmlEntity         = parsers.HtmlEntity,
6587      EscapedChar        = parsers.EscapedChar,
6588      Smart              = parsers.Smart,
6589      Symbol             = parsers.Symbol,
6590      SpecialChar        = parsers.fail,
6591      InitializeState    = parsers.succeed,
6592  }

```

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]`.

```

6593      self.update_rule = function(rule_name, get_pattern)
6594          assert(current_extension_name ~= nil)
6595          assert(syntax[rule_name] ~= nil,
6596              [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
6597          local previous_pattern
6598          local extension_name
6599          if walkable_syntax[rule_name] then
6600              local previous_accountable_pattern = walkable_syntax[rule_name][1]
6601              previous_pattern = previous_accountable_pattern[1]
6602              extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
6603          else
6604              previous_pattern = nil
6605              extension_name = current_extension_name
6606          end
6607          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
```

```
6608     if type(get_pattern) == "function" then
6609         pattern = get_pattern(previous_pattern)
6610     else
6611         assert(previous_pattern == nil,
6612             [[Rule ]] .. rule_name ..
6613             [[ has already been updated by ]] .. extension_name)
6614         pattern = get_pattern
6615     end
6616     local accountable_pattern = { pattern, extension_name, rule_name }
6617     walkable_syntax[rule_name] = { accountable_pattern }
6618 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.

```
6619     local special_characters = {}
6620     self.add_special_character = function(c)
6621         table.insert(special_characters, c)
6622         syntax.SpecialChar = S(table.concat(special_characters, ""))
6623     end
6624
6625     self.add_special_character("*")
6626     self.add_special_character("[")
6627     self.add_special_character("]")
6628     self.add_special_character("<")
6629     self.add_special_character("!")
6630     self.add_special_character("\\")
```

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

```
6631     self.initialize_named_group = function(name, value)
6632         syntax.InitializeState = syntax.InitializeState
6633             * Cg(Ct("") / value, name)
6634     end
```

Apply syntax extensions.

```
6635     for _, extension in ipairs(extensions) do
6636         current_extension_name = extension.name
6637         extension.extend_writer(writer)
```

```

6638     extension.extend_reader(self)
6639 end
6640 current_extension_name = nil

```

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

```

6641 if options.debugExtensions then
6642   local sorted_lhs = {}
6643   for lhs, _ in pairs(walkable_syntax) do
6644     table.insert(sorted_lhs, lhs)
6645   end
6646   table.sort(sorted_lhs)
6647
6648   local output_lines = {"{"}
6649   for lhs_index, lhs in ipairs(sorted_lhs) do
6650     local encoded_lhs = util.encode_json_string(lhs)
6651     table.insert(output_lines, [[    ]] .. encoded_lhs .. [[:  ]])
6652     local rule = walkable_syntax[lhs]
6653     for rhs_index, rhs in ipairs(rule) do
6654       local human_readable_rhs
6655       if type(rhs) == "string" then
6656         human_readable_rhs = rhs
6657       else
6658         local pattern_name
6659         if rhs[3] then
6660           pattern_name = rhs[3]
6661         else
6662           pattern_name = "Anonymous Pattern"
6663         end
6664         local extension_name = rhs[2]
6665         human_readable_rhs = pattern_name .. [[ (]] .. extension_name .. [[)] ]
6666       end
6667       local encoded_rhs = util.encode_json_string(human_readable_rhs)
6668       local output_line = [[    ]] .. encoded_rhs
6669       if rhs_index < #rule then
6670         output_line = output_line .. ", "
6671       end
6672       table.insert(output_lines, output_line)
6673     end
6674     local output_line = "  ]"
6675     if lhs_index < #sorted_lhs then
6676       output_line = output_line .. ", "
6677     end
6678     table.insert(output_lines, output_line)
6679   end
6680   table.insert(output_lines, "}")
6681

```

```

6682     local output = table.concat(output_lines, "\n")
6683     local output_filename = options.debugExtensionsFileName
6684     local output_file = assert(io.open(output_filename, "w"),
6685         [[Could not open file ]] .. output_filename .. [[ for writing]])
6686     assert(output_file:write(output))
6687     assert(output_file:close())
6688 end

```

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

```

6689     walkable_syntax["IndentedInline"] = util.table_copy(
6690         walkable_syntax["Inline"])
6691     self.insert_pattern(
6692         "IndentedInline instead of Space",
6693         "OptionalIndent")

```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```

6694     for lhs, rule in pairs(walkable_syntax) do
6695         syntax[lhs] = parsers.fail
6696         for _, rhs in ipairs(rule) do
6697             local pattern

```

Although the interface of the `reader->insert_pattern` method does document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

6698         if type(rhs) == "string" then
6699             pattern = V(rhs)
6700         else
6701             pattern = rhs[1]
6702             if type(pattern) == "string" then
6703                 pattern = V(pattern)
6704             end
6705         end
6706         syntax[lhs] = syntax[lhs] + pattern
6707     end
6708 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.

```

6709     if options.underscores then
6710         self.add_special_character("_")
6711     end
6712
6713     if not options.codeSpans then

```

```

6714     syntax.Code = parsers.fail
6715 else
6716     self.add_special_character("`")
6717 end
6718
6719 if not options.html then
6720     syntax.DisplayHtml = parsers.fail
6721     syntax.InlineHtml = parsers.fail
6722     syntax.HtmlEntity = parsers.fail
6723 else
6724     self.add_special_character("&")
6725 end
6726
6727 if options.preserveTabs then
6728     options.stripIndent = false
6729 end
6730
6731 if not options.smartEllipses then
6732     syntax.Smart = parsers.fail
6733 else
6734     self.add_special_character(".")
6735 end
6736
6737 if not options.relativeReferences then
6738     syntax.AutoLinkRelativeReference = parsers.fail
6739 end
6740
6741 local blocks_nested_t = util.table_copy(syntax)
6742 blocks_nested_t.ExpectedJekyllData = parsers.fail
6743 parsers.blocks_nested = Ct(blocks_nested_t)
6744
6745 parsers.blocks = Ct(syntax)
6746
6747 local inlines_t = util.table_copy(syntax)
6748 inlines_t[1] = "Inlines"
6749 inlines_t.Inlines = V("InitializeState")
6750     * parsers.Inline^0
6751     * ( parsers.spacing^0
6752     * parsers.eof / "")
6753 parsers.inlines = Ct(inlines_t)
6754
6755 local inlines_no_link_t = util.table_copy(inlines_t)
6756 inlines_no_link_t.Link = parsers.fail
6757 parsers.inlines_no_link = Ct(inlines_no_link_t)
6758
6759 local inlines_no_inline_note_t = util.table_copy(inlines_t)
6760 inlines_no_inline_note_t.InlineNote = parsers.fail

```

```

6761     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6762
6763     local inlines_no_html_t = util.table_copy(inlines_t)
6764     inlines_no_html_t.DisplayHtml = parsers.fail
6765     inlines_no_html_t.InlineHtml = parsers.fail
6766     inlines_no_html_t.HtmlEntity = parsers.fail
6767     parsers.inlines_no_html = Ct(inlines_no_html_t)
6768
6769     local inlines_nbsp_t = util.table_copy(inlines_t)
6770     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6771     inlines_nbsp_t.Space = parsers.NonbreakingSpace
6772     parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

Return a function that converts markdown string `input` into a plain T<sub>E</sub>X output and returns it..

```

6773     return function(input)

```

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

```

6774         input = input:gsub("\r\n?", "\n")
6775         if input:sub(-1) ~= "\n" then
6776             input = input .. "\n"
6777         end

```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```

6778         references = {}
6779         local opt_string = {}
6780         for k, _ in pairs(defaultOptions) do
6781             local v = options[k]
6782             if type(v) == "table" then
6783                 for _, i in ipairs(v) do
6784                     opt_string[#opt_string+1] = k .. "=" .. tostring(i)
6785                 end
6786             elseif k ~= "cacheDir" then
6787                 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
6788             end
6789         end
6790         table.sort(opt_string)
6791         local salt = table.concat(opt_string, ",") .. "," .. metadata.version
6792         local output

```

If we cache markdown documents, produce the cache file and transform its filename to plain T<sub>E</sub>X output via the `writer->pack` method.

```

6793         local function convert(input)
6794             local document = self.parser_functions.parse_blocks(input)
6795             return util.ropetostring(writer.document(document))

```

```

6796     end
6797     if options.eagerCache or options.finalizeCache then
6798         local name = util.cache(options.cacheDir, input, salt, convert,
6799                                ".md" .. writer.suffix)
6800         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

6801     else
6802         output = convert(input)
6803     end

```

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

```

6804     if options.finalizeCache then
6805         local file, mode
6806         if options.frozenCacheCounter > 0 then
6807             mode = "a"
6808         else
6809             mode = "w"
6810         end
6811         file = assert(io.open(options.frozenCacheFileName, mode),
6812                          [[Could not open file "]] .. options.frozenCacheFileName
6813                          .. [[" for writing]])
6814         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
6815                          .. [[markdownFrozenCache]] .. options.frozenCacheCounter
6816                          .. [[\\endcsname{]] .. output .. [[}}]] .. "\\n"))
6817         assert(file:close())
6818     end
6819     return output
6820 end
6821 end
6822 return self
6823 end

```

### 3.1.6 Built-In Syntax Extensions

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

```

6824 M.extensions = {}

```

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



```

6825 M.extensions.bracketed_spans = function()
6826   return {
6827     name = "built-in bracketed_spans syntax extension",
6828     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.

```

6829       function self.span(s, attr)
6830         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
6831               self.attributes(attr),
6832               s,
6833               "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
6834       end
6835     end, extend_reader = function(self)
6836       local parsers = self.parsers
6837       local writer = self.writer
6838
6839       local Span = parsers.between(parsers.Inline,
6840                                   parsers.lbracket,
6841                                   parsers.rbracket)
6842                                   * Ct(parsers.attributes)
6843                                   / writer.span
6844
6845       self.insert_pattern("Inline after Emph",
6846                           Span, "Span")
6847     end
6848   }
6849 end

```

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

```

6850 M.extensions.citations = function(citation_nbsps)
6851   return {
6852     name = "built-in citations syntax extension",
6853     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.

```

6854     function self.citations(text_cites, cites)
6855         local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
6856             "{", #cites, "}"}
6857         for _,cite in ipairs(cites) do
6858             buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6859                 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}" }
6860         end
6861         return buffer
6862     end
6863 end, extend_reader = function(self)
6864     local parsers = self.parsers
6865     local writer = self.writer
6866
6867     local citation_chars
6868         = parsers.alphanumeric
6869         + S("#$%&-+<>~/_")
6870
6871     local citation_name
6872         = Cs(parsers.dash^-1) * parsers.at
6873         * Cs(citation_chars
6874             * (((citation_chars + parsers.internal_punctuation
6875                 - parsers.comma - parsers.semicolon)
6876                 * -#((parsers.internal_punctuation - parsers.comma
6877                     - parsers.semicolon)^0
6878                     * -(citation_chars + parsers.internal_punctuation
6879                         - parsers.comma - parsers.semicolon)))^0
6880                 * citation_chars)^-1)
6881
6882     local citation_body_prenote
6883         = Cs((parsers.alphanumeric^1
6884             + parsers.bracketed
6885             + parsers.inticks
6886             + (parsers.anyescaped
6887                 - (parsers.rbracket + parsers.blankline^2))
6888             - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6889
6890     local citation_body_postnote
6891         = Cs((parsers.alphanumeric^1
6892             + parsers.bracketed
6893             + parsers.inticks

```

```

6894         + (parsers.anyescaped
6895         - (parsers.rbracket + parsers.semicolon
6896         + parsers.blankline^2))
6897         - (parsers.spnl * parsers.rbracket))^0)
6898
6899     local citation_body_chunk
6900         = citation_body_prenote
6901         * parsers.spnl * citation_name
6902         * (parsers.internal_punctuation - parsers.semicolon)^-
1
6903         * parsers.spnl * citation_body_postnote
6904
6905     local citation_body
6906         = citation_body_chunk
6907         * (parsers.semicolon * parsers.spnl
6908         * citation_body_chunk)^0
6909
6910     local citation_headless_body_postnote
6911         = Cs((parsers.alphanumeric^1
6912         + parsers.bracketed
6913         + parsers.inticks
6914         + (parsers.anyescaped
6915         - (parsers.rbracket + parsers.at
6916         + parsers.semicolon + parsers.blankline^2))
6917         - (parsers.spnl * parsers.rbracket))^0)
6918
6919     local citation_headless_body
6920         = citation_headless_body_postnote
6921         * (parsers.sp * parsers.semicolon * parsers.spnl
6922         * citation_body_chunk)^0
6923
6924     local citations
6925         = function(text_cites, raw_cites)
6926         local function normalize(str)
6927             if str == "" then
6928                 str = nil
6929             else
6930                 str = (citation_nbsps and
6931                 self.parser_functions.parse_inlines_nbsp or
6932                 self.parser_functions.parse_inlines)(str)
6933             end
6934             return str
6935         end
6936
6937         local cites = {}
6938         for i = 1,#raw_cites,4 do
6939             cites[#cites+1] = {

```

```

6940         prenote = normalize(raw_cites[i]),
6941         suppress_author = raw_cites[i+1] == "-",
6942         name = writer.identifier(raw_cites[i+2]),
6943         postnote = normalize(raw_cites[i+3]),
6944     }
6945     end
6946     return writer.citations(text_cites, cites)
6947 end
6948
6949 local TextCitations
6950     = Ct((parsers.spnl
6951     * Cc("")
6952     * citation_name
6953     * ((parsers.spnl
6954     * parsers.lbracket
6955     * citation_headless_body
6956     * parsers.rbracket) + Cc("")))~1)
6957 / function(raw_cites)
6958     return citations(true, raw_cites)
6959 end
6960
6961 local ParenthesizedCitations
6962     = Ct((parsers.spnl
6963     * parsers.lbracket
6964     * citation_body
6965     * parsers.rbracket)^1)
6966 / function(raw_cites)
6967     return citations(false, raw_cites)
6968 end
6969
6970 local Citations = TextCitations + ParenthesizedCitations
6971
6972 self.insert_pattern("Inline after Emph",
6973     Citations, "Citations")
6974
6975 self.add_special_character("@")
6976 self.add_special_character("-")
6977 end
6978 }
6979 end

```

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

```

6980 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.

```

6981 local languages_json = (function()
6982     local base, prev, curr
6983     for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
6984         local file = io.open(pathname, "r")
6985         if not file then goto continue end
6986         local input = assert(file:read("*a"))
6987         assert(file:close())
6988         local json = input:gsub('("[^\\n]-"):','[%1]=')
6989         curr = load("_ENV = {}; return "..json")()
6990         if type(curr) == "table" then
6991             if base == nil then
6992                 base = curr
6993             else
6994                 setmetatable(prev, { __index = curr })
6995             end
6996             prev = curr
6997         end
6998         ::continue::
6999     end
7000     return base or {}
7001 end)()
7002
7003 return {
7004     name = "built-in content_blocks syntax extension",
7005     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.

```

7006     function self.contentblock(src,suf,type,tit)
7007         if not self.is_writing then return "" end
7008         src = src..".."..suf
7009         suf = suf:lower()
7010         if type == "onlineimage" then
7011             return {"\\markdownRendererContentBlockOnlineImage{"..suf.."},"",
7012                     {"",self.string(src),"},",
7013                     {"",self.uri(src),"},",
7014                     {"",self.string(tit or ""),"},"}
7015         elseif languages_json[suf] then
7016             return {"\\markdownRendererContentBlockCode{"..suf.."},"",
7017                     {"",self.string(languages_json[suf]),"},",
7018                     {"",self.string(src),"},",

```

```

7019             {"",self.uri(src),""},
7020             {"",self.string(tit or ""),""}
7021     else
7022         return {"\\markdownRendererContentBlock{"",suf,""},
7023             {"",self.string(src),""},
7024             {"",self.uri(src),""},
7025             {"",self.string(tit or ""),""}
7026     end
7027 end
7028 end, extend_reader = function(self)
7029     local parsers = self.parsers
7030     local writer = self.writer
7031
7032     local contentblock_tail
7033         = parsers.optionaltitle
7034         * (parsers.newline + parsers.eof)
7035
7036     -- case insensitive online image suffix:
7037     local onlineimagesuffix
7038         = (function(...)
7039             local parser = nil
7040             for _, suffix in ipairs({...}) do
7041                 local pattern=nil
7042                 for i=1,#suffix do
7043                     local char=suffix:sub(i,i)
7044                     char = S(char:lower()..char:upper())
7045                     if pattern == nil then
7046                         pattern = char
7047                     else
7048                         pattern = pattern * char
7049                     end
7050                 end
7051                 if parser == nil then
7052                     parser = pattern
7053                 else
7054                     parser = parser + pattern
7055                 end
7056             end
7057             return parser
7058         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
7059
7060     -- online image url for iA Writer content blocks with mandatory suffix,
7061     -- allowing nested brackets:
7062     local onlineimageurl
7063         = (parsers.less
7064             * Cs((parsers.anyescaped
7065                 - parsers.more

```

```

7066         - #(parsers.period
7067           * onlineimagesuffix
7068           * parsers.more
7069           * contentblock_tail))^0)
7070     * parsers.period
7071     * Cs(onlineimagesuffix)
7072     * parsers.more
7073     + (Cs((parsers.inparens
7074         + (parsers.anyescaped
7075           - parsers.spacing
7076           - parsers.rparent
7077           - #(parsers.period
7078             * onlineimagesuffix
7079             * contentblock_tail))))^0)
7080     * parsers.period
7081     * Cs(onlineimagesuffix))
7082   ) * Cc("onlineimage")
7083
7084   -- filename for iA Writer content blocks with mandatory suffix:
7085   local localfilepath
7086     = parsers.slash
7087     * Cs((parsers.anyescaped
7088         - parsers.tab
7089         - parsers.newline
7090         - #(parsers.period
7091             * parsers.alphanumeric^1
7092             * contentblock_tail))^1)
7093     * parsers.period
7094     * Cs(parsers.alphanumeric^1)
7095     * Cc("localfile")
7096
7097   local ContentBlock
7098     = parsers.leader
7099     * (localfilepath + onlineimageurl)
7100     * contentblock_tail
7101     / writer.contentblock
7102
7103   self.insert_pattern("Block before Blockquote",
7104                       ContentBlock, "ContentBlock")
7105 end
7106 }
7107 end

```

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

```

7108 M.extensions.definition_lists = function(tight_lists)
7109   return {
7110     name = "built-in definition_lists syntax extension",
7111     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.

```

7112     local function dlitem(term, defs)
7113       local retVal = {"\\markdownRendererDlItem{",term,""}
7114       for _, def in ipairs(defs) do
7115         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
7116                               "\\markdownRendererDlDefinitionEnd "}
7117       end
7118       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
7119       return retVal
7120     end
7121
7122     function self.definitionlist(items,tight)
7123       if not self.is_writing then return "" end
7124       local buffer = {}
7125       for _,item in ipairs(items) do
7126         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
7127       end
7128       if tight and tight_lists then
7129         return {"\\markdownRendererDlBeginTight\\n", buffer,
7130               "\\n\\markdownRendererDlEndTight"}
7131       else
7132         return {"\\markdownRendererDlBegin\\n", buffer,
7133               "\\n\\markdownRendererDlEnd"}
7134       end
7135     end
7136   end, extend_reader = function(self)
7137     local parsers = self.parsers
7138     local writer = self.writer
7139
7140     local defstartchar = S("~:")
7141
7142     local defstart = ( defstartchar * #parsers.spacing
7143                       * (parsers.tab + parsers.space~-
7144 3)
7145                       + parsers.space * defstartchar * #parsers.spacing
7146                       * (parsers.tab + parsers.space~-
7147 2)
7148                       + parsers.space * parsers.space * defstartchar
7149                       * #parsers.spacing

```



```

7148                                     * (parsers.tab + parsers.space^-
1)
7149                                     + parsers.space * parsers.space * parsers.space
7150                                     * defstartchar * #parsers.spacing
7151                                 )
7152
7153     local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
7154
7155     local function definition_list_item(term, defs, _)
7156         return { term = self.parser_functions.parse_inlines(term),
7157                 definitions = defs }
7158     end
7159
7160     local DefinitionListItemLoose
7161         = C(parsers.line) * parsers.skipblanklines
7162         * Ct((defstart
7163             * parsers.indented_blocks(dlchunk)
7164             / self.parser_functions.parse_blocks_nested)^1)
7165         * Cc(false) / definition_list_item
7166
7167     local DefinitionListItemTight
7168         = C(parsers.line)
7169         * Ct((defstart * dlchunk
7170             / self.parser_functions.parse_blocks_nested)^1)
7171         * Cc(true) / definition_list_item
7172
7173     local DefinitionList
7174         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
7175           + Ct(DefinitionListItemTight^1)
7176           * (parsers.skipblanklines
7177             * -DefinitionListItemLoose * Cc(true))
7178           ) / writer.definitionlist
7179
7180     self.insert_pattern("Block after Heading",
7181                       DefinitionList, "DefinitionList")
7182 end
7183 }
7184 end

```

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

```

7185 M.extensions.fancy_lists = function()
7186     return {
7187         name = "built-in fancy_lists syntax extension",
7188         extend_writer = function(self)
7189             local options = self.options

```

7190

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.

```
7191     function self.fancylist(items,tight,startnum,numstyle,numdelim)
7192     if not self.is_writing then return "" end
7193     local buffer = {}
7194     local num = startnum
7195     for _,item in ipairs(items) do
7196         buffer[#buffer + 1] = self.fancyitem(item,num)
7197         if num ~= nil then
7198             num = num + 1
7199         end
7200     end
7201     local contents = util.intersperse(buffer,"\n")
7202     if tight and options.tightLists then
7203         return {"\markdownRenderFancyOlBeginTight{",
7204             numstyle,"}{",numdelim,"}",contents,
7205             "\n\markdownRenderFancyOlEndTight "}
7206     else
7207         return {"\markdownRenderFancyOlBegin{",
7208             numstyle,"}{",numdelim,"}",contents,
7209             "\n\markdownRenderFancyOlEnd "}
7210     end
7211 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.

```
7212     function self.fancyitem(s,num)
7213     if num ~= nil then
```

```

7214         return {"\\markdownRendererFancyOlItemWithNumber{" ,num,"} ",s,
7215                 "\\markdownRendererFancyOlItemEnd "}
7216     else
7217         return {"\\markdownRendererFancyOlItem " ,s,"\\markdownRendererFancyOlItemEnd "}
7218     end
7219 end
7220 end, extend_reader = function(self)
7221     local parsers = self.parsers
7222     local options = self.options
7223     local writer = self.writer
7224
7225     local label = parsers.dig + parsers.letter
7226     local numdelim = parsers.period + parsers.rparent
7227     local enumerator = C(label^3 * numdelim) * #parsers.spacing
7228                     + C(label^2 * numdelim) * #parsers.spacing
7229                     * (parsers.tab + parsers.space^1)
7230                     + C(label * numdelim) * #parsers.spacing
7231                     * (parsers.tab + parsers.space^-
2)
7232                     + parsers.space * C(label^2 * numdelim)
7233                     * #parsers.spacing
7234                     + parsers.space * C(label * numdelim)
7235                     * #parsers.spacing
7236                     * (parsers.tab + parsers.space^-
1)
7237                     + parsers.space * parsers.space * C(label^1
7238                     * numdelim) * #parsers.spacing
7239     local starter = parsers.bullet + enumerator
7240
7241     local NestedList = Cs((parsers.optionallyindentedline
7242                         - starter)^1)
7243                         / function(a) return "\\001"..a end
7244
7245     local ListBlockLine = parsers.optionallyindentedline
7246                         - parsers.blankline - (parsers.indent^-1
7247                         * starter)
7248
7249     local ListBlock = parsers.line * ListBlockLine^0
7250
7251     local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
7252                         * ListBlock
7253
7254     local TightListItem = function(starter)
7255         return -parsers.ThematicBreak
7256                 * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
7257                 / self.parser_functions.parse_blocks_nested)

```

```

7258             * -(parsers.blanklines * parsers.indent)
7259     end
7260
7261     local LooseListItem = function(starter)
7262         return -parsers.ThematicBreak
7263             * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
7264                 * (NestedList + ListContinuationBlock^0)
7265                 * (parsers.blanklines / "\n\n")
7266                 ) / self.parser_functions.parse_blocks_nested
7267     end
7268
7269     local function roman2number(roman)
7270         local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
7271         local numeral = 0
7272
7273         local i = 1
7274         local len = string.len(roman)
7275         while i < len do
7276             local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
7277                 if z1 < z2 then
7278                     numeral = numeral + (z2 - z1)
7279                     i = i + 2
7280                 else
7281                     numeral = numeral + z1
7282                     i = i + 1
7283                 end
7284             end
7285             if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
7286             return numeral
7287         end
7288
7289         local function sniffstyle(itemprefix)
7290             local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])*")
7291             local numdelim
7292             if delimend == ")" then
7293                 numdelim = "OneParen"
7294             elseif delimend == "." then
7295                 numdelim = "Period"
7296             else
7297                 numdelim = "Default"
7298             end
7299             numstr = numstr or itemprefix
7300
7301             local num
7302             num = numstr:match("^([IVXL]+)")
7303             if num then
7304                 return roman2number(num), "UpperRoman", numdelim

```

```

7305     end
7306     num = numstr:match("^([ivxl]+)")
7307     if num then
7308         return roman2number(string.upper(num)), "LowerRoman", numdelim
7309     end
7310     num = numstr:match("^([A-Z])")
7311     if num then
7312         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
7313     end
7314     num = numstr:match("^([a-z])")
7315     if num then
7316         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
7317     end
7318     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
7319 end
7320
7321 local function fancylist(items,tight,start)
7322     local startnum, numstyle, numdelim = sniffstyle(start)
7323     return writer.fancylist(items,tight,
7324                             options.startNumber and startnum,
7325                             numstyle or "Decimal",
7326                             numdelim or "Default")
7327 end
7328
7329 local FancyList = Cg(enumerator, "listtype") *
7330     ( Ct(TightListItem(Cb("listtype")))
7331       * TightListItem(enumerator)^0)
7332   * Cc(true) * parsers.skipblanklines * -enumerator
7333   + Ct(LooseListItem(Cb("listtype")))
7334     * LooseListItem(enumerator)^0)
7335   * Cc(false) * parsers.skipblanklines
7336   ) * Cb("listtype") / fancylist
7337
7338 self.update_rule("OrderedList", FancyList)
7339 end
7340 }
7341 end

```

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

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

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

```

7342 M.extensions.fenced_code = function(blank_before_code_fence,
7343                                     allow_attributes,
7344                                     allow_raw_blocks)
7345   return {
7346     name = "built-in fenced_code syntax extension",
7347     extend_writer = function(self)
7348       local options = self.options
7349

```

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.

```

7350     function self.fencedCode(s, i, attr)
7351       if not self.is_writing then return "" end
7352       s = s:gsub("\n$", "")
7353       local buf = {}
7354       if attr ~= nil then
7355         table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
7356                           self.attributes(attr)})
7357       end
7358       local name = util.cache_verbatim(options.cacheDir, s)
7359       table.insert(buf, {"\\markdownRendererInputFencedCode{",
7360                           name,"}{",self.string(i),"}")
7361       if attr ~= nil then
7362         table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd")
7363       end
7364       return buf
7365     end
7366

```

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

```

7367     if allow_raw_blocks then
7368       function self.rawBlock(s, attr)
7369         if not self.is_writing then return "" end
7370         s = s:gsub("\n$", "")
7371         local name = util.cache_verbatim(options.cacheDir, s)
7372         return {"\\markdownRendererInputRawBlock{",
7373                 name,"}{", self.string(attr),""}
7374       end
7375     end
7376   end, extend_reader = function(self)
7377     local parsers = self.parsers
7378     local writer = self.writer
7379
7380     local function captures_geq_length(_,i,a,b)

```

```

7381         return #a >= #b and i
7382     end
7383
7384     local tilde_infostring
7385         = C((parsers.linechar
7386             - (parsers.spacechar^1 * parsers.newline))^0)
7387
7388     local backtick_infostring
7389         = C((parsers.linechar
7390             - (parsers.backtick
7391               + parsers.spacechar^1 * parsers.newline))^0)
7392
7393     local fenceindent
7394     local fencehead      = function(char, infostring)
7395         return           C(parsers.nonindentSPACE) / function(s) fenceindent = #s
7396                         * Cg(char^3, "fencelength")
7397                         * parsers.optionalspace
7398                         * infostring
7399                         * (parsers.newline + parsers.eof)
7400     end
7401
7402     local fencetail      = function(char)
7403         return           parsers.nonindentSPACE
7404                         * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
7405                         * parsers.optionalspace * (parsers.newline + parsers.eof)
7406                         + parsers.eof
7407     end
7408
7409     local fencedline     = function(char)
7410         return           C(parsers.line - fencetail(char))
7411                         / function(s)
7412                             local i = 1
7413                             local remaining = fenceindent
7414                             while true do
7415                                 local c = s:sub(i, i)
7416                                 if c == " " and remaining > 0 then
7417                                     remaining = remaining - 1
7418                                     i = i + 1
7419                                 elseif c == "\t" and remaining > 3 then
7420                                     remaining = remaining - 4
7421                                     i = i + 1
7422                                 else
7423                                     break
7424                                 end
7425                             end
7426                         return s:sub(i)
7427     end

```

```

7428     end
7429
7430     local TildeFencedCode
7431         = fencehead(parsers.tilde, tilde_infostring)
7432         * Cs(fencedline(parsers.tilde)^0)
7433         * fencetail(parsers.tilde)
7434
7435     local BacktickFencedCode
7436         = fencehead(parsers.backtick, backtick_infostring)
7437         * Cs(fencedline(parsers.backtick)^0)
7438         * fencetail(parsers.backtick)
7439
7440     local infostring_with_attributes
7441         = Ct(C((parsers.linechar
7442             - ( parsers.optionalspace
7443               * parsers.attributes))^0)
7444             * parsers.optionalspace
7445             * Ct(parsers.attributes))
7446
7447     local FencedCode
7448         = (TildeFencedCode + BacktickFencedCode)
7449         / function(infostring, code)
7450             local expanded_code = self.expandtabs(code)
7451
7452             if allow_raw_blocks then
7453                 local raw_attr = lpeg.match(parsers.raw_attribute,
7454                                         infostring)
7455
7456                 if raw_attr then
7457                     return writer.rawBlock(expanded_code, raw_attr)
7458                 end
7459             end
7460
7461             local attr = nil
7462             if allow_attributes then
7463                 local match = lpeg.match(infostring_with_attributes,
7464                                         infostring)
7465
7466                 if match then
7467                     infostring, attr = table.unpack(match)
7468                 end
7469             end
7470             return writer.fencedCode(expanded_code, infostring, attr)
7471         end
7472
7473     self.insert_pattern("Block after Verbatim",
7474                       FencedCode, "FencedCode")
7475
7476     local fencestart

```



```

7475     if blank_before_code_fence then
7476         fencestart = parsers.fail
7477     else
7478         fencestart = fencehead(parsers.backtick, backtick_infostring)
7479             + fencehead(parsers.tilde, tilde_infostring)
7480     end
7481
7482     self.update_rule("EndlineExceptions", function(previous_pattern)
7483         if previous_pattern == nil then
7484             previous_pattern = parsers.EndlineExceptions
7485         end
7486         return previous_pattern + fencestart
7487     end)
7488
7489     self.add_special_character("`")
7490     self.add_special_character("~")
7491 end
7492 }
7493 end

```

**3.1.6.7 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.

```

7494 M.extensions.fenced_divs = function(blank_before_div_fence)
7495     return {
7496         name = "built-in fenced_divs syntax extension",
7497         extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with attributes `attributes` to the output format.

```

7498         function self.div_begin(attributes)
7499             local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\\n",
7500                 self.attributes(attributes)}
7501             local end_output = {"\\n\\markdownRendererFencedDivAttributeContextEnd "}
7502             return self.push_attributes("div", attributes, start_output, end_output)
7503         end

```

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

```

7504         function self.div_end()
7505             return self.pop_attributes("div")
7506         end
7507     end, extend_reader = function(self)
7508         local parsers = self.parsers
7509         local writer = self.writer

```

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

```
7510     local fenced_div_infostring
7511           = C((parsers.linechar
7512             - ( parsers.spacechar^1
7513               * parsers.colon^1))^1)
7514
7515     local fenced_div_begin = parsers.nonindentspace
7516           * parsers.colon^3
7517           * parsers.optionalspace
7518           * fenced_div_infostring
7519           * ( parsers.spacechar^1
7520             * parsers.colon^1)^0
7521           * parsers.optionalspace
7522           * (parsers.newline + parsers.eof)
7523
7524     local fenced_div_end = parsers.nonindentspace
7525           * parsers.colon^3
7526           * parsers.optionalspace
7527           * (parsers.newline + parsers.eof)
```

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

```
7528     self.initialize_named_group("div_level", "0")
7529
7530     local function increment_div_level(increment)
7531       local function update_div_level(s, i, current_level) -- luacheck: ignore s i
7532         current_level = tonumber(current_level)
7533         local next_level = tostring(current_level + increment)
7534         return true, next_level
7535       end
7536
7537       return Cg( Cmt(Cb("div_level"), update_div_level)
7538         , "div_level")
7539     end
7540
7541     local FencedDiv = fenced_div_begin
7542           / function (infostring)
7543             local attr = lpeg.match(Ct(parsers.attributes), infostring)
7544             if attr == nil then
7545               attr = {"." .. infostring}
7546             end
7547             return attr
7548           end
7549           / writer.div_begin
7550           * increment_div_level(1)
7551           * parsers.skipblanklines
7552           * Ct( (V("Block") - fenced_div_end)^-1
```

```

7553             * ( parsers.blanklines
7554               / function()
7555                 return writer.interblocksep
7556               end
7557             * (V("Block") - fenced_div_end))^0)
7558         * parsers.skipblanklines
7559         * fenced_div_end * increment_div_level(-1)
7560         * (Cc("") / writer.div_end)
7561
7562     self.insert_pattern("Block after Verbatim",
7563                       FencedDiv, "FencedDiv")
7564
7565     self.add_special_character(":".")
7566

```

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

```

7567     local function check_div_level(s, i, current_level) -- luacheck: ignore s i
7568       current_level = tonumber(current_level)
7569       return current_level > 0
7570     end
7571
7572     local is_inside_div = Cmt(Cb("div_level"), check_div_level)
7573     local fencestart = is_inside_div * fenced_div_end
7574
7575     self.update_rule("BlockquoteExceptions", function(previous_pattern)
7576       if previous_pattern == nil then
7577         previous_pattern = parsers.BlockquoteExceptions
7578       end
7579       return previous_pattern + fencestart
7580     end)
7581

```

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.

```

7582     if not blank_before_div_fence then
7583       self.update_rule("EndlineExceptions", function(previous_pattern)
7584         if previous_pattern == nil then
7585           previous_pattern = parsers.EndlineExceptions
7586         end
7587         return previous_pattern + fencestart
7588       end)
7589     end
7590   end
7591 }
7592 end

```

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

```

7593 M.extensions.header_attributes = function()
7594   return {
7595     name = "built-in header_attributes syntax extension",
7596     extend_writer = function()
7597     end, extend_reader = function(self)
7598       local parsers = self.parsers
7599       local writer = self.writer
7600
7601       local AtxHeading = Cg(parsers.heading_start, "level")
7602         * parsers.optionalspace
7603         * (C(((parsers.linechar
7604           - ((parsers.hash^1
7605             * parsers.optionalspace
7606             * parsers.attributes^-1
7607             + parsers.attributes)
7608             * parsers.optionalspace
7609             * parsers.newline)))
7610           * (parsers.linechar
7611             - parsers.hash
7612             - parsers.lbrace)^0)^1)
7613         / self.parser_functions.parse_inlines)
7614       * Cg(Ct(parsers.newline
7615         + (parsers.hash^1
7616           * parsers.optionalspace
7617           * parsers.attributes^-1
7618           + parsers.attributes)
7619           * parsers.optionalspace
7620           * parsers.newline), "attributes")
7621       * Cb("level")
7622       * Cb("attributes")
7623       / writer.heading
7624
7625       local SettextHeading = #(parsers.line * S("="))
7626         * (C(((parsers.linechar
7627           - (parsers.attributes
7628             * parsers.optionalspace
7629             * parsers.newline)))
7630           * (parsers.linechar
7631             - parsers.lbrace)^0)^1)
7632         / self.parser_functions.parse_inlines)
7633       * Cg(Ct(parsers.newline
7634         + (parsers.attributes
7635           * parsers.optionalspace
7636           * parsers.newline)), "attributes")
7637       * parsers.heading_level

```

```

7638             * Cb("attributes")
7639             * parsers.optionalspace
7640             * parsers.newline
7641             / writer.heading
7642
7643     local Heading = AtxHeading + SettextHeading
7644     self.update_rule("Heading", Heading)
7645 end
7646 }
7647 end

```

**3.1.6.9 Inline Code Attributes** The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

7648 M.extensions.inline_code_attributes = function()
7649   return {
7650     name = "built-in inline_code_attributes syntax extension",
7651     extend_writer = function()
7652     end, extend_reader = function(self)
7653       local writer = self.writer
7654
7655       local CodeWithAttributes = parsers.inticks
7656         * Ct(parsers.attributes)
7657         / writer.code
7658
7659       self.insert_pattern("Inline before Code",
7660         CodeWithAttributes,
7661         "CodeWithAttributes")
7662     end
7663   }
7664 end

```

**3.1.6.10 Line Blocks** The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

7665 M.extensions.line_blocks = function()
7666   return {
7667     name = "built-in line_blocks syntax extension",
7668     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.

```

7669     function self.lineblock(lines)
7670       if not self.is_writing then return "" end
7671       local buffer = {}
7672       for i = 1, #lines - 1 do
7673         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
7674       end

```

```

7675         buffer[#buffer + 1] = lines[#lines]
7676
7677         return {"\\markdownRendererLineBlockBegin\\n"
7678             ,buffer,
7679             "\\n\\markdownRendererLineBlockEnd "}
7680     end
7681 end, extend_reader = function(self)
7682     local parsers = self.parsers
7683     local writer = self.writer
7684
7685     local LineBlock = Ct(
7686         (Cs(
7687             ( (parsers.pipe * parsers.space)/""
7688             * ((parsers.space)/entities.char_entity("nbsp"))^0
7689             * parsers.linechar^0 * (parsers.newline/"")
7690             * (-parsers.pipe
7691             * (parsers.space^1/" ")
7692             * parsers.linechar^1
7693             * (parsers.newline/"")
7694             )^0
7695             * (parsers.blankline/"")^0
7696             ) / self.parser_functions.parse_inlines)^1) / writer.lineblock
7697
7698     self.insert_pattern("Block after Blockquote",
7699         LineBlock, "LineBlock")
7700 end
7701 }
7702 end

```

**3.1.6.11 Link Attributes** The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

7703 M.extensions.link_attributes = function()
7704     return {
7705         name = "built-in link_attributes syntax extension",
7706         extend_writer = function()
7707         end, extend_reader = function(self)
7708             local parsers = self.parsers
7709             local writer = self.writer
7710             local options = self.options
7711

```

The following patterns define link reference definitions with attributes.

```

7712
7713     local define_reference_parser = parsers.define_reference_parser
7714         * ( parsers.spnl
7715         * Ct(parsers.attributes))^~1
7716

```

```

7717     local ReferenceWithAttributes = define_reference_parser
7718                                     * parsers.blankline~1
7719                                     / self.register_link
7720
7721     self.update_rule("Reference", ReferenceWithAttributes)
7722

```

The following patterns define direct and indirect links with attributes.

```

7723
7724     local function indirect_link(label, sps, tag,
7725                                 attribute_text,
7726                                 attributes)
7727     return writer.defer_call(function()
7728         local r, fallback = self.lookup_reference(label, sps, tag,
7729                                                     attributes)
7730
7731         if r then
7732             return writer.link(
7733                 self.parser_functions.parse_inlines_no_link(label),
7734                 r.url, r.title, r.attributes)
7735         else
7736             local buf = {fallback}
7737             if attributes then
7738                 table.insert(buf, writer.string(attribute_text))
7739             end
7740             return buf
7741         end
7742     end)
7743
7744     local DirectLinkWithAttributes = parsers.direct_link
7745                                     * (Ct(parsers.attributes))~-1
7746                                     / writer.link
7747
7748     local IndirectLinkWithAttributes = parsers.indirect_link
7749                                     * (C(Ct(parsers.attributes)))~-1
7750                                     / indirect_link
7751
7752     local LinkWithAttributes = DirectLinkWithAttributes
7753                             + IndirectLinkWithAttributes
7754

```

Here, we directly update the `Link` grammar rule to keep the method `reader->parser_functions.parse_inlines_no_link` aware of `LinkWithAttributes` and prevent nested links.

If we used `reader->insert_pattern` instead of `reader->update_rule`, this correspondence would have been lost and link text would be able to contain nested links.

```

7755     self.update_rule("Link", LinkWithAttributes)
7756

```

The following patterns define direct and indirect images with attributes.

```

7757
7758     local function indirect_image(label, sps, tag,
7759                                   attribute_text,
7760                                   attributes)
7761     return writer.defer_call(function()
7762         local r, fallback = self.lookup_reference(label, sps, tag,
7763                                                     attributes)
7764         if r then
7765             return writer.image(writer.string(label),
7766                                 r.url, r.title, r.attributes)
7767         else
7768             local buf = {"!", fallback}
7769             if attributes then
7770                 table.insert(buf, writer.string(attribute_text))
7771             end
7772             return buf
7773         end
7774     end)
7775 end
7776
7777 local DirectImageWithAttributes = parsers.direct_image
7778                                * Ct(parsers.attributes)
7779                                / writer.image
7780
7781 local IndirectImageWithAttributes = parsers.indirect_image
7782                                   * C(Ct(parsers.attributes))
7783                                   / indirect_image
7784
7785 local ImageWithAttributes = DirectImageWithAttributes
7786                             + IndirectImageWithAttributes
7787
7788 self.insert_pattern("Inline before Image",
7789                    ImageWithAttributes,
7790                    "ImageWithAttributes")
7791

```

The following patterns define autolinks with attributes.

```

7792
7793     local AutoLinkUrlWithAttributes
7794         = parsers.auto_link_url
7795         * Ct(parsers.attributes)
7796         / self.auto_link_url
7797
7798     self.insert_pattern("Inline before AutoLinkUrl",

```



```

7799             AutoLinkUrlWithAttributes,
7800             "AutoLinkUrlWithAttributes")
7801
7802     local AutoLinkEmailWithAttributes
7803         = parsers.auto_link_email
7804         * Ct(parsers.attributes)
7805         / self.auto_link_email
7806
7807     self.insert_pattern("Inline before AutoLinkEmail",
7808                        AutoLinkEmailWithAttributes,
7809                        "AutoLinkEmailWithAttributes")
7810
7811     if options.relativeReferences then
7812
7813         local AutoLinkRelativeReferenceWithAttributes
7814             = parsers.auto_link_relative_reference
7815             * Ct(parsers.attributes)
7816             / self.auto_link_url
7817
7818         self.insert_pattern(
7819             "Inline before AutoLinkRelativeReference",
7820             AutoLinkRelativeReferenceWithAttributes,
7821             "AutoLinkRelativeReferenceWithAttributes")
7822
7823     end
7824
7825 end
7826 }
7827 end

```

**3.1.6.12 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.

```

7828 M.extensions.notes = function(notes, inline_notes)
7829     assert(notes or inline_notes)
7830     return {
7831         name = "built-in notes syntax extension",
7832         extend_writer = function(self)

```

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

```

7833         function self.note(s)
7834             return {"\\markdownRendererNote{",s,""}
7835         end
7836     end, extend_reader = function(self)

```

```

7837     local parsers = self.parsers
7838     local writer = self.writer
7839
7840     if inline_notes then
7841         local InlineNote
7842             = parsers.circumflex
7843             * (parsers.tag / self.parser_functions.parse_inlines_no_inline_no
7844             / writer.note
7845
7846         self.insert_pattern("Inline after Emph",
7847             InlineNote, "InlineNote")
7848     end
7849     if notes then
7850         local function strip_first_char(s)
7851             return s:sub(2)
7852         end
7853
7854         local RawNoteRef
7855             = #(parsers.lbracket * parsers.circumflex)
7856             * parsers.tag / strip_first_char
7857
7858         local rawnotes = {}
7859
7860         -- like indirect_link
7861         local function lookup_note(ref)
7862             return writer.defer_call(function()
7863                 local found = rawnotes[self.normalize_tag(ref)]
7864                 if found then
7865                     return writer.note(
7866                         self.parser_functions.parse_blocks_nested(found))
7867                 else
7868                     return {"[",
7869                         self.parser_functions.parse_inlines("^" .. ref), "]" }
7870                 end
7871             end)
7872         end
7873
7874         local function register_note(ref, rawnote)
7875             rawnotes[self.normalize_tag(ref)] = rawnote
7876             return ""
7877         end
7878
7879         local NoteRef = RawNoteRef / lookup_note
7880
7881         local NoteBlock
7882             = parsers.leader * RawNoteRef * parsers.colon
7883             * parsers.spnl * parsers.indented_blocks(parsers.chunk)

```

```

7884             / register_note
7885
7886         local Blank = NoteBlock + parsers.Blank
7887         self.update_rule("Blank", Blank)
7888
7889         self.insert_pattern("Inline after Emph",
7890                             NoteRef, "NoteRef")
7891     end
7892
7893     self.add_special_character("^")
7894 end
7895 }
7896 end

```

**3.1.6.13 Pipe Tables** The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions.

```

7897 M.extensions.pipe_tables = function(table_captions)
7898
7899     local function make_pipe_table_rectangular(rows)
7900         local num_columns = #rows[2]
7901         local rectangular_rows = {}
7902         for i = 1, #rows do
7903             local row = rows[i]
7904             local rectangular_row = {}
7905             for j = 1, num_columns do
7906                 rectangular_row[j] = row[j] or ""
7907             end
7908             table.insert(rectangular_rows, rectangular_row)
7909         end
7910         return rectangular_rows
7911     end
7912
7913     local function pipe_table_row(allow_empty_first_column
7914                                   , nonempty_column
7915                                   , column_separator
7916                                   , column)
7917         local row_beginning
7918         if allow_empty_first_column then
7919             row_beginning = -- empty first column
7920                             #(parsers.spacechar^4
7921                               * column_separator)
7922                             * parsers.optionalspace
7923                             * column
7924                             * parsers.optionalspace

```

```

7925         -- non-empty first column
7926         + parsers.nonindentspace
7927         * nonempty_column^-1
7928         * parsers.optionalspace
7929     else
7930         row_beginning = parsers.nonindentspace
7931         * nonempty_column^-1
7932         * parsers.optionalspace
7933     end
7934
7935     return Ct(row_beginning
7936         * (-- single column with no leading pipes
7937           #(column_separator
7938             * parsers.optionalspace
7939             * parsers.newline)
7940           * column_separator
7941           * parsers.optionalspace
7942           -- single column with leading pipes or
7943           -- more than a single column
7944           + (column_separator
7945             * parsers.optionalspace
7946             * column
7947             * parsers.optionalspace)^1
7948           * (column_separator
7949             * parsers.optionalspace)^-1))
7950     end
7951
7952     return {
7953         name = "built-in pipe_tables syntax extension",
7954         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.

```

7955     function self.table(rows, caption)
7956         if not self.is_writing then return "" end
7957         local buffer = {"\\markdownRendererTable{",
7958             caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}" }
7959         local temp = rows[2] -- put alignments on the first row
7960         rows[2] = rows[1]
7961         rows[1] = temp
7962         for i, row in ipairs(rows) do
7963             table.insert(buffer, "{")
7964             for _, column in ipairs(row) do
7965                 if i > 1 then -- do not use braces for alignments
7966                     table.insert(buffer, "{")
7967                 end

```

```

7968         table.insert(buffer, column)
7969         if i > 1 then
7970             table.insert(buffer, "|")
7971         end
7972     end
7973     table.insert(buffer, "|")
7974 end
7975 return buffer
7976 end
7977 end, extend_reader = function(self)
7978     local parsers = self.parsers
7979     local writer = self.writer
7980
7981     local table_hline_separator = parsers.pipe + parsers.plus
7982
7983     local table_hline_column = (parsers.dash
7984         - #(parsers.dash
7985             * (parsers.spacechar
7986                 + table_hline_separator
7987                 + parsers.newline)))^1
7988     * (parsers.colon * Cc("r")
7989         + parsers.dash * Cc("d"))
7990     + parsers.colon
7991     * (parsers.dash
7992         - #(parsers.dash
7993             * (parsers.spacechar
7994                 + table_hline_separator
7995                 + parsers.newline)))^1
7996     * (parsers.colon * Cc("c")
7997         + parsers.dash * Cc("l"))
7998
7999     local table_hline = pipe_table_row(false
8000         , table_hline_column
8001         , table_hline_separator
8002         , table_hline_column)
8003
8004     local table_caption_beginning = parsers.skipblanklines
8005         * parsers.nonindentSPACE
8006         * (P("Table")^-1 * parsers.colon)
8007         * parsers.optionalspace
8008
8009     local table_row = pipe_table_row(true
8010         , (C((parsers.linechar - parsers.pipe)^1)
8011             / self.parser_functions.parse_inlines)
8012         , parsers.pipe
8013         , (C((parsers.linechar - parsers.pipe)^0)
8014             / self.parser_functions.parse_inlines))

```

```

8015
8016     local table_caption
8017     if table_captions then
8018         table_caption = #table_caption_beginning
8019                         * table_caption_beginning
8020                         * Ct(parsers.IndentedInline^1)
8021                         * parsers.newline
8022     else
8023         table_caption = parsers.fail
8024     end
8025
8026     local PipeTable = Ct(table_row * parsers.newline
8027                         * table_hline
8028                         * (parsers.newline * table_row)^0)
8029                        / make_pipe_table_rectangular
8030                        * table_caption^-1
8031                        / writer.table
8032
8033     self.insert_pattern("Block after Blockquote",
8034                        PipeTable, "PipeTable")
8035 end
8036 }
8037 end

```

**3.1.6.14 Raw Attributes** The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

8038 M.extensions.raw_inline = function()
8039     return {
8040         name = "built-in raw_inline syntax extension",
8041         extend_writer = function(self)
8042             local options = self.options
8043

```

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

```

8044     function self.rawInline(s, attr)
8045         if not self.is_writing then return "" end
8046         local name = util.cache_verbatim(options.cacheDir, s)
8047         return {"\\markdownRendererInputRawInline{",
8048                name,"}{" , self.string(attr),"}" }
8049     end
8050 end, extend_reader = function(self)
8051     local writer = self.writer
8052
8053     local RawInline = parsers.inticks
8054                        * parsers.raw_attribute
8055                        / writer.rawInline

```

```

8056
8057     self.insert_pattern("Inline before Code",
8058                           RawInline, "RawInline")
8059   end
8060 }
8061 end

```

**3.1.6.15 Strike-Through** The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

8062 M.extensions.strike_through = function()
8063   return {
8064     name = "built-in strike_through syntax extension",
8065     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.

```

8066     function self.strike_through(s)
8067       return {"\\markdownRendererStrikeThrough{" ,s,"}"}
8068     end
8069   end, extend_reader = function(self)
8070     local parsers = self.parsers
8071     local writer = self.writer
8072
8073     local StrikeThrough = (
8074       parsers.between(parsers.Inline, parsers.doubletildes,
8075                     parsers.doubletildes)
8076     ) / writer.strike_through
8077
8078     self.insert_pattern("Inline after Emph",
8079                         StrikeThrough, "StrikeThrough")
8080
8081     self.add_special_character("~")
8082   end
8083 }
8084 end

```

**3.1.6.16 Subscripts** The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

8085 M.extensions.subscripts = function()
8086   return {
8087     name = "built-in subscripts syntax extension",
8088     extend_writer = function(self)

```

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

```

8089     function self.subscript(s)

```

```

8090         return {"\\markdownRendererSubscript{" ,s,""}
8091     end
8092 end, extend_reader = function(self)
8093     local parsers = self.parsers
8094     local writer = self.writer
8095
8096     local Subscript = (
8097         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
8098     ) / writer.subscript
8099
8100     self.insert_pattern("Inline after Emph",
8101                        Subscript, "Subscript")
8102
8103     self.add_special_character("~")
8104 end
8105 }
8106 end

```

**3.1.6.17 Superscripts** The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

8107 M.extensions.superscripts = function()
8108     return {
8109         name = "built-in superscripts syntax extension",
8110         extend_writer = function(self)

```

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

```

8111         function self.superscript(s)
8112             return {"\\markdownRendererSuperscript{" ,s,""}
8113         end
8114     end, extend_reader = function(self)
8115         local parsers = self.parsers
8116         local writer = self.writer
8117
8118         local Superscript = (
8119             parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
8120         ) / writer.superscript
8121
8122         self.insert_pattern("Inline after Emph",
8123                            Superscript, "Superscript")
8124
8125         self.add_special_character("^")
8126     end
8127 }
8128 end

```



**3.1.6.18 Tex Math** The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

8129 M.extensions.tex_math = function(tex_math_dollars,
8130                                   tex_math_single_backslash,
8131                                   tex_math_double_backslash)
8132   return {
8133     name = "built-in tex_math syntax extension",
8134     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.

```

8135     function self.display_math(s)
8136       if not self.is_writing then return "" end
8137       return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
8138     end

```

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

```

8139     function self.inline_math(s)
8140       if not self.is_writing then return "" end
8141       return {"\\markdownRendererInlineMath{",self.math(s),"}"}
8142     end
8143   end, extend_reader = function(self)
8144     local parsers = self.parsers
8145     local writer = self.writer
8146
8147     local function between(p, starter, ender)
8148       return (starter * C(p * (p - ender)^0) * ender)
8149     end
8150
8151     local allowed_before_closing = B( parsers.backslash * parsers.any
8152                                     + parsers.any * (parsers.nonspacechar - parsers

```

The following patterns implement the Pandoc dollar math syntax extension.

```

8153     local dollar_math_content = parsers.backslash^-1
8154                               * parsers.any
8155                               - parsers.blankline^2
8156                               - parsers.dollar
8157
8158     local inline_math_opening_dollars = parsers.dollar
8159                                       * #(parsers.nonspacechar)
8160
8161     local inline_math_closing_dollars = allowed_before_closing
8162                                       * parsers.dollar
8163                                       * -(parsers.digit)
8164
8165     local inline_math_dollars = between(C( dollar_math_content),
8166                                       inline_math_opening_dollars,

```

```

8167             inline_math_closing_dollars)
8168
8169     local display_math_opening_dollars = parsers.dollar
8170                                     * parsers.dollar
8171
8172     local display_math_closing_dollars = parsers.dollar
8173                                     * parsers.dollar
8174
8175     local display_math_dollars = between(C( dollar_math_content),
8176                                         display_math_opening_dollars,
8177                                         display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

8178     local backslash_math_content = parsers.any
8179                                 - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

8180     local inline_math_opening_double = parsers.backslash
8181                                     * parsers.backslash
8182                                     * parsers.lparent
8183                                     * #(parsers.nonspacechar)
8184
8185     local inline_math_closing_double = allowed_before_closing
8186                                     * parsers.backslash
8187                                     * parsers.backslash
8188                                     * parsers.rparent
8189
8190     local inline_math_double = between(C( backslash_math_content),
8191                                       inline_math_opening_double,
8192                                       inline_math_closing_double)
8193
8194     local display_math_opening_double = parsers.backslash
8195                                     * parsers.backslash
8196                                     * parsers.lbracket
8197
8198     local display_math_closing_double = allowed_before_closing
8199                                     * parsers.backslash
8200                                     * parsers.backslash
8201                                     * parsers.rbracket
8202
8203     local display_math_double = between(C( backslash_math_content),
8204                                       display_math_opening_double,
8205                                       display_math_closing_double)

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

8206     local inline_math_opening_single = parsers.backslash

```

```

8207             * parsers.lparent
8208             * #(parsers.nonspacechar)
8209
8210     local inline_math_closing_single = allowed_before_closing
8211             * parsers.backslash
8212             * parsers.rparent
8213
8214     local inline_math_single = between(C( backslash_math_content),
8215                                     inline_math_opening_single,
8216                                     inline_math_closing_single)
8217
8218     local display_math_opening_single = parsers.backslash
8219             * parsers.lbracket
8220
8221     local display_math_closing_single = allowed_before_closing
8222             * parsers.backslash
8223             * parsers.rbracket
8224
8225     local display_math_single = between(C( backslash_math_content),
8226                                     display_math_opening_single,
8227                                     display_math_closing_single)
8228
8229     local display_math = parsers.fail
8230
8231     local inline_math = parsers.fail
8232
8233     if tex_math_dollars then
8234         display_math = display_math + display_math_dollars
8235         inline_math = inline_math + inline_math_dollars
8236     end
8237
8238     if tex_math_double_backslash then
8239         display_math = display_math + display_math_double
8240         inline_math = inline_math + inline_math_double
8241     end
8242
8243     if tex_math_single_backslash then
8244         display_math = display_math + display_math_single
8245         inline_math = inline_math + inline_math_single
8246     end
8247
8248     local TexMath = display_math / writer.display_math
8249             + inline_math / writer.inline_math
8250
8251     self.insert_pattern("Inline after Emph",
8252                       TexMath, "TexMath")
8253

```

```

8254     if tex_math_dollars then
8255         self.add_special_character("$")
8256     end
8257
8258     if tex_math_single_backslash or tex_math_double_backslash then
8259         self.add_special_character("\\")
8260         self.add_special_character("[")
8261         self.add_special_character("]")
8262         self.add_special_character("(")
8263         self.add_special_character("(")
8264     end
8265 end
8266 }
8267 end

```

**3.1.6.19 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.

```

8268 M.extensions.jekyll_data = function(expect_jekyll_data)
8269     return {
8270         name = "built-in jekyll_data syntax extension",
8271         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`.

```

8272     function self.jekyllData(d, t, p)
8273         if not self.is_writing then return "" end
8274
8275         local buf = {}
8276
8277         local keys = {}
8278         for k, _ in pairs(d) do
8279             table.insert(keys, k)
8280         end
8281         table.sort(keys)
8282
8283         if not p then
8284             table.insert(buf, "\\markdownRendererJekyllDataBegin")
8285         end
8286
8287         if #d > 0 then
8288             table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")

```

```

8289         table.insert(buf, self.identifier(p or "null"))
8290         table.insert(buf, "{")
8291         table.insert(buf, #keys)
8292         table.insert(buf, "}")
8293     else
8294         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
8295         table.insert(buf, self.identifier(p or "null"))
8296         table.insert(buf, "{")
8297         table.insert(buf, #keys)
8298         table.insert(buf, "}")
8299     end
8300
8301     for _, k in ipairs(keys) do
8302         local v = d[k]
8303         local typ = type(v)
8304         k = tostring(k or "null")
8305         if typ == "table" and next(v) ~= nil then
8306             table.insert(
8307                 buf,
8308                 self.jekyllData(v, t, k)
8309             )
8310         else
8311             k = self.identifier(k)
8312             v = tostring(v)
8313             if typ == "boolean" then
8314                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
8315                 table.insert(buf, k)
8316                 table.insert(buf, "{")
8317                 table.insert(buf, v)
8318                 table.insert(buf, "}")
8319             elseif typ == "number" then
8320                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
8321                 table.insert(buf, k)
8322                 table.insert(buf, "{")
8323                 table.insert(buf, v)
8324                 table.insert(buf, "}")
8325             elseif typ == "string" then
8326                 table.insert(buf, "\\markdownRendererJekyllDataString{")
8327                 table.insert(buf, k)
8328                 table.insert(buf, "{")
8329                 table.insert(buf, t(v))
8330                 table.insert(buf, "}")
8331             elseif typ == "table" then
8332                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
8333                 table.insert(buf, k)
8334                 table.insert(buf, "}")
8335             else

```

```

8336         error(format("Unexpected type %s for value of " ..
8337                        "YAML key %s", typ, k))
8338     end
8339 end
8340 end
8341
8342 if #d > 0 then
8343     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
8344 else
8345     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
8346 end
8347
8348 if not p then
8349     table.insert(buf, "\\markdownRendererJekyllDataEnd")
8350 end
8351
8352 return buf
8353 end
8354 end, extend_reader = function(self)
8355     local parsers = self.parsers
8356     local writer = self.writer
8357
8358     local JekyllData
8359         = Cmt( C((parsers.line - P("----") - P("..."))^0)
8360               , function(s, i, text) -- luacheck: ignore s i
8361                   local data
8362                   local ran_ok, _ = pcall(function()
8363                       local tinyyaml = require("markdown-tinyyaml")
8364                       data = tinyyaml.parse(text, {timestamps=false})
8365                   end)
8366                   if ran_ok and data ~= nil then
8367                       return true, writer.jekyllData(data, function(s)
8368                           return self.parser_functions.parse_blocks_nested(s)
8369                       end, nil)
8370                   else
8371                       return false
8372                   end
8373               end
8374             )
8375
8376     local UnexpectedJekyllData
8377         = P("----")
8378         * parsers.blankline / 0
8379         * #(-parsers.blankline) -- if followed by blank, it's thematic b
8380         * JekyllData
8381         * (P("----") + P("..."))
8382

```

```

8383     local ExpectedJekyllData
8384         = ( P("----")
8385             * parsers.blankline / 0
8386             * #(-parsers.blankline) -- if followed by blank, it's thematic
8387             )^-1
8388             * JekyllData
8389             * (P("----") + P("..."))^-1
8390
8391     self.insert_pattern("Block before Blockquote",
8392                        UnexpectedJekyllData, "UnexpectedJekyllData")
8393     if expect_jekyll_data then
8394         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
8395     end
8396 end
8397 }
8398 end

```

### 3.1.7 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```

8399 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

8400     options = options or {}
8401     setmetatable(options, { __index = function (_, key)
8402         return defaultOptions[key] end })

```

Apply built-in syntax extensions based on `options`.

```

8403     local extensions = {}
8404
8405     if options.bracketedSpans then
8406         local bracketed_spans_extension = M.extensions.bracketed_spans()
8407         table.insert(extensions, bracketed_spans_extension)
8408     end
8409
8410     if options.contentBlocks then
8411         local content_blocks_extension = M.extensions.content_blocks(
8412             options.contentBlocksLanguageMap)
8413         table.insert(extensions, content_blocks_extension)
8414     end
8415
8416     if options.definitionLists then
8417         local definition_lists_extension = M.extensions.definition_lists(
8418             options.tightLists)
8419         table.insert(extensions, definition_lists_extension)
8420     end

```

```

8421
8422 if options.fencedCode then
8423     local fenced_code_extension = M.extensions.fenced_code(
8424         options.blankBeforeCodeFence,
8425         options.fencedCodeAttributes,
8426         options.rawAttribute)
8427     table.insert(extensions, fenced_code_extension)
8428 end
8429
8430 if options.fencedDivs then
8431     local fenced_div_extension = M.extensions.fenced_divs(
8432         options.blankBeforeDivFence)
8433     table.insert(extensions, fenced_div_extension)
8434 end
8435
8436 if options.headerAttributes then
8437     local header_attributes_extension = M.extensions.header_attributes()
8438     table.insert(extensions, header_attributes_extension)
8439 end
8440
8441 if options.inlineCodeAttributes then
8442     local inline_code_attributes_extension =
8443         M.extensions.inline_code_attributes()
8444     table.insert(extensions, inline_code_attributes_extension)
8445 end
8446
8447 if options.jekyllData then
8448     local jekyll_data_extension = M.extensions.jekyll_data(
8449         options.expectJekyllData)
8450     table.insert(extensions, jekyll_data_extension)
8451 end
8452
8453 if options.linkAttributes then
8454     local link_attributes_extension =
8455         M.extensions.link_attributes()
8456     table.insert(extensions, link_attributes_extension)
8457 end
8458
8459 if options.lineBlocks then
8460     local line_block_extension = M.extensions.line_blocks()
8461     table.insert(extensions, line_block_extension)
8462 end
8463
8464 if options.pipeTables then
8465     local pipe_tables_extension = M.extensions.pipe_tables(
8466         options.tableCaptions)
8467     table.insert(extensions, pipe_tables_extension)

```



```

8468 end
8469
8470 if options.rawAttribute then
8471     local raw_inline_extension = M.extensions.raw_inline()
8472     table.insert(extensions, raw_inline_extension)
8473 end
8474
8475 if options.strikeThrough then
8476     local strike_through_extension = M.extensions.strike_through()
8477     table.insert(extensions, strike_through_extension)
8478 end
8479
8480 if options.subscripts then
8481     local subscript_extension = M.extensions.subscripts()
8482     table.insert(extensions, subscript_extension)
8483 end
8484
8485 if options.superscripts then
8486     local superscript_extension = M.extensions.superscripts()
8487     table.insert(extensions, superscript_extension)
8488 end
8489
8490 if options.texMathDollars or
8491     options.texMathSingleBackslash or
8492     options.texMathDoubleBackslash then
8493     local tex_math_extension = M.extensions.tex_math(
8494         options.texMathDollars,
8495         options.texMathSingleBackslash,
8496         options.texMathDoubleBackslash)
8497     table.insert(extensions, tex_math_extension)
8498 end
8499

```

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

```

8500 if options.footnotes or options.inlineFootnotes or
8501     options.notes or options.inlineNotes then
8502     local notes_extension = M.extensions.notes(
8503         options.footnotes or options.notes,
8504         options.inlineFootnotes or options.inlineNotes)
8505     table.insert(extensions, notes_extension)
8506 end
8507
8508 if options.citations then
8509     local citations_extension = M.extensions.citations(options.citationNbsps)
8510     table.insert(extensions, citations_extension)
8511 end

```

```

8512
8513   if options.fancyLists then
8514       local fancy_lists_extension = M.extensions.fancy_lists()
8515       table.insert(extensions, fancy_lists_extension)
8516   end

  Apply user-defined syntax extensions based on options.extensions.
8517   for _, user_extension_filename in ipairs(options.extensions) do
8518       local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

8519       local pathname = util.lookup_files(filename)
8520       local input_file = assert(io.open(pathname, "r"),
8521           [[Could not open user-defined syntax extension "]]
8522           .. pathname .. [[" for reading]])
8523       local input = assert(input_file:read("*a"))
8524       assert(input_file:close())
8525       local user_extension, err = load([[
8526           local sandbox = {}
8527           setmetatable(sandbox, {__index = _G})
8528           _ENV = sandbox
8529       ]] .. input)()
8530       assert(user_extension,
8531           [[Failed to compile user-defined syntax extension "]]
8532           .. pathname .. [[: ] .. (err or [[]])])

```

Then, validate the user-defined syntax extension.

```

8533       assert(user_extension.api_version ~= nil,
8534           [[User-defined syntax extension "]] .. pathname
8535           .. [[" does not specify mandatory field "api_version"]])
8536       assert(type(user_extension.api_version) == "number",
8537           [[User-defined syntax extension "]] .. pathname
8538           .. [[" specifies field "api_version" of type ]]
8539           .. type(user_extension.api_version)
8540           .. [[" but "number" was expected]])
8541       assert(user_extension.api_version > 0
8542           and user_extension.api_version <= metadata.user_extension_api_version,
8543           [[User-defined syntax extension "]] .. pathname
8544           .. [[" uses syntax extension API version "]]
8545           .. user_extension.api_version .. [[" but markdown.lua ]]
8546           .. metadata.version .. [[" uses API version ]]
8547           .. metadata.user_extension_api_version
8548           .. [[, which is incompatible]])
8549
8550       assert(user_extension.grammar_version ~= nil,
8551           [[User-defined syntax extension "]] .. pathname
8552           .. [[" does not specify mandatory field "grammar_version"]])
8553       assert(type(user_extension.grammar_version) == "number",

```

```

8554     [[User-defined syntax extension "]] .. pathname
8555     .. [[[" specifies field "grammar_version" of type "]]
8556     .. type(user_extension.grammar_version)
8557     .. [[[" but "number" was expected]])
8558     assert(user_extension.grammar_version == metadata.grammar_version,
8559            [[User-defined syntax extension "]] .. pathname
8560            .. [[[" uses grammar version "]] .. user_extension.grammar_version
8561            .. [[[" but markdown.lua ]] .. metadata.version
8562            .. [[[" uses grammar version ]] .. metadata.grammar_version
8563            .. [[[" which is incompatible]])
8564
8565     assert(user_extension.finalize_grammar ~= nil,
8566            [[User-defined syntax extension "]] .. pathname
8567            .. [[[" does not specify mandatory "finalize_grammar" field]])
8568     assert(type(user_extension.finalize_grammar) == "function",
8569            [[User-defined syntax extension "]] .. pathname
8570            .. [[[" specifies field "finalize_grammar" of type "]]
8571            .. type(user_extension.finalize_grammar)
8572            .. [[[" but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.6.)

```

8573     local extension = {
8574         name = [[user-defined "]] .. pathname .. [[[" syntax extension]],
8575         extend_reader = user_extension.finalize_grammar,
8576         extend_writer = function() end,
8577     }
8578     return extension
8579 end)(user_extension_filename)
8580 table.insert(extensions, user_extension)
8581 end

```

Produce and return a conversion function from markdown to plain TeX.

```

8582     local writer = M.writer.new(options)
8583     local reader = M.reader.new(writer, options)
8584     local convert = reader.finalize_grammar(extensions)
8585
8586     return convert
8587 end
8588
8589 return M

```

### 3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.6.

```

8590

```

```

8591 local input
8592 if input_filename then
8593     local input_file = assert(io.open(input_filename, "r"),
8594         [[Could not open file ]] .. input_filename .. [[ for reading]])
8595     input = assert(input_file:read("*a"))
8596     assert(input_file:close())
8597 else
8598     input = assert(io.read("*a"))
8599 end
8600

```

First, ensure that the `options.cacheDir` directory exists.

```

8601 local lfs = require("lfs")
8602 if options.cacheDir and not lfs.isdir(options.cacheDir) then
8603     assert(lfs.mkdir(options["cacheDir"]))
8604 end

```

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

```

8605 local kpse
8606 (function()
8607     local should_initialize = package.loaded.kpse == nil
8608                             or tex.initialize ~= nil
8609     local ran_ok
8610     ran_ok, kpse = pcall(require, "kpse")
8611     if ran_ok and should_initialize then
8612         kpse.set_program_name("luatex")
8613     end
8614 end)()
8615 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.

```

8616 if metadata.version ~= md.metadata.version then
8617     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
8618         "markdown.lua " .. md.metadata.version .. ".")
8619 end
8620 local convert = md.new(options)
8621 local output = convert(input)
8622
8623 if output_filename then
8624     local output_file = assert(io.open(output_filename, "w"),
8625         [[Could not open file ]] .. output_filename .. [[ for writing]])
8626     assert(output_file:write(output))
8627     assert(output_file:close())
8628 else
8629     assert(io.write(output))
8630 end

```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```
8631 \ifx\markdownInfo\undefined
8632   \def\markdownInfo#1{%
8633     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
8634 \fi
8635 \ifx\markdownWarning\undefined
8636   \def\markdownWarning#1{%
8637     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
8638 \fi
8639 \ifx\markdownError\undefined
8640   \def\markdownError#1#2{%
8641     \errhelp{#2.}%
8642     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
8643 \fi
```

### 3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
8644 \def\markdownRendererInterblockSeparatorPrototype{\par}%
8645 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
8646 \let\markdownRendererEllipsisPrototype\dots
8647 \def\markdownRendererNbspPrototype{~}%
8648 \def\markdownRendererLeftBracePrototype{\char`\{}%
8649 \def\markdownRendererRightBracePrototype{\char`\}%
8650 \def\markdownRendererDollarSignPrototype{\char`\$}%
8651 \def\markdownRendererPercentSignPrototype{\char`\}%
8652 \def\markdownRendererAmpersandPrototype{\&}%
8653 \def\markdownRendererUnderscorePrototype{\char`\_}%
8654 \def\markdownRendererHashPrototype{\char`\#}%
8655 \def\markdownRendererCircumflexPrototype{\char`\^}%
8656 \def\markdownRendererBackslashPrototype{\char`\}%
8657 \def\markdownRendererTildePrototype{\char`\~}%
8658 \def\markdownRendererPipePrototype{|}%
8659 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
8660 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
8661 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
8662   \markdownInput{#3}}%
8663 \def\markdownRendererContentBlockOnlineImagePrototype{%
8664   \markdownRendererImage}%
```

```

8665 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
8666   \markdownRendererInputFencedCode{#3}{#2}}%
8667 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
8668 \def\markdownRendererUlBeginPrototype{}%
8669 \def\markdownRendererUlBeginTightPrototype{}%
8670 \def\markdownRendererUlItemPrototype{}%
8671 \def\markdownRendererUlItemEndPrototype{}%
8672 \def\markdownRendererUlEndPrototype{}%
8673 \def\markdownRendererUlEndTightPrototype{}%
8674 \def\markdownRendererOlBeginPrototype{}%
8675 \def\markdownRendererOlBeginTightPrototype{}%
8676 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
8677 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
8678 \def\markdownRendererOlItemPrototype{}%
8679 \def\markdownRendererOlItemWithNumberPrototype#1{}%
8680 \def\markdownRendererOlItemEndPrototype{}%
8681 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
8682 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
8683 \def\markdownRendererFancyOlItemEndPrototype{}%
8684 \def\markdownRendererOlEndPrototype{}%
8685 \def\markdownRendererOlEndTightPrototype{}%
8686 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
8687 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
8688 \def\markdownRendererDlBeginPrototype{}%
8689 \def\markdownRendererDlBeginTightPrototype{}%
8690 \def\markdownRendererDlItemPrototype#1{#1}%
8691 \def\markdownRendererDlItemEndPrototype{}%
8692 \def\markdownRendererDlDefinitionBeginPrototype{}%
8693 \def\markdownRendererDlDefinitionEndPrototype{\par}%
8694 \def\markdownRendererDlEndPrototype{}%
8695 \def\markdownRendererDlEndTightPrototype{}%
8696 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
8697 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
8698 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
8699 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
8700 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
8701 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
8702 \def\markdownRendererInputVerbatimPrototype#1{%
8703   \par{\tt\input#1\relax{}}\par}%
8704 \def\markdownRendererInputFencedCodePrototype#1#2{%
8705   \markdownRendererInputVerbatim{#1}}%
8706 \def\markdownRendererHeadingOnePrototype#1{#1}%
8707 \def\markdownRendererHeadingTwoPrototype#1{#1}%
8708 \def\markdownRendererHeadingThreePrototype#1{#1}%
8709 \def\markdownRendererHeadingFourPrototype#1{#1}%
8710 \def\markdownRendererHeadingFivePrototype#1{#1}%
8711 \def\markdownRendererHeadingSixPrototype#1{#1}%

```

```

8712 \def\markdownRendererThematicBreakPrototype{}%
8713 \def\markdownRendererNotePrototype#1{#1}%
8714 \def\markdownRendererCitePrototype#1{}%
8715 \def\markdownRendererTextCitePrototype#1{}%
8716 \def\markdownRendererTickedBoxPrototype{[X]}%
8717 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
8718 \def\markdownRendererUntickedBoxPrototype{[ ]}%
8719 \def\markdownRendererStrikeThroughPrototype#1{#1}%
8720 \def\markdownRendererSuperscriptPrototype#1{#1}%
8721 \def\markdownRendererSubscriptPrototype#1{#1}%
8722 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
8723 \def\markdownRendererInlineMathPrototype#1{$#1$}%
8724 \ExplSyntaxOn
8725 \cs_gset:Npn
8726   \markdownRendererHeaderAttributeContextBeginPrototype
8727   {
8728     \group_begin:
8729     \color_group_begin:
8730   }
8731 \cs_gset:Npn
8732   \markdownRendererHeaderAttributeContextEndPrototype
8733   {
8734     \color_group_end:
8735     \group_end:
8736   }
8737 \cs_gset_eq:NN
8738   \markdownRendererBracketedSpanAttributeContextBeginPrototype
8739   \markdownRendererHeaderAttributeContextBeginPrototype
8740 \cs_gset_eq:NN
8741   \markdownRendererBracketedSpanAttributeContextEndPrototype
8742   \markdownRendererHeaderAttributeContextEndPrototype
8743 \cs_gset_eq:NN
8744   \markdownRendererFencedDivAttributeContextBeginPrototype
8745   \markdownRendererHeaderAttributeContextBeginPrototype
8746 \cs_gset_eq:NN
8747   \markdownRendererFencedDivAttributeContextEndPrototype
8748   \markdownRendererHeaderAttributeContextEndPrototype
8749 \cs_gset_eq:NN
8750   \markdownRendererFencedCodeAttributeContextBeginPrototype
8751   \markdownRendererHeaderAttributeContextBeginPrototype
8752 \cs_gset_eq:NN
8753   \markdownRendererFencedCodeAttributeContextEndPrototype
8754   \markdownRendererHeaderAttributeContextEndPrototype
8755 \cs_gset:Npn
8756   \markdownRendererReplacementCharacterPrototype
8757   {
8758     % TODO: Replace with \codepoint_generate:nn in TeX Live 2023

```

```

8759 \sys_if_engine_pdftex:TF
8760 { ^^ef^^bf^^bd }
8761 { ^^^^fffd }
8762 }
8763 \ExplSyntaxOff
8764 \def\markdownRendererSectionBeginPrototype{%
8765 \def\markdownRendererSectionEndPrototype{%

```

**3.2.2.1 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.

```

8766 \ExplSyntaxOn
8767 \cs_new:Nn
8768 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
8769 {
8770   \str_case:nn
8771     { #2 }
8772     {
8773       { md } { \markdownInput{#1} }
8774       { tex } { \markdownEscape{#1} \unskip }
8775     }
8776 }
8777 \cs_new:Nn
8778 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
8779 {
8780   \str_case:nn
8781     { #2 }
8782     {
8783       { md } { \markdownInput{#1} }
8784       { tex } { \markdownEscape{#1} }
8785     }
8786 }
8787 \cs_gset:Npn
8788 \markdownRendererInputRawInlinePrototype#1#2
8789 {
8790   \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
8791     { #1 }
8792     { #2 }
8793 }
8794 \cs_gset:Npn
8795 \markdownRendererInputRawBlockPrototype#1#2
8796 {
8797   \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
8798     { #1 }
8799     { #2 }
8800 }

```



8801 \ExplSyntaxOff

**3.2.2.2 YAML Metadata Renderer Prototypes** To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```
8802 \ExplSyntaxOn
8803 \seq_new:N \g_@@_jekyll_data_datatypes_seq
8804 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
8805 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
8806 \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.

```
8807 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
8808 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
8809 {
8810   \seq_if_empty:NF
8811     \g_@@_jekyll_data_datatypes_seq
8812     {
8813       \seq_get_right:NN
8814       \g_@@_jekyll_data_datatypes_seq
8815       \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.

```
8816   \str_if_eq:NNTF
8817     \l_tmpa_tl
8818     \c_@@_jekyll_data_sequence_tl
8819     {
8820       \seq_put_right:Nn
8821       \g_@@_jekyll_data_wildcard_absolute_address_seq
```

```

8822         { * }
8823     }
8824     {
8825         \seq_put_right:Nn
8826         \g_@@_jekyll_data_wildcard_absolute_address_seq
8827         { #1 }
8828     }
8829 }
8830 }

```

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.

```

8831 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
8832 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
8833 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
8834 {
8835     \seq_pop_left:NN #1 \l_tmpa_tl
8836     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
8837     \seq_put_left:NV #1 \l_tmpa_tl
8838 }
8839 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
8840 {

```

```

8841 \markdown_jekyll_data_concatenate_address:Nn
8842   \g_@@_jekyll_data_wildcard_absolute_address_seq
8843   \g_@@_jekyll_data_wildcard_absolute_address_tl
8844 \seq_get_right:NN
8845   \g_@@_jekyll_data_wildcard_absolute_address_seq
8846   \g_@@_jekyll_data_wildcard_relative_address_tl
8847 }

```

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.

```

8848 \cs_new:Nn \markdown_jekyll_data_push:nN
8849 {
8850   \markdown_jekyll_data_push_address_segment:n
8851     { #1 }
8852   \seq_put_right:NV
8853     \g_@@_jekyll_data_datatypes_seq
8854     #2
8855   \markdown_jekyll_data_update_address_tls:
8856 }
8857 \cs_new:Nn \markdown_jekyll_data_pop:
8858 {
8859   \seq_pop_right:NN
8860     \g_@@_jekyll_data_wildcard_absolute_address_seq
8861     \l_tmpa_tl
8862   \seq_pop_right:NN
8863     \g_@@_jekyll_data_datatypes_seq
8864     \l_tmpa_tl
8865   \markdown_jekyll_data_update_address_tls:
8866 }

```

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.

```

8867 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
8868 {
8869   \keys_set_known:nn
8870     { markdown/jekyllData }
8871     { { #1 } = { #2 } }
8872 }
8873 \cs_generate_variant:Nn
8874   \markdown_jekyll_data_set_keyval:nn
8875   { Vn }
8876 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
8877 {
8878   \markdown_jekyll_data_push:nN
8879     { #1 }
8880     \c_@@_jekyll_data_scalar_tl
8881   \markdown_jekyll_data_set_keyval:Vn

```

```

8882     \g_@@_jekyll_data_wildcard_absolute_address_tl
8883     { #2 }
8884     \markdown_jekyll_data_set_keyval:Vn
8885     \g_@@_jekyll_data_wildcard_relative_address_tl
8886     { #2 }
8887     \markdown_jekyll_data_pop:
8888 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

8889 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
8890     \markdown_jekyll_data_push:nN
8891     { #1 }
8892     \c_@@_jekyll_data_sequence_tl
8893 }
8894 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
8895     \markdown_jekyll_data_push:nN
8896     { #1 }
8897     \c_@@_jekyll_data_mapping_tl
8898 }
8899 \def\markdownRendererJekyllDataSequenceEndPrototype{
8900     \markdown_jekyll_data_pop:
8901 }
8902 \def\markdownRendererJekyllDataMappingEndPrototype{
8903     \markdown_jekyll_data_pop:
8904 }
8905 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
8906     \markdown_jekyll_data_set_keyvals:nn
8907     { #1 }
8908     { #2 }
8909 }
8910 \def\markdownRendererJekyllDataEmptyPrototype#1{}
8911 \def\markdownRendererJekyllDataNumberPrototype#1#2{
8912     \markdown_jekyll_data_set_keyvals:nn
8913     { #1 }
8914     { #2 }
8915 }
8916 \def\markdownRendererJekyllDataStringPrototype#1#2{
8917     \markdown_jekyll_data_set_keyvals:nn
8918     { #1 }
8919     { #2 }
8920 }
8921 \ExplSyntaxOff

```

### 3.2.3 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the

`\markdownLuaOptions` macro will expand to a Lua table that contains the plain TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

8922 \ExplSyntaxOn
8923 \tl_new:N \g_@@_formatted_lua_options_tl
8924 \cs_new:Nn \@@_format_lua_options:
8925 {
8926   \tl_gclear:N
8927   \g_@@_formatted_lua_options_tl
8928   \seq_map_function:NN
8929   \g_@@_lua_options_seq
8930   \@@_format_lua_option:n
8931 }
8932 \cs_new:Nn \@@_format_lua_option:n
8933 {
8934   \@@_typecheck_option:n
8935   { #1 }
8936   \@@_get_option_type:nN
8937   { #1 }
8938   \l_tmpa_tl
8939   \bool_case_true:nF
8940   {
8941     {
8942       \str_if_eq_p:VV
8943       \l_tmpa_tl
8944       \c_@@_option_type_boolean_tl ||
8945       \str_if_eq_p:VV
8946       \l_tmpa_tl
8947       \c_@@_option_type_number_tl ||
8948       \str_if_eq_p:VV
8949       \l_tmpa_tl
8950       \c_@@_option_type_counter_tl
8951     }
8952     {
8953       \@@_get_option_value:nN
8954       { #1 }
8955       \l_tmpa_tl
8956       \tl_gput_right:Nx
8957       \g_@@_formatted_lua_options_tl
8958       { #1~== \l_tmpa_tl ,~ }
8959     }
8960   }
8961   \str_if_eq_p:VV
8962   \l_tmpa_tl
8963   \c_@@_option_type_clist_tl
8964 }
8965 {
8966   \@@_get_option_value:nN

```

```

8967         { #1 }
8968         \l_tmpa_tl
8969         \tl_gput_right:Nx
8970         \g_@@_formatted_lua_options_tl
8971         { #1~::~\c_left_brace_str }
8972         \clist_map_inline:Nn
8973         \l_tmpa_tl
8974         {
8975             \tl_gput_right:Nx
8976             \g_@@_formatted_lua_options_tl
8977             { "##1" ,~ }
8978         }
8979         \tl_gput_right:Nx
8980         \g_@@_formatted_lua_options_tl
8981         { \c_right_brace_str ,~ }
8982     }
8983 }
8984 {
8985     \@@_get_option_value:nN
8986     { #1 }
8987     \l_tmpa_tl
8988     \tl_gput_right:Nx
8989     \g_@@_formatted_lua_options_tl
8990     { #1~::~ " \l_tmpa_tl " ,~ }
8991 }
8992 }
8993 \cs_generate_variant:Nn
8994 \clist_map_inline:nn
8995 { Vn }
8996 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
8997 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
8998 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```
8999 \def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```

9000     local lfs = require("lfs")
9001     local cacheDir = "\markdownOptionCacheDir"
9002     if not lfs.isdir(cacheDir) then
9003         assert(lfs.mkdir(cacheDir))
9004     end

```

Next, load the `markdown` module and create a converter function using the plain T<sub>E</sub>X options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
9005     local md = require("markdown")
```

```

9006   local convert = md.new(\markdownLuaOptions)
9007 }%

```

### 3.2.4 Buffering Markdown Input

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

```

9008 \ExplSyntaxOn
9009 \cs_new:Nn
9010   \@@_if_option:nTF
9011   {
9012     \@@_get_option_type:nN
9013     { #1 }
9014     \l_tmpa_tl
9015     \str_if_eq:NNF
9016     \l_tmpa_tl
9017     \c_@@_option_type_boolean_tl
9018     {
9019       \msg_error:nnxx
9020       { @@ }
9021       { expected-boolean-option }
9022       { #1 }
9023       { \l_tmpa_tl }
9024     }
9025     \@@_get_option_value:nN
9026     { #1 }
9027     \l_tmpa_tl
9028     \str_if_eq:NNTF
9029     \l_tmpa_tl
9030     \c_@@_option_value_true_tl
9031     { #2 }
9032     { #3 }
9033   }
9034 \msg_new:nnn
9035   { @@ }
9036   { expected-boolean-option }
9037   {
9038     Option~#1~has~type~#2,~
9039     but~a~boolean~was~expected.
9040   }
9041 \let\markdownIfOption=\@@_if_option:nTF
9042 \ExplSyntaxOff

```

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```

9043 \csname newread\endcsname\markdownInputFileStream
9044 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

9045 \begingroup
9046 \catcode\^^I=12%
9047 \gdef\markdownReadAndConvertTab{^^I}%
9048 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

9049 \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.

```

9050 \catcode\^^M=13%
9051 \catcode\^^I=13%
9052 \catcode|=0%
9053 \catcode\=12%
9054 |catcode@=14%
9055 |catcode|=12@
9056 |gdef|markdownReadAndConvert#1#2{@
9057 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

9058 |markdownIfOption{frozenCache}{-}{@
9059 |immediate|openout|markdownOutputFileStream@
9060 |markdownOptionInputTempFileName|relax@
9061 |markdownInfo{Buffering markdown input into the temporary @
9062 |input file "|markdownOptionInputTempFileName" and scanning @
9063 |for the closing token sequence "#1"}@
9064 }@

```

Locally change the category of the special plain  $\text{\TeX}$  characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

9065 |def|do##1{|catcode`##1=12}|dospecials@
9066 |catcode`|=12@
9067 |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (`%`) when `stripPercentSigns` is enabled. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.



```

9068 |def|markdownReadAndConvertStripPercentSign##1{@
9069 |markdownIfOption{stripPercentSigns}{@
9070 |if##1%@
9071 |expandafter|expandafter|expandafter@
9072 |markdownReadAndConvertProcessLine@
9073 |else@
9074 |expandafter|expandafter|expandafter@
9075 |markdownReadAndConvertProcessLine@
9076 |expandafter|expandafter|expandafter##1@
9077 |fi@
9078 }{@
9079 |expandafter@
9080 |markdownReadAndConvertProcessLine@
9081 |expandafter##1@
9082 }@
9083 }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

9084 |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.

```

9085 |ifx|relax##3|relax@
9086 |markdownIfOption{frozenCache}{}{@
9087 |immediate|write|markdownOutputFileStream{##1}@
9088 }@
9089 |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

9090 |def^^M{@
9091 |markdownInfo{The ending token sequence was found}@
9092 |markdownIfOption{frozenCache}{}{@
9093 |immediate|closeout|markdownOutputFileStream@
9094 }@
9095 |endgroup@
9096 |markdownInput{@
9097 |markdownOptionOutputDir@
9098 /|markdownOptionInputTempFileName@
9099 }@
9100 #2}@

```

```
9101      |fi@
```

Repeat with the next line.

```
9102      ^^M}@
```

Make the tab character active at expansion time and make it expand to a literal tab character.

```
9103      |catcode`\^^I=13@
```

```
9104      |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.

```
9105      |catcode`\^^M=13@
```

```
9106      |def^^M##1^^M{@
```

```
9107          |def^^M###1^^M{@
```

```
9108              |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
```

```
9109      ^^M}@
```

```
9110      ^^M}@
```

Reset the character categories back to the former state.

```
9111 |endgroup
```

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

```
9112 \ExplSyntaxOn
```

```
9113 \int_compare:nT
```

```
9114   { \markdownMode = 3 }
```

```
9115   {
```

```
9116       \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
```

```
9117       \file_input:n { lt3luabridge.tex }
```

```
9118       \cs_new:Npn
```

```
9119           \markdownLuaExecute
```

```
9120           { \luabridgeExecute }
```

```
9121   }
```

```
9122 \ExplSyntaxOff
```

### 3.2.5 Lua Shell Escape Bridge

The following  $\text{\TeX}$  code is intended for  $\text{\TeX}$  engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the Lua $\text{\TeX}$  engine, their  $\text{\TeX}$  distribution contains it, and uses shell access to produce and execute Lua scripts using the  $\text{\TeX}$ Lua interpreter [1, Section 4.1.1].

```

9123 \ifnum\markdownMode<2\relax
9124 \ifnum\markdownMode=0\relax
9125   \markdownWarning{Using mode 0: Shell escape via write18
9126                     (deprecated, to be removed in Markdown 3.0.0)}%
9127 \else
9128   \markdownWarning{Using mode 1: Shell escape via os.execute
9129                     (deprecated, to be removed in Markdown 3.0.0)}%
9130 \fi

```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (1), disabled (0), or restricted (2).

Inherit the value of the `\pdfshellescape` (Lua<sub>TeX</sub>, Pdf<sub>TeX</sub>) or the `\shellescape` (X<sub>Y</sub>TeX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```

9131 \ifx\pdfshellescape\undefined
9132   \ifx\shellescape\undefined
9133     \ifnum\markdownMode=0\relax
9134       \def\markdownExecuteShellEscape{1}%
9135     \else
9136       \def\markdownExecuteShellEscape{%
9137         \directlua{tex.sprint(status.shell_escape or "1")}}%
9138     \fi
9139   \else
9140     \let\markdownExecuteShellEscape\shellescape
9141   \fi
9142 \else
9143   \let\markdownExecuteShellEscape\pdfshellescape
9144 \fi

```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```

9145 \ifnum\markdownMode=0\relax
9146   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
9147 \else
9148   \def\markdownExecuteDirect#1{%
9149     \directlua{os.execute("\luaescapestring{#1}")}}%
9150 \fi

```

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

```

9151 \def\markdownExecute#1{%
9152   \ifnum\markdownExecuteShellEscape=1\relax
9153     \markdownExecuteDirect{#1}%
9154   \else

```

```

9155     \markdownError{I can not access the shell}{Either run the TeX
9156         compiler with the --shell-escape or the --enable-write18 flag,
9157         or set shell_escape=t in the texmf.cnf file}%
9158     \fi}%

```

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TeX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

```

9159 \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.

```

9160     \catcode`\|=0%
9161     \catcode`\|=12%
9162     \gdef\markdownLuaExecute#1{%

```

Create the file `helperScriptFileName` and fill it with the input Lua code prepended with kpathsea initialization, so that Lua modules from the TeX distribution are available.

```

9163         |immediate|openout|markdownOutputFileStream=%
9164         |markdownOptionHelperScriptFileName
9165         |markdownInfo{Writing a helper Lua script to the file
9166         " |markdownOptionHelperScriptFileName"}%
9167         |immediate|write|markdownOutputFileStream{%
9168         local ran_ok, error = pcall(function()

```

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

```

9169             local kpse
9170             (function()
9171                 local should_initialize = package.loaded.kpse == nil
9172                                     or tex.initialize
9173             local ran_ok
9174             ran_ok, kpse = pcall(require, "kpse")
9175             if ran_ok and should_initialize then
9176                 kpse.set_program_name("luatex")
9177             end
9178             end)()
9179             #1
9180         end)

```

If there was an error, use the file `errorTempFileName` to store the error message.

```

9181         if not ran_ok then
9182             local file = io.open("%
9183             |markdownOptionOutputDir
9184             /|markdownOptionErrorTempFileName", "w")
9185             if file then
9186                 file:write(error .. "\n")

```

```

9187         file:close()
9188     end
9189     print('\markdownError{An error was encountered while executing
9190         Lua code}{For further clues, examine the file
9191         "|markdownOptionOutputDir
9192         /|markdownOptionErrorTempFileName"}')
9193     end}%
9194 |immediate|closeout|markdownOutputFileStream

```

Execute the generated `helperScriptFileName` Lua script using the `TeXLua` binary and store the output in the `outputTempFileName` file.

```

9195 |markdownInfo{Executing a helper Lua script from the file
9196     "|markdownOptionHelperScriptFileName" and storing the result in the
9197     file "|markdownOptionOutputTempFileName"}%
9198 |markdownExecute{texlua "|markdownOptionOutputDir
9199     /|markdownOptionHelperScriptFileName" > %
9200     "|markdownOptionOutputDir
9201     /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `outputTempFileName` file.

```

9202 |input|markdownOptionOutputTempFileName|relax}%
9203 |endgroup

```

### 3.2.6 Direct Lua Access

The following `TeX` code is intended for `TeX` engines that provide direct access to Lua (Lua`TeX`). The macro `\markdownLuaExecute` defined here and in Section 3.2.5 are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```

9204 \fi
9205 \ifnum\markdownMode=2\relax
9206     \markdownWarning{Using mode 2: Direct Lua access
9207         (deprecated, to be removed in Markdown 3.0.0)}%

```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.5,

```

9208 \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.

```

9209     \catcode`\|=0%
9210     \catcode`\|=12%
9211     |gdef|markdownLuaExecute#1{%
9212         |directlua{%
9213             local function print(input)

```

```

9214         local output = {}
9215         for line in input:gmatch("[^\\r\\n]+") do
9216             table.insert(output, line)
9217         end
9218         tex.print(output)
9219     end
9220     #1
9221 }%
9222 }%
9223 |endgroup
9224 \\fi

```

### 3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain TeX.

```

9225 \\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.

```

9226     \\catcode`|=0%
9227     \\catcode`\\=12%
9228     \\catcode`|&=6%
9229     |gdef|markdownInput#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

9230         |begingroup
9231         |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.

```

9232         |catcode`|#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

9233     |markdownIfOption{frozenCache}{%
9234         |ifnum|markdownOptionFrozenCacheCounter=0|relax
9235         |markdownInfo{Reading frozen cache from
9236             "|markdownOptionFrozenCacheFileName"}%
9237         |input|markdownOptionFrozenCacheFileName|relax
9238     |fi
9239     |markdownInfo{Including markdown document number

```

```

9240      "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
9241      |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
9242      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
9243    }{%
9244      |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  $\text{\LaTeX}$ Mk to track changes to the markdown document.

```

9245      |openin|markdownInputFileStream&1
9246      |closein|markdownInputFileStream
9247      |markdownPrepareLuaOptions
9248      |markdownLuaExecute{%
9249        |markdownPrepare
9250        local file = assert(io.open("&1", "r"),
9251          [[Could not open file "&1" for reading]])
9252        local input = assert(file:read("*a"))
9253        assert(file:close())

```

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

```

9254      print(convert(input))}%

```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```

9255      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
9256    }%
9257  |endgroup
9258 }%
9259 |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  $\text{\TeX}$  to execute a  $\text{\TeX}$  document in the middle of a markdown document fragment.

```

9260 \gdef\markdownEscape#1{%
9261   \catcode`\%=14\relax
9262   \catcode`\#=6\relax
9263   \input #1\relax
9264   \catcode`\%=12\relax
9265   \catcode`\#=12\relax
9266 }%

```

### 3.3 $\text{\LaTeX}$ Implementation

The  $\text{\LaTeX}$  implementation makes use of the fact that, apart from some subtle differences,  $\text{\LaTeX}$  implements the majority of the plain  $\text{\TeX}$  format [12, Section 9]. As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```

9267 \def\markdownVersionSpace{ }%
9268 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
9269 \markdownVersion\markdownVersionSpace markdown renderer]%

```

Use reflection to define the `renderers` and `rendererPrototypes` keys of `\markdownSetup` as well as the keys that correspond to Lua options.

```

9270 \ExplSyntaxOn
9271 \@@_latex_define_renderers:
9272 \@@_latex_define_renderer_prototypes:
9273 \ExplSyntaxOff

```

### 3.3.1 Logging Facilities

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

### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain  $\text{\TeX}$  implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the  $\text{\LaTeX}$  interface (see Section 2.3.2).

```

9274 \let\markdownInputPlainTeX\markdownInput
9275 \renewcommand\markdownInput[2][]{}%
9276 \begingroup
9277 \markdownSetup{#1}%
9278 \markdownInputPlainTeX{#2}%
9279 \endgroup}%

```

The `markdown`, and `markdown*`  $\text{\LaTeX}$  environments are implemented using the `\markdownReadAndConvert` macro.

```

9280 \renewenvironment{markdown}{%
9281 \markdownReadAndConvert@markdown{}}{%
9282 \markdownEnd}%
9283 \renewenvironment{markdown*}[1]{%
9284 \markdownSetup{#1}%
9285 \markdownReadAndConvert@markdown*}{%
9286 \markdownEnd}%
9287 \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*.

```

9288 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
9289 \catcode`\{=12\catcode`\|=12\catcode`\}=12%

```



```

9290 |gdef|markdownReadAndConvert@markdown#1<%
9291 |markdownReadAndConvert<\end{markdown#1}>%
9292 <|end<markdown#1>>>%
9293 |endgroup

```

**3.3.2.1  $\LaTeX$  Themes** This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
9294 \ExplSyntaxOn
```

To keep track of our current place when packages themes have been nested, we will maintain the `\g_@@_latex_themes_seq` stack of theme names.

```

9295 \newcommand\markdownLaTeXThemeName{}
9296 \seq_new:N \g_@@_latex_themes_seq
9297 \seq_gput_right:NV
9298   \g_@@_latex_themes_seq
9299   \markdownLaTeXThemeName
9300 \newcommand\markdownLaTeXThemeLoad[2]{
9301   \def\@tempa{%
9302     \def\markdownLaTeXThemeName{#2}
9303     \seq_gput_right:NV
9304       \g_@@_latex_themes_seq
9305       \markdownLaTeXThemeName
9306     \RequirePackage{#1}
9307     \seq_pop_right:NN
9308       \g_@@_latex_themes_seq
9309     \l_tmpa_tl
9310     \seq_get_right:NN
9311       \g_@@_latex_themes_seq
9312     \l_tmpa_tl
9313     \exp_args:NNV
9314       \def
9315         \markdownLaTeXThemeName
9316         \l_tmpa_tl}
9317 \ifmarkdownLaTeXLoaded
9318   \@tempa
9319 \else
9320   \exp_args:No
9321     \AtEndOfPackage
9322     { \@tempa }
9323 \fi}
9324 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
9325 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
9326 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
9327 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
9328 \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T<sub>E</sub>X option is disabled and the code block has not been previously typeset:

```
9329 \renewcommand\markdownRendererInputFencedCodePrototype[2]{%
9330 \def\next##1 ##2\relax{%
9331 \ifthenelse{\equal{##1}{dot}}{%
9332 \markdownIfOption{frozenCache}{}}{%
9333 \immediate\write18{%
9334 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
9335 then
9336 dot -Tpdf -o #1.pdf #1;
9337 cp #1 #1.pdf.source;
9338 fi}}%
9339 }
```

We include the typeset image using the image token renderer:

```
9339 \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:

```
9340 }{%
9341 \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
9342 }%
9343 }%
9344 \next#2 \relax}%
```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```
9345 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
9346 \markdownRendererImagePrototype
```

We load the catchfile and grffile packages, see also Section 1.1.3:

```
9347 \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.

```
9348 \newcount\markdown@witiko@graphicx@http@counter
9349 \markdown@witiko@graphicx@http@counter=0
9350 \newcommand\markdown@witiko@graphicx@http@filename{%
9351 \markdownOptionCacheDir/witiko_graphicx_http%
9352 .\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.

```
9353 \newcommand\markdown@witiko@graphicx@http@download[2]{%
9354   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:

```
9355 \begingroup
9356 \catcode`\%=12
9357 \catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
9358 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
9359   \begingroup
9360   \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 T<sub>E</sub>X option is disabled:

```
9361   \markdownIfOption{frozenCache}{}{^^A
9362     \immediate\write18{^^A
9363       mkdir -p "\markdownOptionCacheDir";
9364       if printf '%s' "#3" | grep -q -E '^https?:';
9365       then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
9366       OUTPUT_PREFIX="\markdownOptionCacheDir";
9367       OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
9368       OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/)');
9369       OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
9370       if ! [ -e "$OUTPUT" ];
9371       then
9372         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
9373         printf '%s' "$OUTPUT" > "\filename";
9374       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:

```
9375       else
9376         printf '%s' '#3' > "\filename";
9377       fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

9378 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
9379 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
9380 {#1}{#2}{\filename}{#4}^^A
9381 \endgroup
9382 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
9383 \endgroup

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

9384 \renewcommand\markdownRendererTildePrototype{~}%

```

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

9385 \DeclareOption*{%
9386 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
9387 \ProcessOptions\relax

```

After processing the options, activate the `jeekyllDataRenderes`, `renderers`, `rendererPrototypes`, and `code` keys.

```

9388 \ExplSyntaxOn
9389 \keys_define:nn
9390 { markdown/latex-options }
9391 {
9392   renderers .code:n = {
9393     \keys_set:nn
9394       { markdown/latex-options/renderers }
9395       { #1 }
9396   },
9397 }
9398 \@@_with_various_cases:nn
9399 { rendererPrototypes }
9400 {
9401   \keys_define:nn
9402     { markdown/latex-options }
9403     {
9404       #1 .code:n = {
9405         \keys_set:nn
9406           { markdown/latex-options/renderer-prototypes }
9407           { ##1 }
9408       },
9409     }
9410 }

```

The `code` key is used to immediately expand and execute code, which can be especially useful in  $\text{\LaTeX}$  snippets.

```

9411 \keys_define:nn
9412 { markdown/latex-options }

```

```

9413 {
9414   code .code:n = { #1 },
9415 }

```

The `jeekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jeekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```

9416 \@@_with_various_cases:nn
9417 { jeekyllDataRenderers }
9418 {
9419   \keys_define:nn
9420     { markdown/latex-options }
9421     {
9422       #1 .code:n = {
9423         \tl_set:Nn
9424           \l_tmpa_tl
9425           { ##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.

```

9426       \tl_replace_all:NnV
9427         \l_tmpa_tl
9428         { / }
9429       \c_backslash_str
9430     \keys_set:nV
9431       { markdown/latex-options/jeekyll-data-renderers }
9432       \l_tmpa_tl
9433   },
9434 }
9435 }
9436 \keys_define:nn
9437 { markdown/latex-options/jeekyll-data-renderers }
9438 {
9439   unknown .code:n = {
9440     \tl_set_eq:NN
9441       \l_tmpa_tl
9442       \l_keys_key_str
9443     \tl_replace_all:NVn
9444       \l_tmpa_tl
9445       \c_backslash_str
9446       { / }
9447     \tl_put_right:Nn
9448       \l_tmpa_tl
9449     {

```

```

9450         .code:n = { #1 }
9451     }
9452     \keys_define:nV
9453     { markdown/jekyllData }
9454     \l_tmpa_tl
9455 }
9456 }
9457 \cs_generate_variant:Nn
9458   \keys_define:nn
9459   { nV }
9460 \ExplSyntaxOff

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.2), none of it will take effect.

```

9461 \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.

```

9462 \@ifclassloaded{beamer}{}{%
9463   \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
9464   \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
9465 }

```

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.

```

9466 \ExplSyntaxOn
9467 \@ifpackageloaded{paralist}{
9468   \tl_new:N
9469     \l_@@_latex_fancy_list_item_label_number_style_tl
9470   \tl_new:N
9471     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9472   \cs_new:Nn
9473     \@@_latex_fancy_list_item_label_number:nn
9474     {
9475       \str_case:nn
9476         { #1 }
9477         {
9478           { Decimal } { #2 }
9479           { LowerRoman } { \int_to_roman:n { #2 } }
9480           { UpperRoman } { \int_to_Roman:n { #2 } }
9481           { LowerAlpha } { \int_to_alph:n { #2 } }
9482           { UpperAlpha } { \int_to_alph:n { #2 } }
9483         }
9484     }

```

```

9485 \cs_new:Nn
9486   \@@_latex_fancy_list_item_label_delimiter:n
9487   {
9488     \str_case:nn
9489       { #1 }
9490       {
9491         { Default } { . }
9492         { OneParen } { ) }
9493         { Period } { . }
9494       }
9495   }
9496 \cs_new:Nn
9497   \@@_latex_fancy_list_item_label:nnn
9498   {
9499     \@@_latex_fancy_list_item_label_number:nn
9500       { #1 }
9501       { #3 }
9502     \@@_latex_fancy_list_item_label_delimiter:n
9503       { #2 }
9504   }
9505 \cs_new:Nn
9506   \@@_latex_paralist_style:nn
9507   {
9508     \str_case:nn
9509       { #1 }
9510       {
9511         { Decimal } { 1 }
9512         { LowerRoman } { i }
9513         { UpperRoman } { I }
9514         { LowerAlpha } { a }
9515         { UpperAlpha } { A }
9516       }
9517     \@@_latex_fancy_list_item_label_delimiter:n
9518       { #2 }
9519   }
9520 \markdownSetup{rendererPrototypes={
9521   ulBeginTight = {\begin{compactitem}},
9522   ulEndTight = {\end{compactitem}},
9523   fancyOlBegin = {
9524     \group_begin:
9525     \tl_set:Nn
9526       \l_@@_latex_fancy_list_item_label_number_style_tl
9527       { #1 }
9528     \tl_set:Nn
9529       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9530       { #2 }
9531     \tl_set:Nn

```

```

9532     \l_tmpa_tl
9533     { \begin{enumerate}[ ]
9534 \tl_put_right:Nx
9535     \l_tmpa_tl
9536     { \@@_latex_paralist_style:nn { #1 } { #2 } }
9537 \tl_put_right:Nn
9538     \l_tmpa_tl
9539     { ] }
9540 \l_tmpa_tl
9541 },
9542 fancyOlEnd = {
9543     \end{enumerate}
9544     \group_end:
9545 },
9546 olBeginTight = {\begin{compactenum}},
9547 olEndTight = {\end{compactenum}},
9548 fancyOlBeginTight = {
9549     \group_begin:
9550     \tl_set:Nn
9551         \l_@@_latex_fancy_list_item_label_number_style_tl
9552         { #1 }
9553     \tl_set:Nn
9554         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9555         { #2 }
9556     \tl_set:Nn
9557         \l_tmpa_tl
9558         { \begin{compactenum}[ ]
9559 \tl_put_right:Nx
9560         \l_tmpa_tl
9561         { \@@_latex_paralist_style:nn { #1 } { #2 } }
9562 \tl_put_right:Nn
9563         \l_tmpa_tl
9564         { ] }
9565     \l_tmpa_tl
9566 },
9567 fancyOlEndTight = {
9568     \end{compactenum}
9569     \group_end:
9570 },
9571 fancyOlItemWithNumber = {
9572     \item
9573     [
9574         \@@_latex_fancy_list_item_label:VVn
9575         \l_@@_latex_fancy_list_item_label_number_style_tl
9576         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9577         { #1 }
9578     ]

```



```

9579     },
9580     dlBeginTight = {\begin{compactdesc}},
9581     dlEndTight = {\end{compactdesc}}}}
9582 \cs_generate_variant:Nn
9583   @@_latex_fancy_list_item_label:nnn
9584   { VVn }
9585 }{
9586   \markdownSetup{rendererPrototypes={
9587     ulBeginTight = {\markdownRendererUlBegin},
9588     ulEndTight = {\markdownRendererUlEnd},
9589     fancyOlBegin = {\markdownRendererOlBegin},
9590     fancyOlEnd = {\markdownRendererOlEnd},
9591     olBeginTight = {\markdownRendererOlBegin},
9592     olEndTight = {\markdownRendererOlEnd},
9593     fancyOlBeginTight = {\markdownRendererOlBegin},
9594     fancyOlEndTight = {\markdownRendererOlEnd},
9595     dlBeginTight = {\markdownRendererDlBegin},
9596     dlEndTight = {\markdownRendererDlEnd}}}
9597 }
9598 \ExplSyntaxOff
9599 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

9600 \ifpackageloaded{unicode-math}{
9601   \markdownSetup{rendererPrototypes={
9602     untickedBox = {\mdlgwhtsquare$},
9603   }}
9604 }{
9605   \RequirePackage{amssymb}
9606   \markdownSetup{rendererPrototypes={
9607     untickedBox = {\square$},
9608   }}
9609 }
9610 \RequirePackage{csvsimple}
9611 \RequirePackage{fancyvrb}
9612 \RequirePackage{graphicx}
9613 \markdownSetup{rendererPrototypes={
9614   hardLineBreak = {\},
9615   leftBrace = {\textbraceleft},
9616   rightBrace = {\textbraceright},
9617   dollarSign = {\textdollar},
9618   underscore = {\textunderscore},
9619   circumflex = {\textasciicircum},
9620   backslash = {\textbackslash},
9621   tilde = {\textasciitilde},
9622   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\text{\TeX}$  during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>28</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

9623   codeSpan = {%
9624     \ifmmode
9625       \text{#1}%
9626     \else
9627       \texttt{#1}%
9628     \fi
9629   }}
9630 \ExplSyntaxOn
9631 \markdownSetup{
9632   rendererPrototypes = {
9633     contentBlock = {
9634       \str_case:nnF
9635         { #1 }
9636         {
9637           { csv }
9638           {
9639             \begin{table}
9640               \begin{center}
9641                 \csvautotabular{#3}
9642               \end{center}
9643               \tl_if_empty:nF
9644                 { #4 }
9645                 { \caption{#4} }
9646             \end{table}
9647           }
9648           { tex } { \markdownEscape{#3} }
9649         }
9650       { \markdownInput{#3} }
9651     },
9652   },
9653 }
9654 \ExplSyntaxOff
9655 \markdownSetup{rendererPrototypes={
9656   image = {%
9657     \begin{figure}%
9658     \begin{center}%
9659     \includegraphics{#3}%

```

---

<sup>28</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

9660     \end{center}%
9661     \ifx\empty#4\empty\else
9662         \caption{#4}%
9663     \fi
9664     \end{figure}},
9665     ulBegin = {\begin{itemize}},
9666     ulEnd = {\end{itemize}},
9667     olBegin = {\begin{enumerate}},
9668     olItem = {\item{}},
9669     olItemWithNumber = {\item[#1.]},
9670     olEnd = {\end{enumerate}},
9671     dlBegin = {\begin{description}},
9672     dlItem = {\item[#1]},
9673     dlEnd = {\end{description}},
9674     emphasis = {\emph{#1}},
9675     tickedTextBox = {\$\boxtimes$},
9676     halfTickedTextBox = {\$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

9677     headerAttributeContextBegin = {%
9678         \markdownSetup{
9679             rendererPrototypes = {
9680                 attributeIdentifier = {%
9681                     \begingroup
9682                     \def\next####1{%
9683                         \def####1#####1{%
9684                             \endgroup
9685                             #####1{#####1}%
9686                             \label{##1}%
9687                         }%
9688                     }%
9689                     \next\markdownRendererHeadingOne
9690                     \next\markdownRendererHeadingTwo
9691                     \next\markdownRendererHeadingThree
9692                     \next\markdownRendererHeadingFour
9693                     \next\markdownRendererHeadingFive
9694                     \next\markdownRendererHeadingSix
9695                 },
9696             },
9697         }%
9698     },
9699     headerAttributeContextEnd = {},
9700     superscript = {\textsuperscript{#1}},
9701     subscript = {\textsubscript{#1}},
9702     displayMath = {\begin{displaymath}#1\end{displaymath}},
9703     inlineMath = {\begin{math}#1\end{math}},

```

```

9704 blockQuoteBegin = {\begin{quotation}},
9705 blockQuoteEnd = {\end{quotation}},
9706 inputVerbatim = {\VerbatimInput{#1}},
9707 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
9708 note = {\footnote{#1}}}}

```

**3.3.4.1 Fenced Code** When no infostring has been specified, default to the indented code block renderer.

```

9709 \RequirePackage{ltxcmds}
9710 \ExplSyntaxOn
9711 \cs_gset:Npn
9712   \markdownRendererInputFencedCodePrototype#1#2
9713   {
9714     \tl_if_empty:nTF
9715       { #2 }
9716       { \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.

```

9717   {
9718     \regex_extract_once:nnN
9719       { \w* }
9720       { #2 }
9721     \l_tmpa_seq
9722     \seq_pop_left:NN
9723     \l_tmpa_seq
9724     \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

9725     \ltx@ifpackageloaded
9726       { minted }
9727       {
9728         \catcode`\#=6\relax
9729         \exp_args:NV
9730         \inputminted
9731         \l_tmpa_tl
9732         { #1 }
9733         \catcode`\#=12\relax
9734       }
9735     {

```

When the listings package is loaded, use it for syntax highlighting.

```

9736     \ltx@ifpackageloaded
9737       { listings }
9738       { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

9739             { \markdownRendererInputFencedCode{#1}{ } }
9740         }
9741     }
9742 }
9743 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

9744 \ExplSyntaxOn
9745 \def\markdownLATEXStrongEmphasis#1{%
9746   \str_if_in:NnTF
9747     \f@series
9748     { b }
9749     { \textnormal{#1} }
9750     { \textbf{#1} }
9751 }
9752 \ExplSyntaxOff
9753 \markdownSetup{rendererPrototypes={strongEmphasis={%
9754   \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

9755 \@ifundefined{chapter}{%
9756   \markdownSetup{rendererPrototypes = {
9757     headingOne = {\section{#1}},
9758     headingTwo = {\subsection{#1}},
9759     headingThree = {\subsubsection{#1}},
9760     headingFour = {\paragraph{#1}\leavevmode},
9761     headingFive = {\subparagraph{#1}\leavevmode}}}
9762 }{%
9763   \markdownSetup{rendererPrototypes = {
9764     headingOne = {\chapter{#1}},
9765     headingTwo = {\section{#1}},
9766     headingThree = {\subsection{#1}},
9767     headingFour = {\subsubsection{#1}},
9768     headingFive = {\paragraph{#1}\leavevmode},
9769     headingSix = {\subparagraph{#1}\leavevmode}}}
9770 }%

```

**3.3.4.2 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

9771 \markdownSetup{
9772   rendererPrototypes = {
9773     ulItem = {%
9774       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
9775     },
9776   },
9777 }
9778 \def\markdownLaTeXUListItem{%

```

```

9779 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
9780   \item[\markdownLaTeXCheckbox]%
9781   \expandafter\@gobble
9782 \else
9783   \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
9784     \item[\markdownLaTeXCheckbox]%
9785     \expandafter\expandafter\expandafter\@gobble
9786   \else
9787     \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
9788       \item[\markdownLaTeXCheckbox]%
9789       \expandafter\expandafter\expandafter\expandafter
9790       \expandafter\expandafter\expandafter\@gobble
9791     \else
9792       \item{}%
9793     \fi
9794   \fi
9795 \fi
9796 }

```

**3.3.4.3 HTML elements** If the `html` option is enabled and we are using `TeX4ht`<sup>29</sup>, we will pass HTML elements to the output HTML document unchanged.

```

9797 \@ifundefined{HCode}{}{
9798   \markdownSetup{
9799     rendererPrototypes = {
9800       inlineHtmlTag = {%
9801         \ifvmode
9802           \IgnorePar
9803         \EndP
9804         \fi
9805         \HCode{#1}%
9806       },
9807       inputBlockHtmlElement = {%
9808         \ifvmode
9809           \IgnorePar
9810         \fi
9811         \EndP
9812         \special{t4ht*<#1}%
9813         \par
9814         \ShowPar
9815       },
9816     },
9817   }
9818 }

```

---

<sup>29</sup>See <https://tug.org/tex4ht/>.

**3.3.4.4 Citations** Here is a basic implementation for citations that uses the  $\text{\LaTeX}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `Bib $\text{\LaTeX}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

9819 \newcount\markdownLaTeXCitationsCounter
9820
9821 % Basic implementation
9822 \RequirePackage{gobble}
9823 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
9824   \advance\markdownLaTeXCitationsCounter by 1\relax
9825   \ifx\relax#4\relax
9826     \ifx\relax#5\relax
9827       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9828         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
9829         \expandafter\expandafter\expandafter
9830         \expandafter\expandafter\expandafter\expandafter
9831         \@gobblethree
9832       \fi
9833     \else% Before a postnote (#5), dump the accumulator
9834       \ifx\relax#1\relax\else
9835         \cite{#1}%
9836       \fi
9837     \cite[#5]{#6}%
9838     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9839     \else
9840       \expandafter\expandafter\expandafter
9841       \expandafter\expandafter\expandafter\expandafter
9842       \expandafter\expandafter\expandafter
9843       \expandafter\expandafter\expandafter\expandafter
9844       \markdownLaTeXBasicCitations
9845     \fi
9846     \expandafter\expandafter\expandafter
9847     \expandafter\expandafter\expandafter\expandafter{%
9848     \expandafter\expandafter\expandafter
9849     \expandafter\expandafter\expandafter\expandafter}%
9850     \expandafter\expandafter\expandafter
9851     \expandafter\expandafter\expandafter\expandafter{%
9852     \expandafter\expandafter\expandafter
9853     \expandafter\expandafter\expandafter\expandafter}%
9854     \expandafter\expandafter\expandafter
9855     \@gobblethree
9856   \fi
9857   \else% Before a prenote (#4), dump the accumulator
9858     \ifx\relax#1\relax\else
9859       \cite{#1}%
9860     \fi

```

```

9861 \ifnum\markdownLaTeXCitationsCounter>1\relax
9862 \space % Insert a space before the prenote in later citations
9863 \fi
9864 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
9865 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9866 \else
9867 \expandafter\expandafter\expandafter
9868 \expandafter\expandafter\expandafter\expandafter
9869 \markdownLaTeXBasicCitations
9870 \fi
9871 \expandafter\expandafter\expandafter{%
9872 \expandafter\expandafter\expandafter}%
9873 \expandafter\expandafter\expandafter{%
9874 \expandafter\expandafter\expandafter}%
9875 \expandafter
9876 \@gobblethree
9877 \fi\markdownLaTeXBasicCitations{#1#2#6},}
9878 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
9879
9880 % Natbib implementation
9881 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
9882 \advance\markdownLaTeXCitationsCounter by 1\relax
9883 \ifx\relax#3\relax
9884 \ifx\relax#4\relax
9885 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9886 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
9887 \expandafter\expandafter\expandafter
9888 \expandafter\expandafter\expandafter\expandafter
9889 \@gobbletwo
9890 \fi
9891 \else% Before a postnote (#4), dump the accumulator
9892 \ifx\relax#1\relax\else
9893 \citep{#1}%
9894 \fi
9895 \citep[] [#4]{#5}%
9896 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9897 \else
9898 \expandafter\expandafter\expandafter
9899 \expandafter\expandafter\expandafter\expandafter
9900 \expandafter\expandafter\expandafter
9901 \expandafter\expandafter\expandafter\expandafter
9902 \markdownLaTeXNatbibCitations
9903 \fi
9904 \expandafter\expandafter\expandafter
9905 \expandafter\expandafter\expandafter\expandafter{%
9906 \expandafter\expandafter\expandafter
9907 \expandafter\expandafter\expandafter\expandafter}%

```



```

9908     \expandafter\expandafter\expandafter
9909     \@gobbletwo
9910     \fi
9911 \else% Before a prenote (#3), dump the accumulator
9912     \ifx\relax#1\relax\relax\else
9913         \citep{#1}%
9914     \fi
9915     \citep[#3][#4]{#5}%
9916     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9917     \else
9918         \expandafter\expandafter\expandafter
9919         \expandafter\expandafter\expandafter\expandafter
9920         \markdownLaTeXNatbibCitations
9921     \fi
9922     \expandafter\expandafter\expandafter{%
9923     \expandafter\expandafter\expandafter}%
9924     \expandafter
9925     \@gobbletwo
9926     \fi\markdownLaTeXNatbibCitations{#1,#5}}
9927 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
9928     \advance\markdownLaTeXCitationsCounter by 1\relax
9929     \ifx\relax#3\relax
9930         \ifx\relax#4\relax
9931             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9932                 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
9933                 \expandafter\expandafter\expandafter
9934                 \expandafter\expandafter\expandafter\expandafter
9935                 \@gobbletwo
9936             \fi
9937         \else% After a prenote or a postnote, dump the accumulator
9938             \ifx\relax#1\relax\else
9939                 \citet{#1}%
9940             \fi
9941             , \citet[#3][#4]{#5}%
9942             \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
9943                 ,
9944             \else
9945                 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
9946                     ,
9947                 \fi
9948             \fi
9949             \expandafter\expandafter\expandafter
9950             \expandafter\expandafter\expandafter\expandafter
9951             \markdownLaTeXNatbibTextCitations
9952             \expandafter\expandafter\expandafter
9953             \expandafter\expandafter\expandafter\expandafter{%
9954             \expandafter\expandafter\expandafter

```

```

9955     \expandafter\expandafter\expandafter\expandafter}%
9956     \expandafter\expandafter\expandafter
9957     \@gobbletwo
9958     \fi
9959 \else% After a prenote or a postnote, dump the accumulator
9960     \ifx\relax#1\relax\relax\else
9961         \citet{#1}%
9962     \fi
9963     , \citet[#3][#4]{#5}%
9964     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
9965     ,
9966     \else
9967         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
9968         ,
9969         \fi
9970     \fi
9971     \expandafter\expandafter\expandafter
9972     \markdownLaTeXNatbibTextCitations
9973     \expandafter\expandafter\expandafter{%
9974     \expandafter\expandafter\expandafter}%
9975     \expandafter
9976     \@gobbletwo
9977     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
9978
9979 % BibLaTeX implementation
9980 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
9981     \advance\markdownLaTeXCitationsCounter by 1\relax
9982     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9983         \autocites#1[#3][#4]{#5}%
9984         \expandafter\@gobbletwo
9985     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
9986 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
9987     \advance\markdownLaTeXCitationsCounter by 1\relax
9988     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9989         \textcites#1[#3][#4]{#5}%
9990         \expandafter\@gobbletwo
9991     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
9992
9993 \markdownSetup{rendererPrototypes = {
9994     cite = {%
9995         \markdownLaTeXCitationsCounter=1%
9996         \def\markdownLaTeXCitationsTotal{#1}%
9997         \@ifundefined{autocites}{%
9998             \@ifundefined{citep}{%
9999                 \expandafter\expandafter\expandafter
10000                 \markdownLaTeXBasicCitations
10001                 \expandafter\expandafter\expandafter{%

```

```

10002         \expandafter\expandafter\expandafter}%
10003         \expandafter\expandafter\expandafter{%
10004         \expandafter\expandafter\expandafter}%
10005     }{%
10006         \expandafter\expandafter\expandafter
10007         \markdownLaTeXNatbibCitations
10008         \expandafter\expandafter\expandafter{%
10009         \expandafter\expandafter\expandafter}%
10010     }%
10011 }{%
10012     \expandafter\expandafter\expandafter
10013     \markdownLaTeXBibLaTeXCitations
10014     \expandafter{\expandafter}%
10015 }},
10016 textCite = {%
10017     \markdownLaTeXCitationsCounter=1%
10018     \def\markdownLaTeXCitationsTotal{#1}%
10019     \@ifundefined{autocites}{%
10020         \@ifundefined{citep}{%
10021             \expandafter\expandafter\expandafter
10022             \markdownLaTeXBasicTextCitations
10023             \expandafter\expandafter\expandafter{%
10024             \expandafter\expandafter\expandafter}%
10025             \expandafter\expandafter\expandafter{%
10026             \expandafter\expandafter\expandafter}%
10027         }{%
10028             \expandafter\expandafter\expandafter
10029             \markdownLaTeXNatbibTextCitations
10030             \expandafter\expandafter\expandafter{%
10031             \expandafter\expandafter\expandafter}%
10032         }%
10033     }{%
10034         \expandafter\expandafter\expandafter
10035         \markdownLaTeXBibLaTeXTextCitations
10036         \expandafter{\expandafter}%
10037     }}}

```

### 3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```

10038 \RequirePackage{url}
10039 \RequirePackage{expl3}
10040 \ExplSyntaxOn
10041 \def\markdownRendererLinkPrototype#1#2#3#4{
10042     \tl_set:Nn \l_tmpa_tl { #1 }
10043     \tl_set:Nn \l_tmpb_tl { #2 }
10044     \bool_set:Nn
10045         \l_tmpa_bool

```

```

10046     {
10047         \tl_if_eq_p:NN
10048             \l_tmpa_tl
10049             \l_tmpb_tl
10050     }
10051     \tl_set:Nn \l_tmpa_tl { #4 }
10052     \bool_set:Nn
10053         \l_tmpb_bool
10054     {
10055         \tl_if_empty_p:N
10056             \l_tmpa_tl
10057     }

```

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.

```

10058     \bool_if:nTF
10059     {
10060         \l_tmpa_bool && \l_tmpb_bool
10061     }
10062     {
10063         \markdownLaTeXRendererAutolink { #2 } { #3 }
10064     }{
10065         \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
10066     }
10067 }
10068 \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.

```

10069     \tl_set:Nn
10070         \l_tmpa_tl
10071         { #2 }
10072     \tl_trim_spaces:N
10073         \l_tmpa_tl
10074     \tl_set:Nx
10075         \l_tmpb_tl
10076     {
10077         \tl_range:Nnn
10078             \l_tmpa_tl
10079             { 1 }
10080             { 1 }
10081     }
10082     \str_if_eq:NNTF
10083         \l_tmpb_tl
10084         \c_hash_str
10085     {
10086         \tl_set:Nx

```

```

10087         \l_tmpb_tl
10088     {
10089         \tl_range:Nnn
10090         \l_tmpa_tl
10091         { 2 }
10092         { -1 }
10093     }
10094     \exp_args:NV
10095     \ref
10096     \l_tmpb_tl
10097 }{
10098     \url { #2 }
10099 }
10100 }
10101 \ExplSyntaxOff
10102 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
10103     #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.

```

10104 \newcount\markdownLaTeXRowCount
10105 \newcount\markdownLaTeXRowTotal
10106 \newcount\markdownLaTeXColumnCounter
10107 \newcount\markdownLaTeXColumnTotal
10108 \newtoks\markdownLaTeXTable
10109 \newtoks\markdownLaTeXTableAlignment
10110 \newtoks\markdownLaTeXTableEnd
10111 \AtBeginDocument{%
10112     \@ifpackageloaded{booktabs}{%
10113         \def\markdownLaTeXTopRule{\toprule}%
10114         \def\markdownLaTeXMidRule{\midrule}%
10115         \def\markdownLaTeXBottomRule{\bottomrule}%
10116     }{%
10117         \def\markdownLaTeXTopRule{\hline}%
10118         \def\markdownLaTeXMidRule{\hline}%
10119         \def\markdownLaTeXBottomRule{\hline}%
10120     }%
10121 }
10122 \markdownSetup{rendererPrototypes={
10123     table = {%
10124         \markdownLaTeXTable={}%
10125         \markdownLaTeXTableAlignment={}%
10126         \markdownLaTeXTableEnd={%
10127             \markdownLaTeXBottomRule
10128             \end{tabular}}%
10129         \ifx\empty#1\empty\else

```

```

10130     \addto@hook\markdownLaTeXTable{%
10131         \begin{table}
10132         \centering}%
10133     \addto@hook\markdownLaTeXTableEnd{%
10134         \caption{#1}
10135         \end{table}}%
10136     \fi
10137     \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
10138     \markdownLaTeXRowCount=0%
10139     \markdownLaTeXRowTotal=#2%
10140     \markdownLaTeXColumnTotal=#3%
10141     \markdownLaTeXRenderTableRow
10142 }
10143 }}
10144 \def\markdownLaTeXRenderTableRow#1{%
10145     \markdownLaTeXColumnCounter=0%
10146     \ifnum\markdownLaTeXRowCount=0\relax
10147         \markdownLaTeXReadAlignments#1%
10148         \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
10149             \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
10150                 \the\markdownLaTeXTableAlignment}}}%
10151         \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
10152     \else
10153         \markdownLaTeXRenderTableCell#1%
10154     \fi
10155     \ifnum\markdownLaTeXRowCount=1\relax
10156         \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
10157     \fi
10158     \advance\markdownLaTeXRowCount by 1\relax
10159     \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
10160         \the\markdownLaTeXTable
10161         \the\markdownLaTeXTableEnd
10162         \expandafter\@gobble
10163     \fi\markdownLaTeXRenderTableRow}
10164 \def\markdownLaTeXReadAlignments#1{%
10165     \advance\markdownLaTeXColumnCounter by 1\relax
10166     \if#1d%
10167         \addto@hook\markdownLaTeXTableAlignment{1}%
10168     \else
10169         \addto@hook\markdownLaTeXTableAlignment{#1}%
10170     \fi
10171     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
10172         \expandafter\@gobble
10173     \fi\markdownLaTeXReadAlignments}
10174 \def\markdownLaTeXRenderTableCell#1{%
10175     \advance\markdownLaTeXColumnCounter by 1\relax
10176     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax

```

```

10177 \addto@hook\markdownLaTeXTable{#1&}%
10178 \else
10179 \addto@hook\markdownLaTeXTable{#1\\}%
10180 \expandafter\@gobble
10181 \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.

```

10182
10183 \markdownIfOption{lineBlocks}{%
10184 \RequirePackage{verse}
10185 \markdownSetup{rendererPrototypes={
10186   lineBlockBegin = {%
10187     \begingroup
10188     \def\markdownRendererHardLineBreak{\\}%
10189     \begin{verse}%
10190   },
10191   lineBlockEnd = {%
10192     \end{verse}%
10193   \endgroup
10194 },
10195 }}
10196 }{}
10197

```

**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.

```

10198 \ExplSyntaxOn
10199 \keys_define:nn
10200 { markdown/jekyllData }
10201 {
10202   author .code:n = { \author{#1} },
10203   date .code:n = { \date{#1} },
10204   title .code:n = { \title{#1} },
10205 }

```

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.

```

10206 % TODO: Remove the command definition in TeX Live 2021.
10207 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
10208 \markdownSetup{
10209   rendererPrototypes = {

```

```

10210     jekyllDataEnd = {
10211 %       TODO: Remove the else branch in TeX Live 2021.
10212     \IfFormatAtLeastTF
10213       { 2020-10-01 }
10214       { \AddToHook{begindocument/end}{\maketitle} }
10215       {
10216         \ifx\@onlypreamble\@notprerr
10217           % We are in the document
10218           \maketitle
10219         \else
10220           % We are in the preamble
10221           \RequirePackage{etoolbox}
10222           \AfterEndPreamble{\maketitle}
10223         \fi
10224       }
10225     },
10226   },
10227 }
10228 \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.

```

10229 \markdownIfOption{strikeThrough}{%
10230   \RequirePackage{soulutf8}%
10231   \markdownSetup{
10232     rendererPrototypes = {
10233       strikeThrough = {%
10234         \st{#1}%
10235       },
10236     }
10237   }
10238 }{}

```

**3.3.4.10 Strike-Through** If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

10239 \markdownIfOption{strikeThrough}{%
10240   \RequirePackage{soulutf8}%
10241   \markdownSetup{
10242     rendererPrototypes = {
10243       strikeThrough = {%
10244         \st{#1}%
10245       },
10246     }
10247   }
10248 }{}

```



**3.3.4.11 Image Attributes** If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values.

```

10249 \ExplSyntaxOn
10250 \@@_if_option:nTF
10251 { linkAttributes }
10252 {
10253   \RequirePackage{graphicx}
10254   \markdownSetup{
10255     rendererPrototypes = {
10256       imageAttributeContextBegin = {
10257         \group_begin:
10258         \markdownSetup{
10259           rendererPrototypes = {
10260             attributeKeyValue = {
10261               \setkeys
10262               { Gin }
10263               { { ##1 } = { ##2 } }
10264             },
10265           },
10266         }
10267       },
10268       imageAttributeContextEnd = {
10269         \group_end:
10270       },
10271     },
10272   }
10273 }
10274 { }
10275 \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`.

```

10276 \ExplSyntaxOn
10277 \cs_gset:Npn
10278   \markdownRendererInputRawInlinePrototype#1#2
10279 {
10280   \str_case:nnF
10281     { #2 }
10282     {
10283       { latex }
10284       {
10285         \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10286         { #1 }
10287         { tex }

```

```

10288     }
10289   }
10290   {
10291     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10292     { #1 }
10293     { #2 }
10294   }
10295 }
10296 \cs_gset:Npn
10297   \markdownRendererInputRawBlockPrototype#1#2
10298   {
10299     \str_case:nnF
10300     { #2 }
10301     {
10302       { latex }
10303       {
10304         \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
10305         { #1 }
10306         { tex }
10307       }
10308     }
10309     {
10310       \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
10311       { #1 }
10312       { #2 }
10313     }
10314   }
10315 \ExplSyntaxOff
10316 \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.

```

10317 \newcommand\markdownMakeOther{%
10318   \count0=128\relax
10319   \loop
10320     \catcode\count0=11\relax
10321     \advance\count0 by 1\relax
10322   \ifnum\count0<256\repeat}%

```

### 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L<sup>A</sup>T<sub>E</sub>X package.

```
10323 \def\markdownMakeOther{%
10324   \count0=128\relax
10325   \loop
10326     \catcode\count0=11\relax
10327     \advance\count0 by 1\relax
10328   \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConT<sub>E</sub>Xt.

```
10329   \catcode`=12}%
```

#### 3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```
10330 \long\def\inputmarkdown{%
10331   \dosingleempty
10332   \doinputmarkdown}%
10333 \long\def\doinputmarkdown[#1]#2{%
10334   \begingroup
10335     \iffirstargument
10336       \setupmarkdown{#1}%
10337     \fi
10338     \markdownInput{#2}%
10339   \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's T<sub>E</sub>X, trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)T<sub>E</sub>X, but ConT<sub>E</sub>Xt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConT<sub>E</sub>Xt MkIV and therefore to insert hard line breaks into markdown text.

```

10340 \ifx\startluacode\undefined % MkII
10341   \begingroup
10342     \catcode`\|=0%
10343     \catcode`\|=12%
10344     |gdef|startmarkdown{%
10345       |markdownReadAndConvert{\stopmarkdown}%
10346                                   {|stopmarkdown}}%
10347     |gdef|stopmarkdown{%
10348       |markdownEnd}%
10349   |endgroup
10350 \else % MkIV
10351   \startluacode
10352     document.markdown_buffering = false
10353     local function preserve_trailing_spaces(line)
10354       if document.markdown_buffering then
10355         line = line:gsub("[ \t][ \t]$", "\t\t")
10356       end
10357       return line
10358     end
10359     resolvers.installinputlinehandler(preserve_trailing_spaces)
10360   \stopluacode
10361   \begingroup
10362     \catcode`\|=0%
10363     \catcode`\|=12%
10364     |gdef|startmarkdown{%
10365       |ctxlua{document.markdown_buffering = true}%
10366       |markdownReadAndConvert{\stopmarkdown}%
10367                                   {|stopmarkdown}}%
10368     |gdef|stopmarkdown{%
10369       |ctxlua{document.markdown_buffering = false}%
10370       |markdownEnd}%
10371   |endgroup
10372 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

10373 \def\markdownRendererHardLineBreakPrototype{\blank}%
10374 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
10375 \def\markdownRendererRightBracePrototype{\textbraceright}%
10376 \def\markdownRendererDollarSignPrototype{\textdollar}%
10377 \def\markdownRendererPercentSignPrototype{\percent}%
10378 \def\markdownRendererUnderscorePrototype{\textunderscore}%
10379 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
10380 \def\markdownRendererBackslashPrototype{\textbackslash}%
10381 \def\markdownRendererTildePrototype{\textasciitilde}%
10382 \def\markdownRendererPipePrototype{\char`|}%

```

```

10383 \def\markdownRendererLinkPrototype#1#2#3#4{%
10384   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
10385   \fi\tt<\hyphenatedurl{#3}>}}%
10386 \usemodule[database]
10387 \defineseparatedlist
10388   [MarkdownConTeXtCSV]
10389   [separator={,},
10390   before=\bTABLE,after=\eTABLE,
10391   first=\bTR,last=\eTR,
10392   left=\bTD,right=\eTD]
10393 \def\markdownConTeXtCSV{csv}
10394 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
10395   \def\markdownConTeXtCSV@arg{#1}%
10396   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
10397     \placetable[] [tab:#1]{#4}{%
10398       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
10399   \else
10400     \markdownInput{#3}%
10401   \fi}%
10402 \def\markdownRendererImagePrototype#1#2#3#4{%
10403   \placefigure[] []{#4}{\externalfigure[#3]}}%
10404 \def\markdownRendererU1BeginPrototype{\startitemize}%
10405 \def\markdownRendererU1BeginTightPrototype{\startitemize[packed]}%
10406 \def\markdownRendererU1ItemPrototype{\item}%
10407 \def\markdownRendererU1EndPrototype{\stopitemize}%
10408 \def\markdownRendererU1EndTightPrototype{\stopitemize}%
10409 \def\markdownRendererO1BeginPrototype{\startitemize[n]}%
10410 \def\markdownRendererO1BeginTightPrototype{\startitemize[packed,n]}%
10411 \def\markdownRendererO1ItemPrototype{\item}%
10412 \def\markdownRendererO1ItemWithNumberPrototype#1{\sym{#1.}}%
10413 \def\markdownRendererO1EndPrototype{\stopitemize}%
10414 \def\markdownRendererO1EndTightPrototype{\stopitemize}%
10415 \definedescription
10416   [MarkdownConTeXtDlItemPrototype]
10417   [location=hanging,
10418   margin=standard,
10419   headstyle=bold]%
10420 \definestartstop
10421   [MarkdownConTeXtDlPrototype]
10422   [before=\blank,
10423   after=\blank]%
10424 \definestartstop
10425   [MarkdownConTeXtDlTightPrototype]
10426   [before=\blank\startpacked,
10427   after=\stoppacked\blank]%
10428 \def\markdownRendererDlBeginPrototype{%
10429   \startMarkdownConTeXtDlPrototype}%

```

```

10430 \def\markdownRendererDlBeginTightPrototype{%
10431   \startMarkdownConTeXtDlTightPrototype}%
10432 \def\markdownRendererDlItemPrototype#1{%
10433   \startMarkdownConTeXtDlItemPrototype{#1}}%
10434 \def\markdownRendererDlItemEndPrototype{%
10435   \stopMarkdownConTeXtDlItemPrototype}%
10436 \def\markdownRendererDlEndPrototype{%
10437   \stopMarkdownConTeXtDlPrototype}%
10438 \def\markdownRendererDlEndTightPrototype{%
10439   \stopMarkdownConTeXtDlTightPrototype}%
10440 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
10441 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
10442 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
10443 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
10444 \def\markdownRendererLineBlockBeginPrototype{%
10445   \begingroup
10446     \def\markdownRendererHardLineBreak{
10447       }%
10448     \startlines
10449   }%
10450 \def\markdownRendererLineBlockEndPrototype{%
10451   \stoplines
10452   \endgroup
10453 }%
10454 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

**3.4.2.1 Fenced Code** When no infostring has been specified, default to the indented code block renderer.

```

10455 \ExplSyntaxOn
10456 \cs_gset:Npn
10457   \markdownRendererInputFencedCodePrototype#1#2
10458   {
10459     \tl_if_empty:nTF
10460       { #2 }
10461       { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptying [latex] [option=TEX]

\starttext
  \startmarkdown

```

```

~~~ latex
\documentclass{article}
\begin{document}
 Hello world!
\end{document}
~~~
\stopmarkdown
\stoptext

```

```

10462     {
10463         \regex_extract_once:nnN
10464         { \w* }
10465         { #2 }
10466         \l_tmpa_seq
10467         \seq_pop_left:NN
10468         \l_tmpa_seq
10469         \l_tmpa_tl
10470         \typefile[\l_tmpa_tl][\{#1}
10471     }
10472 }
10473 \ExplSyntaxOff
10474 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
10475 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
10476 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
10477 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
10478 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
10479 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
10480 \def\markdownRendererThematicBreakPrototype{%
10481     \blackrule[height=1pt, width=\hsize]}%
10482 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
10483 \def\markdownRendererTickedBoxPrototype{$\boxtimes$}
10484 \def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
10485 \def\markdownRendererUntickedBoxPrototype{$\square$}
10486 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
10487 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
10488 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
10489 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%
10490 \def\markdownRendererInlineMathPrototype#1{\$#1\$}%

```

### 3.4.2.2 Tables

There is a basic implementation of tables.

```

10491 \newcount\markdownConTeXtRowCounter
10492 \newcount\markdownConTeXtRowTotal
10493 \newcount\markdownConTeXtColumnCounter
10494 \newcount\markdownConTeXtColumnTotal
10495 \newtoks\markdownConTeXtTable

```

```

10496 \newtoks\markdownConTeXtTableFloat
10497 \def\markdownRendererTablePrototype#1#2#3{%
10498   \markdownConTeXtTable={}%
10499   \ifx\empty#1\empty
10500     \markdownConTeXtTableFloat={%
10501       \the\markdownConTeXtTable}%
10502   \else
10503     \markdownConTeXtTableFloat={%
10504       \placetable{#1}{\the\markdownConTeXtTable}}%
10505   \fi
10506   \begingroup
10507   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
10508   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
10509   \setupTABLE[r][1][topframe=on, bottomframe=on]
10510   \setupTABLE[r][#1][bottomframe=on]
10511   \markdownConTeXtRowCounter=0%
10512   \markdownConTeXtRowTotal=#2%
10513   \markdownConTeXtColumnTotal=#3%
10514   \markdownConTeXtRenderTableRow}
10515 \def\markdownConTeXtRenderTableRow#1{%
10516   \markdownConTeXtColumnCounter=0%
10517   \ifnum\markdownConTeXtRowCounter=0\relax
10518     \markdownConTeXtReadAlignments#1%
10519     \markdownConTeXtTable={\bTABLE}%
10520   \else
10521     \markdownConTeXtTable=\expandafter{%
10522       \the\markdownConTeXtTable\bTR}%
10523     \markdownConTeXtRenderTableCell#1%
10524     \markdownConTeXtTable=\expandafter{%
10525       \the\markdownConTeXtTable\eTR}%
10526   \fi
10527   \advance\markdownConTeXtRowCounter by 1\relax
10528   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
10529     \markdownConTeXtTable=\expandafter{%
10530       \the\markdownConTeXtTable\eTABLE}%
10531     \the\markdownConTeXtTableFloat
10532   \endgroup
10533   \expandafter\gobbleoneargument
10534   \fi\markdownConTeXtRenderTableRow}
10535 \def\markdownConTeXtReadAlignments#1{%
10536   \advance\markdownConTeXtColumnCounter by 1\relax
10537   \if#1d%
10538     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
10539   \fi\if#1l%
10540     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
10541   \fi\if#1c%
10542     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]

```



```

10543 \fi\if#1r%
10544 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
10545 \fi
10546 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
10547 \expandafter\gobbleoneargument
10548 \fi\markdownConTeXtReadAlignments}
10549 \def\markdownConTeXtRenderTableCell#1{%
10550 \advance\markdownConTeXtColumnCounter by 1\relax
10551 \markdownConTeXtTable=\expandafter{%
10552 \the\markdownConTeXtTable\bTD#1\eTD}%
10553 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
10554 \expandafter\gobbleoneargument
10555 \fi\markdownConTeXtRenderTableCell}

```

**3.4.2.3 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`.

```

10556 \ExplSyntaxOn
10557 \cs_gset:Npn
10558 \markdownRendererInputRawInlinePrototype#1#2
10559 {
10560 \str_case:nnF
10561 { #2 }
10562 {
10563 { latex }
10564 {
10565 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10566 { #1 }
10567 { context }
10568 }
10569 }
10570 {
10571 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10572 { #1 }
10573 { #2 }
10574 }
10575 }
10576 \cs_gset:Npn
10577 \markdownRendererInputRawBlockPrototype#1#2
10578 {
10579 \str_case:nnF
10580 { #2 }
10581 {
10582 { context }
10583 {
10584 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn

```

```

10585         { #1 }
10586         { tex }
10587     }
10588 }
10589 {
10590     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
10591     { #1 }
10592     { #2 }
10593 }
10594 }
10595 \cs_gset_eq:NN
10596     \markdownRendererInputRawBlockPrototype
10597     \markdownRendererInputRawInlinePrototype
10598 \ExplSyntaxOff
10599 \stopmodule\protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [8] Till Tantau, Joseph Wright, and Vedran 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 Novotný. *L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> no longer keys packages by pathnames*. Feb. 20, 2021. URL: <https://github.com/latex3/latex2e/issues/510> (visited on 02/21/2021).

- [10] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [11] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [12] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [13] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [14] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

## Index

blankBeforeBlockquote	17
blankBeforeCodeFence	18
blankBeforeDivFence	18
blankBeforeHeading	18
bracketedSpans	19, 55
breakableBlockquotes	19
cacheDir	4, 16, 22, 49, 114, 215, 270, 283
citationNbsps	19
citations	20, 81, 88
codeSpans	20
contentBlocks	16, 21
contentBlocksLanguageMap	16
debugExtensions	9, 17, 21, 212
debugExtensionsFileName	17, 21
defaultOptions	10, 45, 255
definitionLists	22, 61
eagerCache	22, 22
\endmarkdown	98
entities.char_entity	170
entities.dec_entity	170
entities.hex_entity	170
errorTempFileName	51, 276
expandtabs	196

expectJekyllData	23
extensions	24, 121, 216
extensions.bracketed_spans	216
extensions.citations	217
extensions.content_blocks	220
extensions.definition_lists	223
extensions.fancy_lists	225
extensions.fenced_code	229
extensions.fenced_divs	233
extensions.header_attributes	236
extensions.inline_code_attributes	237
extensions.jekyll_data	252
extensions.line_blocks	237
extensions.link_attributes	238
extensions.notes	241
extensions.pipe_table	243
extensions.raw_inline	246
extensions.strike_through	247
extensions.subscripts	247
extensions.superscripts	248
extensions.tex_math	249
fancyLists	25, 76–81, 286
fencedCode	26, 34, 58, 65, 82, 281
fencedCodeAttributes	26
fencedDiv	65
fencedDivs	27, 36
finalizeCache	17, 22, 27, 28, 49, 114, 216
frozenCache	17, 27, 49, 105, 106, 114, 282, 283
frozenCacheCounter	28, 216, 278, 279
frozenCacheFileName	17, 27, 49, 216
hardLineBreaks	28
hashEnumerators	29
headerAttributes	29, 36, 54, 66
helperScriptFileName	50, 51, 276, 277
html	29, 69, 70, 294
hybrid	30, 35, 39, 42, 51, 83, 106, 114, 174, 197, 278
inlineCodeAttributes	30, 59
inlineNotes	31
\inputmarkdown	118, 118, 119, 307
inputTempFileName	272, 273

iterlines	196
jeekyllData	4, 23, 31, 91–94
languages_json	221, 221
lineBlocks	32, 72
linkAttributes	32, 71, 74, 305
\markdown	98
markdown	98, 98, 99, 280
markdown*	98, 98, 99, 280
\markdown_jeekyll_data_concatenate_address:NN	266
\markdown_jeekyll_data_pop:	267
\markdown_jeekyll_data_push:nN	267
\markdown_jeekyll_data_push_address_segment:n	265
\markdown_jeekyll_data_set_keyval:Nn	267
\markdown_jeekyll_data_set_keyvals:nn	267
\markdown_jeekyll_data_update_address_tls:	266
\markdownBegin	47, 47, 48, 96, 98, 118
\markdownEnd	47, 47, 48, 96, 98, 118
\markdownError	96, 96
\markdownEscape	47, 48, 279
\markdownExecute	275
\markdownExecuteDirect	275, 275
\markdownExecuteShellEscape	275, 275
\markdownIfOption	271
\markdownIfSnippetExists	100
\markdownInfo	96
\markdownInput	47, 48, 98, 99, 118, 119, 278, 280
\markdownInputFileStream	271
\markdownInputPlainTeX	280
\markdownLuaExecute	274, 276, 277, 278
\markdownLuaOptions	269, 270
\markdownLuaRegisterIBCallback	97
\markdownLuaUnregisterIBCallback	97
\markdownMakeOther	96, 306, 307
\markdownMode	5, 50, 97, 97, 274, 277
\markdownOptionErrorTempFileName	50
\markdownOptionFinalizeCache	49
\markdownOptionFrozenCache	49
\markdownOptionHelperScriptFileName	49
\markdownOptionHybrid	51
\markdownOptionInputTempFileName	50

<code>\markdownOptionOutputDir</code>	51
<code>\markdownOptionOutputTempFileName</code>	50
<code>\markdownOptionStripPercentSigns</code>	53
<code>\markdownOutputFileStream</code>	271
<code>\markdownPrepare</code>	270
<code>\markdownPrepareLuaOptions</code>	268
<code>\markdownReadAndConvert</code>	96, 272, 280, 307
<code>\markdownReadAndConvertProcessLine</code>	273, 274
<code>\markdownReadAndConvertStripPercentSigns</code>	272
<code>\markdownReadAndConvertTab</code>	272
<code>\markdownRendererAttributeClassName</code>	54
<code>\markdownRendererAttributeIdentifier</code>	54
<code>\markdownRendererAttributeKeyValue</code>	54
<code>\markdownRendererBlockHtmlCommentBegin</code>	69
<code>\markdownRendererBlockHtmlCommentEnd</code>	69
<code>\markdownRendererBlockQuoteBegin</code>	55
<code>\markdownRendererBlockQuoteEnd</code>	55
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	56
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	56
<code>\markdownRendererCite</code>	81, 88
<code>\markdownRendererCodeSpan</code>	59
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	59
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	59
<code>\markdownRendererContentBlock</code>	60, 60
<code>\markdownRendererContentBlockCode</code>	60
<code>\markdownRendererContentBlockOnlineImage</code>	60
<code>\markdownRendererDisplayMath</code>	88
<code>\markdownRendererDlBegin</code>	61
<code>\markdownRendererDlBeginTight</code>	62
<code>\markdownRendererDlDefinitionBegin</code>	62
<code>\markdownRendererDlDefinitionEnd</code>	63
<code>\markdownRendererDlEnd</code>	63
<code>\markdownRendererDlEndTight</code>	63
<code>\markdownRendererDlItem</code>	62
<code>\markdownRendererDlItemEnd</code>	62
<code>\markdownRendererDocumentBegin</code>	74
<code>\markdownRendererDocumentEnd</code>	74
<code>\markdownRendererEllipsis</code>	36, 64
<code>\markdownRendererEmphasis</code>	64, 116
<code>\markdownRendererFancyOlBegin</code>	77, 77
<code>\markdownRendererFancyOlBeginTight</code>	77
<code>\markdownRendererFancyOlEnd</code>	80

<code>\markdownRendererFancyOlEndTight</code>	81
<code>\markdownRendererFancyOlItem</code>	79
<code>\markdownRendererFancyOlItemEnd</code>	79
<code>\markdownRendererFancyOlItemWithNumber</code>	79
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	65
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	65
<code>\markdownRendererFencedDivAttributeContextBegin</code>	66
<code>\markdownRendererFencedDivAttributeContextEnd</code>	66
<code>\markdownRendererFootnote</code>	75, 95
<code>\markdownRendererFootnotePrototype</code>	75, 95
<code>\markdownRendererHalfTickedBox</code>	90
<code>\markdownRendererHardLineBreak</code>	72
<code>\markdownRendererHeaderAttributeContextBegin</code>	66
<code>\markdownRendererHeaderAttributeContextEnd</code>	66
<code>\markdownRendererHeadingFive</code>	68
<code>\markdownRendererHeadingFour</code>	68
<code>\markdownRendererHeadingOne</code>	67
<code>\markdownRendererHeadingSix</code>	68
<code>\markdownRendererHeadingThree</code>	67
<code>\markdownRendererHeadingTwo</code>	67
<code>\markdownRendererHorizontalRule</code>	89, 95
<code>\markdownRendererHorizontalRulePrototype</code>	89, 95
<code>\markdownRendererImage</code>	70
<code>\markdownRendererImageAttributeContextBegin</code>	71
<code>\markdownRendererImageAttributeContextEnd</code>	71
<code>\markdownRendererInlineHtmlComment</code>	69
<code>\markdownRendererInlineHtmlTag</code>	70
<code>\markdownRendererInlineMath</code>	88
<code>\markdownRendererInputBlockHtmlElement</code>	70
<code>\markdownRendererInputFencedCode</code>	58
<code>\markdownRendererInputRawBlock</code>	82
<code>\markdownRendererInputRawInline</code>	82
<code>\markdownRendererInputVerbatim</code>	58
<code>\markdownRendererInterblockSeparator</code>	71
<code>\markdownRendererJekyllDataBegin</code>	91
<code>\markdownRendererJekyllDataBoolean</code>	93
<code>\markdownRendererJekyllDataEmpty</code>	94
<code>\markdownRendererJekyllDataEnd</code>	91
<code>\markdownRendererJekyllDataMappingBegin</code>	91
<code>\markdownRendererJekyllDataMappingEnd</code>	92
<code>\markdownRendererJekyllDataNumber</code>	93
<code>\markdownRendererJekyllDataSequenceBegin</code>	92

\markdownRendererJekyllDataSequenceEnd	92
\markdownRendererJekyllDataString	93
\markdownRendererLineBlockBegin	72
\markdownRendererLineBlockEnd	72
\markdownRendererLineBreak	72
\markdownRendererLineBreakPrototype	72
\markdownRendererLink	73, 116
\markdownRendererLinkAttributeContextBegin	74
\markdownRendererLinkAttributeContextEnd	74
\markdownRendererNbsp	75
\markdownRendererNote	75
\markdownRendererOlBegin	76
\markdownRendererOlBeginTight	77
\markdownRendererOlEnd	80
\markdownRendererOlEndTight	80
\markdownRendererOlItem	37, 78
\markdownRendererOlItemEnd	78
\markdownRendererOlItemWithNumber	36, 78
\markdownRendererReplacementCharacter	83
\markdownRendererSectionBegin	82
\markdownRendererSectionEnd	82
\markdownRendererStrikeThrough	86
\markdownRendererStrongEmphasis	64
\markdownRendererSubscript	86
\markdownRendererSuperscript	87
\markdownRendererTable	87
\markdownRendererTextCite	88
\markdownRendererThematicBreak	89
\markdownRendererTickedBox	90
\markdownRendererUlBegin	56
\markdownRendererUlBeginTight	56
\markdownRendererUlEnd	57
\markdownRendererUlEndTight	58
\markdownRendererUlItem	57
\markdownRendererUlItemEnd	57
\markdownRendererUntickedBox	90
\markdownSetup	99, 99, 280, 284
\markdownSetupSnippet	100, 100
\markdownWarning	96
new	8, 255
notes	33, 75



outputTempFileName	50, 277
parsers	184, 195
parsers.commented_line	186
pipeTables	7, 33, 38, 87
preserveTabs	34, 37, 196
rawAttribute	34, 35, 82
reader	8, 121, 184, 195, 216
reader->add_special_character	8, 10, 211
reader->auto_link_email	203
reader->auto_link_url	203
reader->create_parser	196
reader->finalize_grammar	207
reader->initialize_named_group	211
reader->insert_pattern	8, 9, 208, 213, 239
reader->lookup_reference	199
reader->normalize_tag	195
reader->options	195
reader->parser_functions	196
reader->parser_functions.name	196
reader->parser_functions.parse_inlines_no_link	239
reader->parsers	195, 195
reader->register_link	199
reader->update_rule	208, 210, 213, 239
reader->writer	195
reader.new	195, 195
relativeReferences	35
\setupmarkdown	119, 119
shiftHeadings	7, 35
slice	7, 36, 171, 180, 181
smartEllipses	36, 64, 114
\startmarkdown	118, 118, 307
startNumber	36, 78, 79
\stopmarkdown	118, 118, 307
strikeThrough	37, 86, 304
stripIndent	37, 196
stripPercentSigns	272
subscripts	38, 86
superscripts	38, 87
syntax	209, 213

tableCaptions	7, 38
taskLists	39, 90, 293
texComments	39, 197
texMathDollars	40, 88
texMathDoubleBackslash	40, 88
texMathSingleBackslash	41, 88
tightLists	41, 56, 58, 62, 63, 77, 80, 81, 286
underscores	42
util.cache	122, 123
util.cache_verbatim	123
util.encode_json_string	123
util.err	122
util.escaper	125
util.expand_tabs_in_line	123
util.flatten	124
util.intersperse	125
util.lookup_files	123
util.map	125
util.pathname	126
util.rope_last	125
util.rope_to_string	124
util.table_copy	123
util.walk	123, 124, 125
walkable_syntax	9, 17, 21, 208, 210–213
writer	121, 121, 170, 171, 216
writer->active_attributes	179, 179–181
writer->attributes	179
writer->block_html_comment	177
writer->block_html_element	177
writer->blockquote	178
writer->bulletitem	176
writer->bulletlist	175
writer->citations	217
writer->code	174
writer->contentblock	221
writer->defer_call	183, 183
writer->definitionlist	224
writer->display_math	249
writer->div_begin	233
writer->div_end	233

writer->document	178
writer->ellipsis	172
writer->emphasis	177
writer->escape	174
writer->escape_minimal	173
writer->escape_programmatic_text	173
writer->escape_typographic_text	173
writer->escaped_chars	173, 173
writer->escaped_minimal_strings	173, 173
writer->escaped_strings	173
writer->escaped_uri_chars	173, 173
writer->fancyitem	226
writer->fancylist	226
writer->fencedCode	230
writer->get_state	183
writer->hard_line_break	172
writer->heading	181
writer->identifier	174
writer->image	175
writer->inline_html_comment	177
writer->inline_html_tag	177
writer->inline_math	249
writer->interblocksep	172
writer->is_writing	171, 171
writer->jekyllData	252
writer->lineblock	237
writer->link	175
writer->math	174
writer->nbsp	172
writer->note	241
writer->options	171
writer->ordereditem	176
writer->orderedlist	176
writer->pack	172, 215
writer->paragraph	172
writer->plain	172
writer->pop_attributes	179, 180, 181
writer->push_attributes	179, 180, 181
writer->rawBlock	230
writer->rawInline	246
writer->set_state	183
writer->slice_begin	171

<code>writer-&gt;slice_end</code>	171
<code>writer-&gt;space</code>	172
<code>writer-&gt;span</code>	217
<code>writer-&gt;strike_through</code>	247
<code>writer-&gt;string</code>	174
<code>writer-&gt;strong</code>	178
<code>writer-&gt;subscript</code>	247
<code>writer-&gt;suffix</code>	172
<code>writer-&gt;superscript</code>	248
<code>writer-&gt;table</code>	244
<code>writer-&gt;thematic_break</code>	173
<code>writer-&gt;textbox</code>	178
<code>writer-&gt;uri</code>	174
<code>writer-&gt;verbatim</code>	178
<code>writer.new</code>	171, 171