

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.23.0-0-g0b22f91
2023-04-27

Contents

1	Introduction	1	3	Implementation	126
1.1	Requirements	2	3.1	Lua Implementation	127
1.2	Feedback	6	3.2	Plain T _E X Implementation	266
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation	285
2	Interfaces	7	3.4	ConT _E Xt Implementation	313
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	47			
2.3	L ^A T _E X Interface	98			
2.4	ConT _E Xt Interface	123			
				References	320
				Index	321

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	44
4	Various formats of mathematical formulae	106
5	The banner of the Markdown package	107
6	A pushdown automaton that recognizes T _E X comments	192

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

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

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

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

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

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

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

```
17   if not ran_ok then
```

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

```

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

```

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

```

21 local md5 = require("md5");

```

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

```

22 (function()

```

If Kpathsea has not been loaded before or if Lua_{TEX} has not yet been initialized, configure Kpathsea on top of loading it. Since ConT_{EX}t 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 Lua_{TEX} engine [1, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

lua-uni-algos A package that implements Unicode case-folding in T_{EX} Live \geq 2020.

```

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/luatinyaml 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 \leq 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```

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 \geq 2008).

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

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

Unless you convert markdown documents to TeX manually using the Lua command-line interface (see Section 2.1.6), 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^AT_EX Requirements

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

2 Interfaces

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

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

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

2.1 Lua Interface

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

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

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

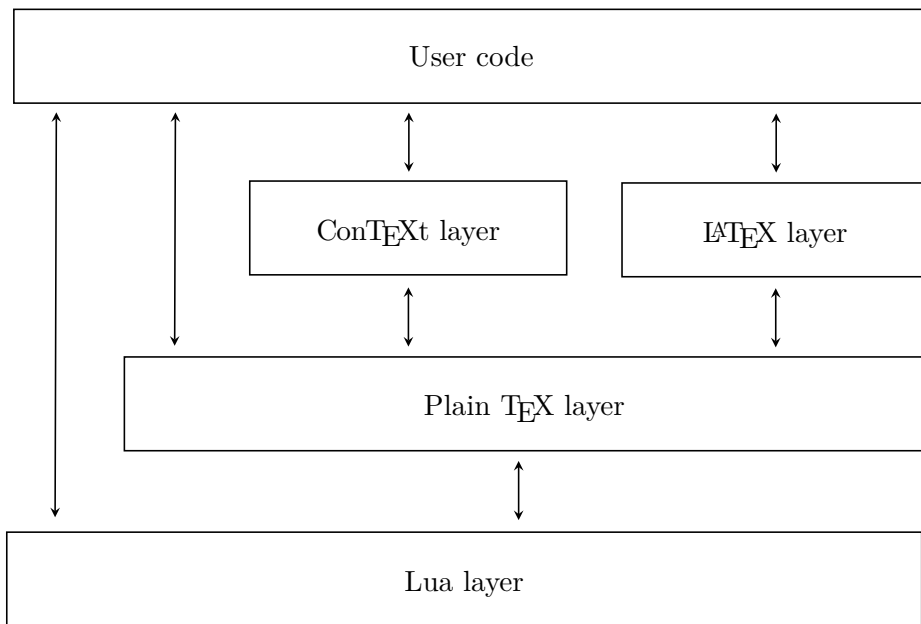


Figure 1: A block diagram of the Markdown package

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     "ULorStarLine",
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         { markdown }
149         { failed-typecheck-for-boolean-option }
150         { #1 }
151         \l_tmpa_tl
152       }
153     }
154   }
155 }
156 \msg_new:nnn
157 { markdown }
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             { markdown }
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         { markdown }
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 { markdown }
226 { unknown-option-type }
227 {
228     Option~#1~has~unknown~type~#2.
229 }
230 \msg_new:nnn
231 { markdown }
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         { markdown }
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 }

```

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

```

314 \cs_new:Nn \@@_with_various_cases:n
315 {
316   \seq_clear:N
317     \l_tmpa_seq
318   \seq_map_inline:Nn
319     \g_@@_cases_seq
320     {
321       \tl_set:Nn
322         \l_tmpa_tl
323         { #1 }
324       \use:c { ##1 }
325       \l_tmpa_tl
326       \seq_put_right:NV
327         \l_tmpa_seq
328         \l_tmpa_tl
329     }
330   \seq_map_inline:Nn
331     \l_tmpa_seq
332     { #2 }
333 }

```

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

```

334 \cs_new:Nn \@@_with_various_cases_break:
335 {
336   \seq_map_break:
337 }

```

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

```

338 \seq_new:N \g_@@_cases_seq
339 \cs_new:Nn \@@_camel_case:N

```

```

340 {
341   \regex_replace_all:nnN
342     { _ ([a-z]) }
343     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
344     #1
345   \tl_set:Nx
346     #1
347     { #1 }
348 }
349 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
350 \cs_new:Nn \@@_snake_case:N
351 {
352   \regex_replace_all:nnN
353     { ([a-z])([A-Z]) }
354     { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
355     #1
356   \tl_set:Nx
357     #1
358     { #1 }
359 }
360 \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.

```

361 \@@_add_lua_option:nnn
362 { cacheDir }
363 { path }
364 { \markdownOptionOutputDir / _markdown_ \jobname }
365 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.

```
366 \@@_add_lua_option:nnn
367   { contentBlocksLanguageMap }
368   { path }
369   { markdown-languages.json }
370 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.

```
371 \@@_add_lua_option:nnn
372   { debugExtensionsFileName }
373   { path }
374   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
375 defaultOptions.debugExtensionsFileName = "debug-extensions.json"
```

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

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

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

```
376 \@@_add_lua_option:nnn
377   { frozenCacheFileName }
378   { path }
379   { \markdownOptionCacheDir / frozenCache.tex }
380 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.

```
381 \@@_add_lua_option:nnn
382 { blankBeforeBlockquote }
383 { boolean }
384 { false }

385 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.

```
386 \@@_add_lua_option:nnn
387 { blankBeforeCodeFence }
388 { boolean }
389 { false }

390 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.

```
391 \@@_add_lua_option:nnn
392 { blankBeforeDivFence }
393 { boolean }
394 { false }

395 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.

```
396 \@@_add_lua_option:nnn
397 { blankBeforeHeading }
398 { boolean }
399 { false }
400 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.

```
401 \@@_add_lua_option:nnn
402 { bracketedSpans }
403 { boolean }
404 { false }
405 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.

```
406 \@@_add_lua_option:nnn
407 { breakableBlockquotes }
408 { boolean }
409 { false }
410 defaultOptions.breakableBlockquotes = false
```

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

`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.

```
411 \@@_add_lua_option:nnn
412 { citationNbsps }
413 { boolean }
414 { true }

415 defaultOptions.citationNbsps = true
```

`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.

```
416 \@@_add_lua_option:nnn
417 { citations }
418 { boolean }
419 { false }

420 defaultOptions.citations = false
```

⁷See <https://pandoc.org/MANUAL.html#extension-citations>.

`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.``
```

```
421 \@@_add_lua_option:nnn  
422 { codeSpans }  
423 { boolean }  
424 { true }  
  
425 defaultOptions.codeSpans = true
```

`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.

```
426 \@@_add_lua_option:nnn
427 { contentBlocks }
428 { boolean }
429 { false }

430 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.

```
431 \@@_add_lua_option:nnn
432 { debugExtensions }
433 { boolean }
434 { false }

435 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.

```
436 \@@_add_lua_option:nnn
437 { definitionLists }
438 { boolean }
439 { false }

440 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.

```
441 \@@_add_lua_option:nnn
442 { eagerCache }
443 { boolean }
444 { true }
445 defaultOptions.eagerCache = true
```

`expectJekyllData=true, false`

default: `false`

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

```
\documentclass{article}
\usepackage[jekyllData]{markdown}
\begin{document}
\begin{markdown}

- this
- is
- YAML
...
- followed
- by
- Markdown
```

```

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

```

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

```

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

```

```

446 \@@_add_lua_option:nnn
447 { expectJekyllData }
448 { boolean }
449 { false }

450 defaultOptions.expectJekyllData = false

```

`extensions=<filenames>`

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

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



```

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

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

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

return strike_through

```

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

```

451 metadata.user_extension_api_version = 2
452 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 <#lua-user-extensions>.
```

```
453 \cs_generate_variant:Nn
454 \@@_add_lua_option:nnn
455 { nnV }
456 \@@_add_lua_option:nnV
457 { extensions }
458 { clist }
459 \c_empty_clist
460 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.

```
461 \@@_add_lua_option:nnn
462 { fancyLists }
463 { boolean }
464 { false }
465 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>
```

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

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

false Disable the commonmark fenced code block extension.

```
466 \@@_add_lua_option:nnn
467 { fencedCode }
468 { boolean }
469 { false }

470 defaultOptions.fencedCode = false
```

fencedCodeAttributes=true, false default: false

true Enable the Pandoc fenced code attribute syntax extension⁹:

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

false Disable the Pandoc fenced code attribute syntax extension.

```
471 \@@_add_lua_option:nnn
472 { fencedCodeAttributes }
473 { boolean }
474 { false }

475 defaultOptions.fencedCodeAttributes = false
```

fencedDivs=true, false default: false

true Enable the Pandoc fenced div syntax extension¹⁰:

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

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

¹⁰See https://pandoc.org/MANUAL.html#extension-fenced_divs.

```
And another.
:::::
```

false Disable the Pandoc fenced div syntax extension.

```
476 \@@_add_lua_option:nnn
477   { fencedDivs }
478   { boolean }
479   { false }

480 defaultOptions.fencedDivs = false
```

finalizeCache=true, false

default: false

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

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

```
481 \@@_add_lua_option:nnn
482   { finalizeCache }
483   { boolean }
484   { false }

485 defaultOptions.finalizeCache = false
```

frozenCacheCounter=<number>

default: 0

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

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

```
486 \@@_add_lua_option:nnn
487   { frozenCacheCounter }
488   { counter }
489   { 0 }

490 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.

```
491 \@@_add_lua_option:nnn
492 { hardLineBreaks }
493 { boolean }
494 { 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.

```
495 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.

```
496 \@@_add_lua_option:nnn
497 { hashEnumerators }
498 { boolean }
499 { false }

500 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.

```

501 \@@_add_lua_option:nnn
502   { headerAttributes }
503   { boolean }
504   { false }

505 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.

```

506 \@@_add_lua_option:nnn
507   { html }
508   { boolean }
509   { false }

510 defaultOptions.html = false

```

`hybrid=true, false` default: false

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

```

511 \@@_add_lua_option:nnn
512   { hybrid }
513   { boolean }
514   { false }

515 defaultOptions.hybrid = false

```

`inlineCodeAttributes=true, false`

default: false

`true` Enable the Pandoc inline code span attribute extension¹¹:

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

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

```
516 \@@_add_lua_option:nnn
517 { inlineCodeAttributes }
518 { boolean }
519 { false }

520 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false`

default: false

`true` Enable the Pandoc inline note syntax extension¹²:

```
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.

```
521 \@@_add_lua_option:nnn
522 { inlineFootnotes }
523 { boolean }
524 { false }
525 \@@_add_lua_option:nnn
526 { inlineNotes }
527 { boolean }
528 { false }

529 defaultOptions.inlineFootnotes = false
530 defaultOptions.inlineNotes = false
```

¹¹See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.

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

`jeekyllData=true, false`

default: false

true Enable the Pandoc YAML metadata block syntax extension¹³ 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.

```
531 \@@_add_lua_option:nnn
532 { jeekyllData }
533 { boolean }
534 { false }
535 defaultOptions.jekyllData = false
```

`linkAttributes=true, false`

default: false

true Enable the Pandoc link and image attribute syntax extension¹⁴:

```
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.

```
536 \@@_add_lua_option:nnn
537 { linkAttributes }
538 { boolean }
539 { false }
540 defaultOptions.linkAttributes = false
```

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

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

`lineBlocks=true, false`

default: false

`true` Enable the Pandoc line block syntax extension¹⁵:

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

`false` Disable the Pandoc line block syntax extension.

```
541 \@@_add_lua_option:nnn
542 { lineBlocks }
543 { boolean }
544 { false }

545 defaultOptions.lineBlocks = false
```

`notes=true, false`

default: false

`true` Enable the Pandoc note syntax extension¹⁶:

```
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.

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

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

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

```
546 \@@_add_lua_option:nnn
547   { footnotes }
548   { boolean }
549   { false }
550 \@@_add_lua_option:nnn
551   { notes }
552   { boolean }
553   { false }

554 defaultOptions.footnotes = false
555 defaultOptions.notes = false
```

`pipeTables=true, false`

default: false

true Enable the PHP Markdown pipe table syntax extension:

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

false Disable the PHP Markdown pipe table syntax extension.

```
556 \@@_add_lua_option:nnn
557   { pipeTables }
558   { boolean }
559   { false }

560 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.

```
561 \@@_add_lua_option:nnn
562   { preserveTabs }
563   { boolean }
564   { false }

565 defaultOptions.preserveTabs = false
```

`rawAttribute=true, false`

default: false

`true` Enable the Pandoc raw attribute syntax extension¹⁷:

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

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

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

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

`false` Disable the Pandoc raw attribute syntax extension.

```
566 \@@_add_lua_option:nmn
567 { rawAttribute }
568 { boolean }
569 { false }

570 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!
```

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

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

**false**      Disable relative references in autolinks.

```
571 \@@_add_lua_option:nnn
572 { relativeReferences }
573 { boolean }
574 { false }
575 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.

```
576 \@@_add_lua_option:nnn
577 { shiftHeadings }
578 { number }
579 { 0 }
580 defaultOptions.shiftHeadings = 0
```

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

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

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

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

```
581 \@@_add_lua_option:nnn
582 { slice }
583 { slice }
584 { ^~$ }
585 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.

```
586 \@@_add_lua_option:nnn
587 { smartEllipses }
588 { boolean }
589 { false }

590 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

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

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

```
591 \@@_add_lua_option:nnn
592 { startNumber }
593 { boolean }
594 { true }

595 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.

```
596 \@@_add_lua_option:nnn
597 { strikeThrough }
598 { boolean }
599 { false }

600 defaultOptions.strikeThrough = false
```

---

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

`stripIndent=true, false`

default: `false`

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

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

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

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

`subscripts=true, false`

default: `false`

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

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

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

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

---

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

`superscripts=true, false`

default: false

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

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

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

```
611 \@@_add_lua_option:nnn
612 { superscripts }
613 { boolean }
614 { false }

615 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      |

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

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

```
616 \@@_add_lua_option:nnn
617 { tableCaptions }
618 { boolean }
619 { false }

620 defaultOptions.tableCaptions = false
```

<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).

`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.

```
621 \@@_add_lua_option:nnn
622 { taskLists }
623 { boolean }
624 { false }

625 defaultOptions.taskLists = false
```

`texComments=true, false`

default: `false`

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

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

Always enabled when `hybrid` is enabled.

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

```
626 \@@_add_lua_option:nnn
627 { texComments }
628 { boolean }
629 { false }

630 defaultOptions.texComments = false
```

---

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



`texMathDollars=true, false`

default: false

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

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

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

```
631 \@@_add_lua_option:nmn
632 { texMathDollars }
633 { boolean }
634 { false }

635 defaultOptions.texMathDollars = false
```

`texMathDoubleBackslash=true, false`

default: false

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

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

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

```
636 \@@_add_lua_option:nmn
637 { texMathDoubleBackslash }
638 { boolean }
639 { false }

640 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false`

default: false

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

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

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

<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).

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

```

641 \@@_add_lua_option:nnn
642 { texMathSingleBackslash }
643 { boolean }
644 { false }

645 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.

```

646 \@@_add_lua_option:nnn
647 { tightLists }
648 { boolean }
649 { true }

650 defaultOptions.tightLists = true

```

`underscores=true, false`

default: true

**true** Both underscores and asterisks can be used to denote emphasis and strong emphasis:

```

single asterisks
single underscores
double asterisks
__double underscores__

```

`false` Only asterisks can be used to denote emphasis and strong emphasis. This makes it easy to write math with the `hybrid` option without the need to constantly escape subscripts.

```

651 \@@_add_lua_option:nnn
652 { underscores }
653 { boolean }
654 { true }
655 \ExplSyntaxOff

656 defaultOptions.underscores = true

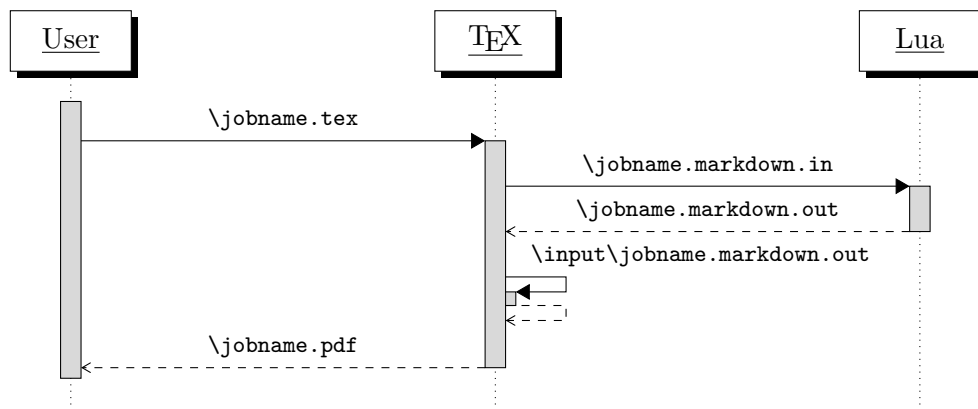
```

### 2.1.6 Command-Line Interface

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

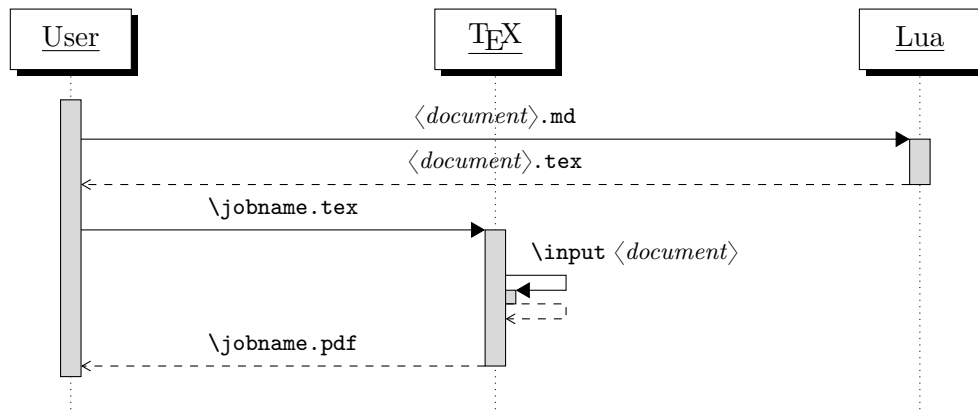
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{T}_{\text{E}}\text{X}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{T}_{\text{E}}\text{X}$  engine includes a Lua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

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



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

657



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

```

658 local HELP_STRING = [[
659 Usage: texlua]] .. arg[0] .. [[[OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
660 where OPTIONS are documented in the Lua interface section of the
661 technical Markdown package documentation.
662
663 When OUTPUT_FILE is unspecified, the result of the conversion will be
664 written to the standard output. When INPUT_FILE is also unspecified, the
665 result of the conversion will be read from the standard input.
666
667 Report bugs to: witiko@mail.muni.cz
668 Markdown package home page: <https://github.com/witiko/markdown>]]
669
670 local VERSION_STRING = [[
671 markdown-cli.lua (Markdown)]] .. metadata.version .. [[
672
673 Copyright (C)]] .. table.concat(metadata.copyright,
674 "\nCopyright (C) ") .. [[
675
676 License:]] .. metadata.license
677
678 local function warn(s)
679 io.stderr:write("Warning: " .. s .. "\n") end
680
681 local function error(s)
682 io.stderr:write("Error: " .. s .. "\n")
683 os.exit(1)
684 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`.

```
685 local function camel_case(option_name)
686 local cased_option_name = option_name:gsub("_(%l)", function(match)
687 return match:sub(2, 2):upper()
688 end)
689 return cased_option_name
690 end
691
692 local function snake_case(option_name)
693 local cased_option_name = option_name:gsub("%l%u", function(match)
694 return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
695 end)
696 return cased_option_name
697 end
698
699 local cases = {camel_case, snake_case}
700 local various_case_options = {}
701 for option_name, _ in pairs(defaultOptions) do
702 for _, case in ipairs(cases) do
703 various_case_options[case(option_name)] = option_name
704 end
705 end
706
707 local process_options = true
708 local options = {}
709 local input_filename
710 local output_filename
711 for i = 1, #arg do
712 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.

```
713 if arg[i] == "--" then
714 process_options = false
715 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.

```
716 elseif arg[i]:match("=") then
717 local key, value = arg[i]:match("(.-)=(.*)")
718 if defaultOptions[key] == nil and
719 various_case_options[key] ~= nil then
720 key = various_case_options[key]
```

```
721 end
```

The `defaultOptions` table is consulted to identify whether  $\langle value \rangle$  should be parsed as a string, number, table, or boolean.

```
722 local default_type = type(defaultOptions[key])
723 if default_type == "boolean" then
724 options[key] = (value == "true")
725 elseif default_type == "number" then
726 options[key] = tonumber(value)
727 elseif default_type == "table" then
728 options[key] = {}
729 for item in value:gmatch("[^,]+") do
730 table.insert(options[key], item)
731 end
732 else
733 if default_type ~= "string" then
734 if default_type == "nil" then
735 warn('Option "' .. key .. '" not recognized.')
736 else
737 warn('Option "' .. key .. '" type not recognized, please file ' ..
738 'a report to the package maintainer.')
739 end
740 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
741 key .. '" as a string.')
742 end
743 options[key] = value
744 end
745 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.

```
746 elseif arg[i] == "--help" or arg[i] == "-h" then
747 print(HELP_STRING)
748 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.

```
749 elseif arg[i] == "--version" or arg[i] == "-v" then
750 print(VERSION_STRING)
751 os.exit()
752 end
753 end
```

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

```

754 if input_filename == nil then
755 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.

```

756 elseif output_filename == nil then
757 output_filename = arg[i]
758 else
759 error('Unexpected argument: "' .. arg[i] .. "'.')
760 end
761 ::continue::
762 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:

```



```

## 2.2 Plain $\text{\TeX}$ Interface

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

```

763 \def\markdownLastModified{((LASTMODIFIED))}%
764 \def\markdownVersion{((VERSION))}%

```

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

```



```

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

### 2.2.1 Typesetting Markdown

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

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

```
765 \let\markdownBegin\relax
```

```
766 \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 TeX.

```
767 \let\markdownInput\relax
```

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

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

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

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

```
768 \let\markdownEscape\relax
```

## 2.2.2 Options

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

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

```
769 \ExplSyntaxOn
```

```
770 \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.

```
771 \prop_new:N \g_@@_plain_tex_option_types_prop
```

```
772 \prop_new:N \g_@@_default_plain_tex_options_prop
```

```
773 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
```

```
774 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
```

```
775 \cs_new:Nn
```

```
776 \@@_add_plain_tex_option:nnn
```

```
777 {
```

```
778 \@@_add_option:Vnnn
```

```
779 \c_@@_option_layer_plain_tex_tl
```

```
780 { #1 }
```

```
781 { #2 }
```

```
782 { #3 }
```

```
783 }
```

**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`.

```
784 \@@_add_plain_tex_option:nnn
785 { frozenCache }
786 { boolean }
787 { 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 `TeX` engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
788 \@@_add_plain_tex_option:nnn
789 { helperScriptFileName }
790 { path }
791 { \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.

```
792 \str_new:N
793 \g_luabridge_helper_script_filename_str
794 \tl_gset:Nn
795 \g_luabridge_helper_script_filename_str
796 { \markdownOptionHelperScriptFileName }
```

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

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

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

```
801 \@@_add_plain_tex_option:nnn
802 { outputTempFileName }
803 { path }
804 { \jobname.markdown.out }
```

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

```
805 \str_new:N
806 \g_luabridge_standard_output_filename_str
807 \tl_gset:Nn
808 \g_luabridge_standard_output_filename_str
809 { \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  $\TeX$  in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `helperScriptFileName` macro.

```
810 \@@_add_plain_tex_option:nnn
811 { errorTempFileName }
812 { path }
813 { \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.

```
814 \str_new:N
815 \g_luabridge_error_output_filename_str
816 \tl_gset:Nn
817 \g_luabridge_error_output_filename_str
818 { \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.

```

819 \@@_add_plain_tex_option:nnn
820 { outputDir }
821 { path }
822 { . }

```

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.

```

823 \cs_new:Nn \@@_plain_tex_define_option_commands:
824 {
825 \seq_map_inline:Nn
826 \g_@@_option_layers_seq
827 {
828 \seq_map_inline:cn
829 { g_@@_ ##1 _options_seq }
830 {
831 \@@_plain_tex_define_option_command:n
832 { #####1 }
833 }
834 }
835 }
836 \cs_new:Nn \@@_plain_tex_define_option_command:n
837 {
838 \@@_get_default_option_value:nN
839 { #1 }
840 \l_tmpa_tl
841 \@@_set_option_value:nV
842 { #1 }
843 \l_tmpa_tl
844 }
845 \cs_new:Nn
846 \@@_set_option_value:nn
847 {
848 \@@_define_option:n
849 { #1 }
850 \@@_get_option_type:nN
851 { #1 }

```

```

852 \l_tmpa_tl
853 \str_if_eq:NNTF
854 \c_@@_option_type_counter_tl
855 \l_tmpa_tl
856 {
857 \@@_option_tl_to_csname:nN
858 { #1 }
859 \l_tmpa_tl
860 \int_gset:cn
861 { \l_tmpa_tl }
862 { #2 }
863 }
864 {
865 \@@_option_tl_to_csname:nN
866 { #1 }
867 \l_tmpa_tl
868 \cs_set:cpn
869 { \l_tmpa_tl }
870 { #2 }
871 }
872 }
873 \cs_generate_variant:Nn
874 \@@_set_option_value:nn
875 { nV }
876 \cs_new:Nn
877 \@@_define_option:n
878 {
879 \@@_option_tl_to_csname:nN
880 { #1 }
881 \l_tmpa_tl
882 \cs_if_free:cT
883 { \l_tmpa_tl }
884 {
885 \@@_get_option_type:nN
886 { #1 }
887 \l_tmpb_tl
888 \str_if_eq:NNT
889 \c_@@_option_type_counter_tl
890 \l_tmpb_tl
891 {
892 \@@_option_tl_to_csname:nN
893 { #1 }
894 \l_tmpa_tl
895 \int_new:c
896 { \l_tmpa_tl }
897 }
898 }

```

```

899 }
900 \@@_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`.

```

901 \seq_gput_right:Nn
902 \g_@@_plain_tex_options_seq
903 { stripPercentSigns }
904 \prop_gput:Nnn
905 \g_@@_plain_tex_option_types_prop
906 { stripPercentSigns }
907 { boolean }
908 \prop_gput:Nnx
909 \g_@@_default_plain_tex_options_prop
910 { stripPercentSigns }
911 { false }
912 \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.

```

913 \ExplSyntaxOn
914 \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.

```

915 \prop_new:N \g_@@_renderer_arities_prop
916 \ExplSyntaxOff

```

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

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

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

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

```

917 \def\markdownRendererAttributeIdentifier{%
918 \markdownRendererAttributeIdentifierPrototype}%
919 \ExplSyntaxOn
920 \seq_gput_right:Nn
921 \g_@@_renderers_seq
922 { attributeIdentifier }
923 \prop_gput:Nnn
924 \g_@@_renderer_arities_prop
925 { attributeIdentifier }
926 { 1 }
927 \ExplSyntaxOff
928 \def\markdownRendererAttributeClassName{%
929 \markdownRendererAttributeClassNamePrototype}%
930 \ExplSyntaxOn
931 \seq_gput_right:Nn
932 \g_@@_renderers_seq
933 { attributeClassName }
934 \prop_gput:Nnn
935 \g_@@_renderer_arities_prop
936 { attributeClassName }
937 { 1 }
938 \ExplSyntaxOff
939 \def\markdownRendererAttributeKeyValue{%
940 \markdownRendererAttributeKeyValuePrototype}%
941 \ExplSyntaxOn
942 \seq_gput_right:Nn
943 \g_@@_renderers_seq
944 { attributeKeyValue }
945 \prop_gput:Nnn
946 \g_@@_renderer_arities_prop
947 { attributeKeyValue }
948 { 2 }
949 \ExplSyntaxOff

```

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

```

950 \def\markdownRendererBlockQuoteBegin{%
951 \markdownRendererBlockQuoteBeginPrototype}%

```

```

952 \ExplSyntaxOn
953 \seq_gput_right:Nn
954 \g_@@_renderers_seq
955 { blockQuoteBegin }
956 \prop_gput:Nnn
957 \g_@@_renderer_arities_prop
958 { blockQuoteBegin }
959 { 0 }
960 \ExplSyntaxOff

```

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

```

961 \def\markdownRendererBlockQuoteEnd{%
962 \markdownRendererBlockQuoteEndPrototype}%
963 \ExplSyntaxOn
964 \seq_gput_right:Nn
965 \g_@@_renderers_seq
966 { blockQuoteEnd }
967 \prop_gput:Nnn
968 \g_@@_renderer_arities_prop
969 { blockQuoteEnd }
970 { 0 }
971 \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.

```

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

```



```

989 \prop_gput:Nnn
990 \g_@@_renderer_arities_prop
991 { bracketedSpanAttributeContextEnd }
992 { 0 }
993 \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.

```

994 \def\markdownRendererUlBegin{%
995 \markdownRendererUlBeginPrototype}%
996 \ExplSyntaxOn
997 \seq_gput_right:Nn
998 \g_@@_renderers_seq
999 { ulBegin }
1000 \prop_gput:Nnn
1001 \g_@@_renderer_arities_prop
1002 { ulBegin }
1003 { 0 }
1004 \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.

```

1005 \def\markdownRendererUlBeginTight{%
1006 \markdownRendererUlBeginTightPrototype}%
1007 \ExplSyntaxOn
1008 \seq_gput_right:Nn
1009 \g_@@_renderers_seq
1010 { ulBeginTight }
1011 \prop_gput:Nnn
1012 \g_@@_renderer_arities_prop
1013 { ulBeginTight }
1014 { 0 }
1015 \ExplSyntaxOff

```

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

```

1016 \def\markdownRendererUlItem{%
1017 \markdownRendererUlItemPrototype}%
1018 \ExplSyntaxOn
1019 \seq_gput_right:Nn
1020 \g_@@_renderers_seq
1021 { ulItem }

```

```

1022 \prop_gput:Nnn
1023 \g_@@_renderer_arities_prop
1024 { ulItem }
1025 { 0 }
1026 \ExplSyntaxOff

```

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

```

1027 \def\markdownRendererUlItemEnd{%
1028 \markdownRendererUlItemEndPrototype}%
1029 \ExplSyntaxOn
1030 \seq_gput_right:Nn
1031 \g_@@_renderers_seq
1032 { ulItemEnd }
1033 \prop_gput:Nnn
1034 \g_@@_renderer_arities_prop
1035 { ulItemEnd }
1036 { 0 }
1037 \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.

```

1038 \def\markdownRendererUlEnd{%
1039 \markdownRendererUlEndPrototype}%
1040 \ExplSyntaxOn
1041 \seq_gput_right:Nn
1042 \g_@@_renderers_seq
1043 { ulEnd }
1044 \prop_gput:Nnn
1045 \g_@@_renderer_arities_prop
1046 { ulEnd }
1047 { 0 }
1048 \ExplSyntaxOff

```

The `\markdownRendererUlEndTight` 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.

```

1049 \def\markdownRendererUlEndTight{%
1050 \markdownRendererUlEndTightPrototype}%
1051 \ExplSyntaxOn
1052 \seq_gput_right:Nn
1053 \g_@@_renderers_seq
1054 { ulEndTight }
1055 \prop_gput:Nnn

```

```

1056 \g_@@_renderer_arities_prop
1057 { ulEndTight }
1058 { 0 }
1059 \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.

```

1060 \def\markdownRendererInputVerbatim{%
1061 \markdownRendererInputVerbatimPrototype}%
1062 \ExplSyntaxOn
1063 \seq_gput_right:Nn
1064 \g_@@_renderers_seq
1065 { inputVerbatim }
1066 \prop_gput:Nnn
1067 \g_@@_renderer_arities_prop
1068 { inputVerbatim }
1069 { 1 }
1070 \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.

```

1071 \def\markdownRendererInputFencedCode{%
1072 \markdownRendererInputFencedCodePrototype}%
1073 \ExplSyntaxOn
1074 \seq_gput_right:Nn
1075 \g_@@_renderers_seq
1076 { inputFencedCode }
1077 \prop_gput:Nnn
1078 \g_@@_renderer_arities_prop
1079 { inputFencedCode }
1080 { 2 }
1081 \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.

```

1082 \def\markdownRendererCodeSpan{%
1083 \markdownRendererCodeSpanPrototype}%
1084 \ExplSyntaxOn
1085 \seq_gput_right:Nn
1086 \g_@@_renderers_seq

```

```

1087 { codeSpan }
1088 \prop_gput:Nnn
1089 \g_@@_renderer_arities_prop
1090 { codeSpan }
1091 { 1 }
1092 \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.

```

1093 \def\markdownRendererCodeSpanAttributeContextBegin{%
1094 \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1095 \ExplSyntaxOn
1096 \seq_gput_right:Nn
1097 \g_@@_renderers_seq
1098 { codeSpanAttributeContextBegin }
1099 \prop_gput:Nnn
1100 \g_@@_renderer_arities_prop
1101 { codeSpanAttributeContextBegin }
1102 { 0 }
1103 \ExplSyntaxOff
1104 \def\markdownRendererCodeSpanAttributeContextEnd{%
1105 \markdownRendererCodeSpanAttributeContextEndPrototype}%
1106 \ExplSyntaxOn
1107 \seq_gput_right:Nn
1108 \g_@@_renderers_seq
1109 { codeSpanAttributeContextEnd }
1110 \prop_gput:Nnn
1111 \g_@@_renderer_arities_prop
1112 { codeSpanAttributeContextEnd }
1113 { 0 }
1114 \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.

```

1115 \def\markdownRendererContentBlock{%
1116 \markdownRendererContentBlockPrototype}%
1117 \ExplSyntaxOn
1118 \seq_gput_right:Nn

```

```

1119 \g_@@_renderers_seq
1120 { contentBlock }
1121 \prop_gput:Nnn
1122 \g_@@_renderer_arities_prop
1123 { contentBlock }
1124 { 4 }
1125 \ExplSyntaxOff

```

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

```

1126 \def\markdownRendererContentBlockOnlineImage{%
1127 \markdownRendererContentBlockOnlineImagePrototype}%
1128 \ExplSyntaxOn
1129 \seq_gput_right:Nn
1130 \g_@@_renderers_seq
1131 { contentBlockOnlineImage }
1132 \prop_gput:Nnn
1133 \g_@@_renderer_arities_prop
1134 { contentBlockOnlineImage }
1135 { 4 }
1136 \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 `TEX` 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.

```

1137 \def\markdownRendererContentBlockCode{%
1138 \markdownRendererContentBlockCodePrototype}%
1139 \ExplSyntaxOn
1140 \seq_gput_right:Nn
1141 \g_@@_renderers_seq

```

---

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

```

1142 { contentBlockCode }
1143 \prop_gput:Nnn
1144 \g_@@_renderer_arities_prop
1145 { contentBlockCode }
1146 { 5 }
1147 \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.

```

1148 \def\markdownRendererDlBegin{%
1149 \markdownRendererDlBeginPrototype}%
1150 \ExplSyntaxOn
1151 \seq_gput_right:Nn
1152 \g_@@_renderers_seq
1153 { dlBegin }
1154 \prop_gput:Nnn
1155 \g_@@_renderer_arities_prop
1156 { dlBegin }
1157 { 0 }
1158 \ExplSyntaxOff

```

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

```

1159 \def\markdownRendererDlBeginTight{%
1160 \markdownRendererDlBeginTightPrototype}%
1161 \ExplSyntaxOn
1162 \seq_gput_right:Nn
1163 \g_@@_renderers_seq
1164 { dlBeginTight }
1165 \prop_gput:Nnn
1166 \g_@@_renderer_arities_prop
1167 { dlBeginTight }
1168 { 0 }
1169 \ExplSyntaxOff

```

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

```

1170 \def\markdownRendererDlItem{%
1171 \markdownRendererDlItemPrototype}%
1172 \ExplSyntaxOn

```

```

1173 \seq_gput_right:Nn
1174 \g_@@_renderers_seq
1175 { dlItem }
1176 \prop_gput:Nnn
1177 \g_@@_renderer_arities_prop
1178 { dlItem }
1179 { 1 }
1180 \ExplSyntaxOff

```

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

```

1181 \def\markdownRendererDlItemEnd{%
1182 \markdownRendererDlItemEndPrototype}%
1183 \ExplSyntaxOn
1184 \seq_gput_right:Nn
1185 \g_@@_renderers_seq
1186 { dlItemEnd }
1187 \prop_gput:Nnn
1188 \g_@@_renderer_arities_prop
1189 { dlItemEnd }
1190 { 0 }
1191 \ExplSyntaxOff

```

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

```

1192 \def\markdownRendererDlDefinitionBegin{%
1193 \markdownRendererDlDefinitionBeginPrototype}%
1194 \ExplSyntaxOn
1195 \seq_gput_right:Nn
1196 \g_@@_renderers_seq
1197 { dlDefinitionBegin }
1198 \prop_gput:Nnn
1199 \g_@@_renderer_arities_prop
1200 { dlDefinitionBegin }
1201 { 0 }
1202 \ExplSyntaxOff

```

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

```

1203 \def\markdownRendererDlDefinitionEnd{%
1204 \markdownRendererDlDefinitionEndPrototype}%
1205 \ExplSyntaxOn
1206 \seq_gput_right:Nn
1207 \g_@@_renderers_seq
1208 { dlDefinitionEnd }
1209 \prop_gput:Nnn
1210 \g_@@_renderer_arities_prop

```

```

1211 { dlDefinitionEnd }
1212 { 0 }
1213 \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.

```

1214 \def\markdownRendererDlEnd{%
1215 \markdownRendererDlEndPrototype}%
1216 \ExplSyntaxOn
1217 \seq_gput_right:Nn
1218 \g_@@_renderers_seq
1219 { dlEnd }
1220 \prop_gput:Nnn
1221 \g_@@_renderer_arities_prop
1222 { dlEnd }
1223 { 0 }
1224 \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.

```

1225 \def\markdownRendererDlEndTight{%
1226 \markdownRendererDlEndTightPrototype}%
1227 \ExplSyntaxOn
1228 \seq_gput_right:Nn
1229 \g_@@_renderers_seq
1230 { dlEndTight }
1231 \prop_gput:Nnn
1232 \g_@@_renderer_arities_prop
1233 { dlEndTight }
1234 { 0 }
1235 \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.

```

1236 \def\markdownRendererEllipsis{%
1237 \markdownRendererEllipsisPrototype}%
1238 \ExplSyntaxOn
1239 \seq_gput_right:Nn
1240 \g_@@_renderers_seq
1241 { ellipsis }
1242 \prop_gput:Nnn

```



```

1243 \g_@@_renderer_arities_prop
1244 { ellipsis }
1245 { 0 }
1246 \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.

```

1247 \def\markdownRendererEmphasis{%
1248 \markdownRendererEmphasisPrototype}%
1249 \ExplSyntaxOn
1250 \seq_gput_right:Nn
1251 \g_@@_renderers_seq
1252 { emphasis }
1253 \prop_gput:Nnn
1254 \g_@@_renderer_arities_prop
1255 { emphasis }
1256 { 1 }
1257 \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.

```

1258 \def\markdownRendererStrongEmphasis{%
1259 \markdownRendererStrongEmphasisPrototype}%
1260 \ExplSyntaxOn
1261 \seq_gput_right:Nn
1262 \g_@@_renderers_seq
1263 { strongEmphasis }
1264 \prop_gput:Nnn
1265 \g_@@_renderer_arities_prop
1266 { strongEmphasis }
1267 { 1 }
1268 \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.

```

1269 \def\markdownRendererFencedCodeAttributeContextBegin{%
1270 \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1271 \ExplSyntaxOn
1272 \seq_gput_right:Nn

```

```

1273 \g_@@_renderers_seq
1274 { fencedCodeAttributeContextBegin }
1275 \prop_gput:Nnn
1276 \g_@@_renderer_arities_prop
1277 { fencedCodeAttributeContextBegin }
1278 { 0 }
1279 \ExplSyntaxOff
1280 \def\markdownRendererFencedCodeAttributeContextEnd{%
1281 \markdownRendererFencedCodeAttributeContextEndPrototype}%
1282 \ExplSyntaxOn
1283 \seq_gput_right:Nn
1284 \g_@@_renderers_seq
1285 { fencedCodeAttributeContextEnd }
1286 \prop_gput:Nnn
1287 \g_@@_renderer_arities_prop
1288 { fencedCodeAttributeContextEnd }
1289 { 0 }
1290 \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.

```

1291 \def\markdownRendererFencedDivAttributeContextBegin{%
1292 \markdownRendererFencedDivAttributeContextBeginPrototype}%
1293 \ExplSyntaxOn
1294 \seq_gput_right:Nn
1295 \g_@@_renderers_seq
1296 { fencedDivAttributeContextBegin }
1297 \prop_gput:Nnn
1298 \g_@@_renderer_arities_prop
1299 { fencedDivAttributeContextBegin }
1300 { 0 }
1301 \ExplSyntaxOff
1302 \def\markdownRendererFencedDivAttributeContextEnd{%
1303 \markdownRendererFencedDivAttributeContextEndPrototype}%
1304 \ExplSyntaxOn
1305 \seq_gput_right:Nn
1306 \g_@@_renderers_seq
1307 { fencedDivAttributeContextEnd }
1308 \prop_gput:Nnn
1309 \g_@@_renderer_arities_prop
1310 { fencedDivAttributeContextEnd }
1311 { 0 }
1312 \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.

```

1313 \def\markdownRendererHeaderAttributeContextBegin{%
1314 \markdownRendererHeaderAttributeContextBeginPrototype}%
1315 \ExplSyntaxOn
1316 \seq_gput_right:Nn
1317 \g_@@_renderers_seq
1318 { headerAttributeContextBegin }
1319 \prop_gput:Nnn
1320 \g_@@_renderer_arities_prop
1321 { headerAttributeContextBegin }
1322 { 0 }
1323 \ExplSyntaxOff
1324 \def\markdownRendererHeaderAttributeContextEnd{%
1325 \markdownRendererHeaderAttributeContextEndPrototype}%
1326 \ExplSyntaxOn
1327 \seq_gput_right:Nn
1328 \g_@@_renderers_seq
1329 { headerAttributeContextEnd }
1330 \prop_gput:Nnn
1331 \g_@@_renderer_arities_prop
1332 { headerAttributeContextEnd }
1333 { 0 }
1334 \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.

```

1335 \def\markdownRendererHeadingOne{%
1336 \markdownRendererHeadingOnePrototype}%
1337 \ExplSyntaxOn
1338 \seq_gput_right:Nn
1339 \g_@@_renderers_seq
1340 { headingOne }
1341 \prop_gput:Nnn
1342 \g_@@_renderer_arities_prop
1343 { headingOne }
1344 { 1 }
1345 \ExplSyntaxOff

```

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

```
1346 \def\markdownRendererHeadingTwo{%
1347 \markdownRendererHeadingTwoPrototype}%
1348 \ExplSyntaxOn
1349 \seq_gput_right:Nn
1350 \g_@@_renderers_seq
1351 { headingTwo }
1352 \prop_gput:Nnn
1353 \g_@@_renderer_arities_prop
1354 { headingTwo }
1355 { 1 }
1356 \ExplSyntaxOff
```

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

```
1357 \def\markdownRendererHeadingThree{%
1358 \markdownRendererHeadingThreePrototype}%
1359 \ExplSyntaxOn
1360 \seq_gput_right:Nn
1361 \g_@@_renderers_seq
1362 { headingThree }
1363 \prop_gput:Nnn
1364 \g_@@_renderer_arities_prop
1365 { headingThree }
1366 { 1 }
1367 \ExplSyntaxOff
```

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

```
1368 \def\markdownRendererHeadingFour{%
1369 \markdownRendererHeadingFourPrototype}%
1370 \ExplSyntaxOn
1371 \seq_gput_right:Nn
1372 \g_@@_renderers_seq
1373 { headingFour }
1374 \prop_gput:Nnn
1375 \g_@@_renderer_arities_prop
1376 { headingFour }
1377 { 1 }
1378 \ExplSyntaxOff
```

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

```
1379 \def\markdownRendererHeadingFive{%
1380 \markdownRendererHeadingFivePrototype}%
```

```

1381 \ExplSyntaxOn
1382 \seq_gput_right:Nn
1383 \g_@@_renderers_seq
1384 { headingFive }
1385 \prop_gput:Nnn
1386 \g_@@_renderer_arities_prop
1387 { headingFive }
1388 { 1 }
1389 \ExplSyntaxOff

```

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

```

1390 \def\markdownRendererHeadingSix{%
1391 \markdownRendererHeadingSixPrototype}%
1392 \ExplSyntaxOn
1393 \seq_gput_right:Nn
1394 \g_@@_renderers_seq
1395 { headingSix }
1396 \prop_gput:Nnn
1397 \g_@@_renderer_arities_prop
1398 { headingSix }
1399 { 1 }
1400 \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.

```

1401 \def\markdownRendererInlineHtmlComment{%
1402 \markdownRendererInlineHtmlCommentPrototype}%
1403 \ExplSyntaxOn
1404 \seq_gput_right:Nn
1405 \g_@@_renderers_seq
1406 { inlineHtmlComment }
1407 \prop_gput:Nnn
1408 \g_@@_renderer_arities_prop
1409 { inlineHtmlComment }
1410 { 1 }
1411 \ExplSyntaxOff
1412 \def\markdownRendererBlockHtmlCommentBegin{%
1413 \markdownRendererBlockHtmlCommentBeginPrototype}%
1414 \ExplSyntaxOn

```

```

1415 \seq_gput_right:Nn
1416 \g_@@_renderers_seq
1417 { blockHtmlCommentBegin }
1418 \prop_gput:Nnn
1419 \g_@@_renderer_arities_prop
1420 { blockHtmlCommentBegin }
1421 { 0 }
1422 \ExplSyntaxOff
1423 \def\markdownRendererBlockHtmlCommentEnd{%
1424 \markdownRendererBlockHtmlCommentEndPrototype}%
1425 \ExplSyntaxOn
1426 \seq_gput_right:Nn
1427 \g_@@_renderers_seq
1428 { blockHtmlCommentEnd }
1429 \prop_gput:Nnn
1430 \g_@@_renderer_arities_prop
1431 { blockHtmlCommentEnd }
1432 { 0 }
1433 \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.

```

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

```

```

1451 \prop_gput:Nnn
1452 \g_@@_renderer_arities_prop
1453 { inputBlockHtmlElement }
1454 { 1 }
1455 \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.

```

1456 \def\markdownRendererImage{%
1457 \markdownRendererImagePrototype}%
1458 \ExplSyntaxOn
1459 \seq_gput_right:Nn
1460 \g_@@_renderers_seq
1461 { image }
1462 \prop_gput:Nnn
1463 \g_@@_renderer_arities_prop
1464 { image }
1465 { 4 }
1466 \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.

```

1467 \def\markdownRendererImageAttributeContextBegin{%
1468 \markdownRendererImageAttributeContextBeginPrototype}%
1469 \ExplSyntaxOn
1470 \seq_gput_right:Nn
1471 \g_@@_renderers_seq
1472 { imageAttributeContextBegin }
1473 \prop_gput:Nnn
1474 \g_@@_renderer_arities_prop
1475 { imageAttributeContextBegin }
1476 { 0 }
1477 \ExplSyntaxOff
1478 \def\markdownRendererImageAttributeContextEnd{%
1479 \markdownRendererImageAttributeContextEndPrototype}%
1480 \ExplSyntaxOn
1481 \seq_gput_right:Nn
1482 \g_@@_renderers_seq
1483 { imageAttributeContextEnd }
1484 \prop_gput:Nnn
1485 \g_@@_renderer_arities_prop

```

```

1486 { imageAttributeContextEnd }
1487 { 0 }
1488 \ExplSyntaxOff

```

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

```

1489 \def\markdownRendererInterblockSeparator{%
1490 \markdownRendererInterblockSeparatorPrototype}%
1491 \ExplSyntaxOn
1492 \seq_gput_right:Nn
1493 \g_@@_renderers_seq
1494 { interblockSeparator }
1495 \prop_gput:Nnn
1496 \g_@@_renderer_arities_prop
1497 { interblockSeparator }
1498 { 0 }
1499 \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.

```

1500 \def\markdownRendererLineBlockBegin{%
1501 \markdownRendererLineBlockBeginPrototype}%
1502 \ExplSyntaxOn
1503 \seq_gput_right:Nn
1504 \g_@@_renderers_seq
1505 { lineBlockBegin }
1506 \prop_gput:Nnn
1507 \g_@@_renderer_arities_prop
1508 { lineBlockBegin }
1509 { 0 }
1510 \ExplSyntaxOff
1511 \def\markdownRendererLineBlockEnd{%
1512 \markdownRendererLineBlockEndPrototype}%
1513 \ExplSyntaxOn
1514 \seq_gput_right:Nn
1515 \g_@@_renderers_seq
1516 { lineBlockEnd }
1517 \prop_gput:Nnn
1518 \g_@@_renderer_arities_prop
1519 { lineBlockEnd }
1520 { 0 }

```



1521 \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.

```
1522 \ExplSyntaxOn
1523 \cs_new:Npn
1524 \markdownRendererHardLineBreak
1525 {
1526 \cs_if_exist:NTF
1527 \markdownRendererLineBreak
1528 {
1529 \markdownWarning
1530 {
1531 Line~break~renderer~has~been~deprecated,~
1532 to~be~removed~in~Markdown~3.0.0
1533 }
1534 \markdownRendererLineBreak
1535 }
1536 {
1537 \cs_if_exist:NTF
1538 \markdownRendererLineBreakPrototype
1539 {
1540 \markdownWarning
1541 {
1542 Line~break~renderer~prototype~has~been~deprecated,~
1543 to~be~removed~in~Markdown~3.0.0
1544 }
1545 \markdownRendererLineBreakPrototype
1546 }
1547 {
1548 \markdownRendererHardLineBreakPrototype
1549 }
1550 }
1551 }
1552 \seq_gput_right:Nn
1553 \g_@@_renderers_seq
1554 { lineBreak }
1555 \prop_gput:Nnn
1556 \g_@@_renderer_arities_prop
1557 { lineBreak }
1558 { 0 }
1559 \seq_gput_right:Nn
1560 \g_@@_renderers_seq
1561 { hardLineBreak }
```

```

1562 \prop_gput:Nnn
1563 \g_@@_renderer_arities_prop
1564 { hardLineBreak }
1565 { 0 }
1566 \ExplSyntaxOff

```

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

```

1567 \def\markdownRendererLink{%
1568 \markdownRendererLinkPrototype}%
1569 \ExplSyntaxOn
1570 \seq_gput_right:Nn
1571 \g_@@_renderers_seq
1572 { link }
1573 \prop_gput:Nnn
1574 \g_@@_renderer_arities_prop
1575 { link }
1576 { 4 }
1577 \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.

```

1578 \def\markdownRendererLinkAttributeContextBegin{%
1579 \markdownRendererLinkAttributeContextBeginPrototype}%
1580 \ExplSyntaxOn
1581 \seq_gput_right:Nn
1582 \g_@@_renderers_seq
1583 { linkAttributeContextBegin }
1584 \prop_gput:Nnn
1585 \g_@@_renderer_arities_prop
1586 { linkAttributeContextBegin }
1587 { 0 }
1588 \ExplSyntaxOff
1589 \def\markdownRendererLinkAttributeContextEnd{%
1590 \markdownRendererLinkAttributeContextEndPrototype}%
1591 \ExplSyntaxOn
1592 \seq_gput_right:Nn
1593 \g_@@_renderers_seq
1594 { linkAttributeContextEnd }
1595 \prop_gput:Nnn
1596 \g_@@_renderer_arities_prop

```

```

1597 { linkAttributeContextEnd }
1598 { 0 }
1599 \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  $\TeX$  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.

```

1600 \def\markdownRendererDocumentBegin{%
1601 \markdownRendererDocumentBeginPrototype}%
1602 \ExplSyntaxOn
1603 \seq_gput_right:Nn
1604 \g_@@_renderers_seq
1605 { documentBegin }
1606 \prop_gput:Nnn
1607 \g_@@_renderer_arities_prop
1608 { documentBegin }
1609 { 0 }
1610 \ExplSyntaxOff
1611 \def\markdownRendererDocumentEnd{%
1612 \markdownRendererDocumentEndPrototype}%
1613 \ExplSyntaxOn
1614 \seq_gput_right:Nn
1615 \g_@@_renderers_seq
1616 { documentEnd }
1617 \prop_gput:Nnn
1618 \g_@@_renderer_arities_prop
1619 { documentEnd }
1620 { 0 }
1621 \ExplSyntaxOff

```

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

```

1622 \def\markdownRendererNbsp{%
1623 \markdownRendererNbspPrototype}%
1624 \ExplSyntaxOn
1625 \seq_gput_right:Nn
1626 \g_@@_renderers_seq
1627 { nbsp }
1628 \prop_gput:Nnn
1629 \g_@@_renderer_arities_prop

```

```

1630 { nbsp }
1631 { 0 }
1632 \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.

```

1633 \ExplSyntaxOn
1634 \cs_new:Npn
1635 \markdownRendererNote
1636 {
1637 \cs_if_exist:NTF
1638 \markdownRendererFootnote
1639 {
1640 \markdownWarning
1641 {
1642 Footnote~renderer~has~been~deprecated,~
1643 to~be~removed~in~Markdown~3.0.0
1644 }
1645 \markdownRendererFootnote
1646 }
1647 {
1648 \cs_if_exist:NTF
1649 \markdownRendererFootnotePrototype
1650 {
1651 \markdownWarning
1652 {
1653 Footnote~renderer~prototype~has~been~deprecated,~
1654 to~be~removed~in~Markdown~3.0.0
1655 }
1656 \markdownRendererFootnotePrototype
1657 }
1658 {
1659 \markdownRendererNotePrototype
1660 }
1661 }
1662 }
1663 \seq_gput_right:Nn
1664 \g_@@_renderers_seq
1665 { footnote }
1666 \prop_gput:Nnn
1667 \g_@@_renderer_arities_prop
1668 { footnote }
1669 { 1 }

```

```

1670 \seq_gput_right:Nn
1671 \g_@@_renderers_seq
1672 { note }
1673 \prop_gput:Nnn
1674 \g_@@_renderer_arities_prop
1675 { note }
1676 { 1 }
1677 \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.

```

1678 \def\markdownRendererOlBegin{%
1679 \markdownRendererOlBeginPrototype}%
1680 \ExplSyntaxOn
1681 \seq_gput_right:Nn
1682 \g_@@_renderers_seq
1683 { olBegin }
1684 \prop_gput:Nnn
1685 \g_@@_renderer_arities_prop
1686 { olBegin }
1687 { 0 }
1688 \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.

```

1689 \def\markdownRendererOlBeginTight{%
1690 \markdownRendererOlBeginTightPrototype}%
1691 \ExplSyntaxOn
1692 \seq_gput_right:Nn
1693 \g_@@_renderers_seq
1694 { olBeginTight }
1695 \prop_gput:Nnn
1696 \g_@@_renderer_arities_prop
1697 { olBeginTight }
1698 { 0 }
1699 \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](#)).

```

1700 \def\markdownRendererFancyOlBegin{%
1701 \markdownRendererFancyOlBeginPrototype}%
1702 \ExplSyntaxOn
1703 \seq_gput_right:Nn
1704 \g_@@_renderers_seq
1705 { fancyOlBegin }
1706 \prop_gput:Nnn
1707 \g_@@_renderer_arities_prop
1708 { fancyOlBegin }
1709 { 2 }
1710 \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.

```

1711 \def\markdownRendererFancyOlBeginTight{%
1712 \markdownRendererFancyOlBeginTightPrototype}%
1713 \ExplSyntaxOn
1714 \seq_gput_right:Nn
1715 \g_@@_renderers_seq
1716 { fancyOlBeginTight }
1717 \prop_gput:Nnn
1718 \g_@@_renderer_arities_prop
1719 { fancyOlBeginTight }
1720 { 2 }
1721 \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.

```

1722 \def\markdownRendererOlItem{%
1723 \markdownRendererOlItemPrototype}%
1724 \ExplSyntaxOn
1725 \seq_gput_right:Nn
1726 \g_@@_renderers_seq
1727 { olItem }
1728 \prop_gput:Nnn
1729 \g_@@_renderer_arities_prop
1730 { olItem }
1731 { 0 }
1732 \ExplSyntaxOff

```

The `\markdownRendererO1ItemEnd` 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.

```

1733 \def\markdownRendererO1ItemEnd{%
1734 \markdownRendererO1ItemEndPrototype}%
1735 \ExplSyntaxOn
1736 \seq_gput_right:Nn
1737 \g_@@_renderers_seq
1738 { olItemEnd }
1739 \prop_gput:Nnn
1740 \g_@@_renderer_arities_prop
1741 { olItemEnd }
1742 { 0 }
1743 \ExplSyntaxOff

```

The `\markdownRendererO1ItemWithNumber` 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.

```

1744 \def\markdownRendererO1ItemWithNumber{%
1745 \markdownRendererO1ItemWithNumberPrototype}%
1746 \ExplSyntaxOn
1747 \seq_gput_right:Nn
1748 \g_@@_renderers_seq
1749 { olItemWithNumber }
1750 \prop_gput:Nnn
1751 \g_@@_renderer_arities_prop
1752 { olItemWithNumber }
1753 { 1 }
1754 \ExplSyntaxOff

```

The `\markdownRendererFancyO1Item` 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.

```

1755 \def\markdownRendererFancyO1Item{%
1756 \markdownRendererFancyO1ItemPrototype}%
1757 \ExplSyntaxOn
1758 \seq_gput_right:Nn
1759 \g_@@_renderers_seq
1760 { fancyO1Item }
1761 \prop_gput:Nnn
1762 \g_@@_renderer_arities_prop
1763 { fancyO1Item }
1764 { 0 }
1765 \ExplSyntaxOff

```

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

```
1766 \def\markdownRendererFancyO1ItemEnd{%
1767 \markdownRendererFancyO1ItemEndPrototype}%
1768 \ExplSyntaxOn
1769 \seq_gput_right:Nn
1770 \g_@@_renderers_seq
1771 { fancyO1ItemEnd }
1772 \prop_gput:Nnn
1773 \g_@@_renderer_arities_prop
1774 { fancyO1ItemEnd }
1775 { 0 }
1776 \ExplSyntaxOff
```

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

```
1777 \def\markdownRendererFancyO1ItemWithNumber{%
1778 \markdownRendererFancyO1ItemWithNumberPrototype}%
1779 \ExplSyntaxOn
1780 \seq_gput_right:Nn
1781 \g_@@_renderers_seq
1782 { fancyO1ItemWithNumber }
1783 \prop_gput:Nnn
1784 \g_@@_renderer_arities_prop
1785 { fancyO1ItemWithNumber }
1786 { 1 }
1787 \ExplSyntaxOff
```

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

```
1788 \def\markdownRendererO1End{%
1789 \markdownRendererO1EndPrototype}%
1790 \ExplSyntaxOn
1791 \seq_gput_right:Nn
1792 \g_@@_renderers_seq
1793 { olEnd }
1794 \prop_gput:Nnn
1795 \g_@@_renderer_arities_prop
1796 { olEnd }
1797 { 0 }
1798 \ExplSyntaxOff
```



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

```

1799 \def\markdownRendererOlEndTight{%
1800 \markdownRendererOlEndTightPrototype}%
1801 \ExplSyntaxOn
1802 \seq_gput_right:Nn
1803 \g_@@_renderers_seq
1804 { olEndTight }
1805 \prop_gput:Nnn
1806 \g_@@_renderer_arities_prop
1807 { olEndTight }
1808 { 0 }
1809 \ExplSyntaxOff

```

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

```

1810 \def\markdownRendererFancyOlEnd{%
1811 \markdownRendererFancyOlEndPrototype}%
1812 \ExplSyntaxOn
1813 \seq_gput_right:Nn
1814 \g_@@_renderers_seq
1815 { fancyOlEnd }
1816 \prop_gput:Nnn
1817 \g_@@_renderer_arities_prop
1818 { fancyOlEnd }
1819 { 0 }
1820 \ExplSyntaxOff

```

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

```

1821 \def\markdownRendererFancyOlEndTight{%
1822 \markdownRendererFancyOlEndTightPrototype}%
1823 \ExplSyntaxOn
1824 \seq_gput_right:Nn
1825 \g_@@_renderers_seq
1826 { fancyOlEndTight }
1827 \prop_gput:Nnn
1828 \g_@@_renderer_arities_prop
1829 { fancyOlEndTight }

```

```

1830 { 0 }
1831 \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.

```

1832 \def\markdownRendererCite{%
1833 \markdownRendererCitePrototype}%
1834 \ExplSyntaxOn
1835 \seq_gput_right:Nn
1836 \g_@@_renderers_seq
1837 { cite }
1838 \prop_gput:Nnn
1839 \g_@@_renderer_arities_prop
1840 { cite }
1841 { 1 }
1842 \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.

```

1843 \def\markdownRendererInputRawInline{%
1844 \markdownRendererInputRawInlinePrototype}%
1845 \ExplSyntaxOn
1846 \seq_gput_right:Nn
1847 \g_@@_renderers_seq
1848 { inputRawInline }
1849 \prop_gput:Nnn
1850 \g_@@_renderer_arities_prop
1851 { inputRawInline }
1852 { 2 }
1853 \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.

```

1854 \def\markdownRendererInputRawBlock{%

```

```

1855 \markdownRendererInputRawBlockPrototype}%
1856 \ExplSyntaxOn
1857 \seq_gput_right:Nn
1858 \g_@@_renderers_seq
1859 { inputRawBlock }
1860 \prop_gput:Nnn
1861 \g_@@_renderer_arities_prop
1862 { inputRawBlock }
1863 { 2 }
1864 \ExplSyntaxOff

```

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

```

1865 \def\markdownRendererSectionBegin{%
1866 \markdownRendererSectionBeginPrototype}%
1867 \ExplSyntaxOn
1868 \seq_gput_right:Nn
1869 \g_@@_renderers_seq
1870 { sectionBegin }
1871 \prop_gput:Nnn
1872 \g_@@_renderer_arities_prop
1873 { sectionBegin }
1874 { 0 }
1875 \ExplSyntaxOff
1876 \def\markdownRendererSectionEnd{%
1877 \markdownRendererSectionEndPrototype}%
1878 \ExplSyntaxOn
1879 \seq_gput_right:Nn
1880 \g_@@_renderers_seq
1881 { sectionEnd }
1882 \prop_gput:Nnn
1883 \g_@@_renderer_arities_prop
1884 { sectionEnd }
1885 { 0 }
1886 \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.

```

1887 \def\markdownRendererReplacementCharacter{%
1888 \markdownRendererReplacementCharacterPrototype}%
1889 \ExplSyntaxOn
1890 \seq_gput_right:Nn
1891 \g_@@_renderers_seq
1892 { replacementCharacter }

```

```

1893 \prop_gput:Nnn
1894 \g_@@_renderer_arities_prop
1895 { replacementCharacter }
1896 { 0 }
1897 \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`.

```

1898 \def\markdownRendererLeftBrace{%
1899 \markdownRendererLeftBracePrototype}%
1900 \ExplSyntaxOn
1901 \seq_gput_right:Nn
1902 \g_@@_renderers_seq
1903 { leftBrace }
1904 \prop_gput:Nnn
1905 \g_@@_renderer_arities_prop
1906 { leftBrace }
1907 { 0 }
1908 \ExplSyntaxOff
1909 \def\markdownRendererRightBrace{%
1910 \markdownRendererRightBracePrototype}%
1911 \ExplSyntaxOn
1912 \seq_gput_right:Nn
1913 \g_@@_renderers_seq
1914 { rightBrace }
1915 \prop_gput:Nnn
1916 \g_@@_renderer_arities_prop
1917 { rightBrace }
1918 { 0 }
1919 \ExplSyntaxOff
1920 \def\markdownRendererDollarSign{%
1921 \markdownRendererDollarSignPrototype}%
1922 \ExplSyntaxOn
1923 \seq_gput_right:Nn
1924 \g_@@_renderers_seq
1925 { dollarSign }
1926 \prop_gput:Nnn
1927 \g_@@_renderer_arities_prop
1928 { dollarSign }
1929 { 0 }
1930 \ExplSyntaxOff
1931 \def\markdownRendererPercentSign{%
1932 \markdownRendererPercentSignPrototype}%
1933 \ExplSyntaxOn
1934 \seq_gput_right:Nn

```

```

1935 \g_@@_renderers_seq
1936 { percentSign }
1937 \prop_gput:Nnn
1938 \g_@@_renderer_arities_prop
1939 { percentSign }
1940 { 0 }
1941 \ExplSyntaxOff
1942 \def\markdownRendererAmpersand{%
1943 \markdownRendererAmpersandPrototype}%
1944 \ExplSyntaxOn
1945 \seq_gput_right:Nn
1946 \g_@@_renderers_seq
1947 { ampersand }
1948 \prop_gput:Nnn
1949 \g_@@_renderer_arities_prop
1950 { ampersand }
1951 { 0 }
1952 \ExplSyntaxOff
1953 \def\markdownRendererUnderscore{%
1954 \markdownRendererUnderscorePrototype}%
1955 \ExplSyntaxOn
1956 \seq_gput_right:Nn
1957 \g_@@_renderers_seq
1958 { underscore }
1959 \prop_gput:Nnn
1960 \g_@@_renderer_arities_prop
1961 { underscore }
1962 { 0 }
1963 \ExplSyntaxOff
1964 \def\markdownRendererHash{%
1965 \markdownRendererHashPrototype}%
1966 \ExplSyntaxOn
1967 \seq_gput_right:Nn
1968 \g_@@_renderers_seq
1969 { hash }
1970 \prop_gput:Nnn
1971 \g_@@_renderer_arities_prop
1972 { hash }
1973 { 0 }
1974 \ExplSyntaxOff
1975 \def\markdownRendererCircumflex{%
1976 \markdownRendererCircumflexPrototype}%
1977 \ExplSyntaxOn
1978 \seq_gput_right:Nn
1979 \g_@@_renderers_seq
1980 { circumflex }
1981 \prop_gput:Nnn

```

```

1982 \g_@@_renderer_arities_prop
1983 { circumflex }
1984 { 0 }
1985 \ExplSyntaxOff
1986 \def\markdownRendererBackslash{%
1987 \markdownRendererBackslashPrototype}%
1988 \ExplSyntaxOn
1989 \seq_gput_right:Nn
1990 \g_@@_renderers_seq
1991 { backslash }
1992 \prop_gput:Nnn
1993 \g_@@_renderer_arities_prop
1994 { backslash }
1995 { 0 }
1996 \ExplSyntaxOff
1997 \def\markdownRendererTilde{%
1998 \markdownRendererTildePrototype}%
1999 \ExplSyntaxOn
2000 \seq_gput_right:Nn
2001 \g_@@_renderers_seq
2002 { tilde }
2003 \prop_gput:Nnn
2004 \g_@@_renderer_arities_prop
2005 { tilde }
2006 { 0 }
2007 \ExplSyntaxOff
2008 \def\markdownRendererPipe{%
2009 \markdownRendererPipePrototype}%
2010 \ExplSyntaxOn
2011 \seq_gput_right:Nn
2012 \g_@@_renderers_seq
2013 { pipe }
2014 \prop_gput:Nnn
2015 \g_@@_renderer_arities_prop
2016 { pipe }
2017 { 0 }
2018 \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.

```

2019 \def\markdownRendererStrikeThrough{%
2020 \markdownRendererStrikeThroughPrototype}%
2021 \ExplSyntaxOn
2022 \seq_gput_right:Nn

```

```

2023 \g_@@_renderers_seq
2024 { strikeThrough }
2025 \prop_gput:Nnn
2026 \g_@@_renderer_arities_prop
2027 { strikeThrough }
2028 { 1 }
2029 \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.

```

2030 \def\markdownRendererSubscript{%
2031 \markdownRendererSubscriptPrototype}%
2032 \ExplSyntaxOn
2033 \seq_gput_right:Nn
2034 \g_@@_renderers_seq
2035 { subscript }
2036 \prop_gput:Nnn
2037 \g_@@_renderer_arities_prop
2038 { subscript }
2039 { 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.

```

2040 \def\markdownRendererSuperscript{%
2041 \markdownRendererSuperscriptPrototype}%
2042 \ExplSyntaxOn
2043 \seq_gput_right:Nn
2044 \g_@@_renderers_seq
2045 { superscript }
2046 \prop_gput:Nnn
2047 \g_@@_renderer_arities_prop
2048 { superscript }
2049 { 1 }
2050 \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

$\langle alignment \rangle$  repeated  $\langle number\ of\ columns \rangle$  times, and  $\langle alignment \rangle$  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.

```

2051 \def\markdownRendererTable{%
2052 \markdownRendererTablePrototype}%
2053 \ExplSyntaxOn
2054 \seq_gput_right:Nn
2055 \g_@@_renderers_seq
2056 { table }
2057 \prop_gput:Nnn
2058 \g_@@_renderer_arities_prop
2059 { table }
2060 { 3 }
2061 \ExplSyntaxOff

```

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

```

2062 \def\markdownRendererInlineMath{%
2063 \markdownRendererInlineMathPrototype}%
2064 \ExplSyntaxOn
2065 \seq_gput_right:Nn
2066 \g_@@_renderers_seq
2067 { inlineMath }
2068 \prop_gput:Nnn
2069 \g_@@_renderer_arities_prop
2070 { inlineMath }
2071 { 1 }
2072 \ExplSyntaxOff
2073 \def\markdownRendererDisplayMath{%
2074 \markdownRendererDisplayMathPrototype}%
2075 \ExplSyntaxOn
2076 \seq_gput_right:Nn
2077 \g_@@_renderers_seq
2078 { displayMath }
2079 \prop_gput:Nnn
2080 \g_@@_renderer_arities_prop
2081 { displayMath }

```



```

2082 { 1 }
2083 \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.

```

2084 \def\markdownRendererTextCite{%
2085 \markdownRendererTextCitePrototype}%
2086 \ExplSyntaxOn
2087 \seq_gput_right:Nn
2088 \g_@@_renderers_seq
2089 { textCite }
2090 \prop_gput:Nnn
2091 \g_@@_renderer_arities_prop
2092 { textCite }
2093 { 1 }
2094 \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.

```

2095 \ExplSyntaxOn
2096 \cs_new:Npn
2097 \markdownRendererThematicBreak
2098 {
2099 \cs_if_exist:NTF
2100 \markdownRendererHorizontalRule
2101 {
2102 \markdownWarning
2103 {
2104 Horizontal~rule~renderer~has~been~deprecated,~
2105 to~be~removed~in~Markdown~3.0.0
2106 }
2107 \markdownRendererHorizontalRule
2108 }
2109 {
2110 \cs_if_exist:NTF
2111 \markdownRendererHorizontalRulePrototype
2112 {
2113 \markdownWarning
2114 {
2115 Horizontal~rule~renderer~prototype~has~been~deprecated,~
2116 to~be~removed~in~Markdown~3.0.0

```

```

2117 }
2118 \markdownRendererHorizontalRulePrototype
2119 }
2120 {
2121 \markdownRendererThematicBreakPrototype
2122 }
2123 }
2124 }
2125 \seq_gput_right:Nn
2126 \g_@@_renderers_seq
2127 { horizontalRule }
2128 \prop_gput:Nnn
2129 \g_@@_renderer_arities_prop
2130 { horizontalRule }
2131 { 0 }
2132 \seq_gput_right:Nn
2133 \g_@@_renderers_seq
2134 { thematicBreak }
2135 \prop_gput:Nnn
2136 \g_@@_renderer_arities_prop
2137 { thematicBreak }
2138 { 0 }
2139 \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.

```

2140 \def\markdownRendererTickedBox{%
2141 \markdownRendererTickedBoxPrototype}%
2142 \ExplSyntaxOn
2143 \seq_gput_right:Nn
2144 \g_@@_renderers_seq
2145 { tickedBox }
2146 \prop_gput:Nnn
2147 \g_@@_renderer_arities_prop
2148 { tickedBox }
2149 { 0 }
2150 \ExplSyntaxOff
2151 \def\markdownRendererHalfTickedBox{%
2152 \markdownRendererHalfTickedBoxPrototype}%
2153 \ExplSyntaxOn
2154 \seq_gput_right:Nn
2155 \g_@@_renderers_seq

```

```

2156 { halfTickedBox }
2157 \prop_gput:Nnn
2158 \g_@@_renderer_arities_prop
2159 { halfTickedBox }
2160 { 0 }
2161 \ExplSyntaxOff
2162 \def\markdownRendererUntickedBox{%
2163 \markdownRendererUntickedBoxPrototype}%
2164 \ExplSyntaxOn
2165 \seq_gput_right:Nn
2166 \g_@@_renderers_seq
2167 { untickedBox }
2168 \prop_gput:Nnn
2169 \g_@@_renderer_arities_prop
2170 { untickedBox }
2171 { 0 }
2172 \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.

```

2173 \def\markdownRendererJekyllDataBegin{%
2174 \markdownRendererJekyllDataBeginPrototype}%
2175 \ExplSyntaxOn
2176 \seq_gput_right:Nn
2177 \g_@@_renderers_seq
2178 { jekyllDataBegin }
2179 \prop_gput:Nnn
2180 \g_@@_renderer_arities_prop
2181 { jekyllDataBegin }
2182 { 0 }
2183 \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.

```

2184 \def\markdownRendererJekyllDataEnd{%
2185 \markdownRendererJekyllDataEndPrototype}%
2186 \ExplSyntaxOn
2187 \seq_gput_right:Nn
2188 \g_@@_renderers_seq
2189 { jekyllDataEnd }
2190 \prop_gput:Nnn
2191 \g_@@_renderer_arities_prop
2192 { jekyllDataEnd }
2193 { 0 }

```

2194 \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.

```
2195 \def\markdownRendererJekyllDataMappingBegin{%
2196 \markdownRendererJekyllDataMappingBeginPrototype}%
2197 \ExplSyntaxOn
2198 \seq_gput_right:Nn
2199 \g_@@_renderers_seq
2200 { jekyllDataMappingBegin }
2201 \prop_gput:Nnn
2202 \g_@@_renderer_arities_prop
2203 { jekyllDataMappingBegin }
2204 { 2 }
2205 \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.

```
2206 \def\markdownRendererJekyllDataMappingEnd{%
2207 \markdownRendererJekyllDataMappingEndPrototype}%
2208 \ExplSyntaxOn
2209 \seq_gput_right:Nn
2210 \g_@@_renderers_seq
2211 { jekyllDataMappingEnd }
2212 \prop_gput:Nnn
2213 \g_@@_renderer_arities_prop
2214 { jekyllDataMappingEnd }
2215 { 0 }
2216 \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.

```
2217 \def\markdownRendererJekyllDataSequenceBegin{%
2218 \markdownRendererJekyllDataSequenceBeginPrototype}%
2219 \ExplSyntaxOn
2220 \seq_gput_right:Nn
2221 \g_@@_renderers_seq
2222 { jekyllDataSequenceBegin }
2223 \prop_gput:Nnn
```

```

2224 \g_@@_renderer_arities_prop
2225 { jekyllDataSequenceBegin }
2226 { 2 }
2227 \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.

```

2228 \def\markdownRendererJekyllDataSequenceEnd{%
2229 \markdownRendererJekyllDataSequenceEndPrototype}%
2230 \ExplSyntaxOn
2231 \seq_gput_right:Nn
2232 \g_@@_renderers_seq
2233 { jekyllDataSequenceEnd }
2234 \prop_gput:Nnn
2235 \g_@@_renderer_arities_prop
2236 { jekyllDataSequenceEnd }
2237 { 0 }
2238 \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.

```

2239 \def\markdownRendererJekyllDataBoolean{%
2240 \markdownRendererJekyllDataBooleanPrototype}%
2241 \ExplSyntaxOn
2242 \seq_gput_right:Nn
2243 \g_@@_renderers_seq
2244 { jekyllDataBoolean }
2245 \prop_gput:Nnn
2246 \g_@@_renderer_arities_prop
2247 { jekyllDataBoolean }
2248 { 2 }
2249 \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.

```

2250 \def\markdownRendererJekyllDataNumber{%
2251 \markdownRendererJekyllDataNumberPrototype}%
2252 \ExplSyntaxOn
2253 \seq_gput_right:Nn

```

```

2254 \g_@@_renderers_seq
2255 { jekyllDataNumber }
2256 \prop_gput:Nnn
2257 \g_@@_renderer_arities_prop
2258 { jekyllDataNumber }
2259 { 2 }
2260 \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.

```

2261 \def\markdownRendererJekyllDataString{%
2262 \markdownRendererJekyllDataStringPrototype}%
2263 \ExplSyntaxOn
2264 \seq_gput_right:Nn
2265 \g_@@_renderers_seq
2266 { jekyllDataString }
2267 \prop_gput:Nnn
2268 \g_@@_renderer_arities_prop
2269 { jekyllDataString }
2270 { 2 }
2271 \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.

```

2272 \def\markdownRendererJekyllDataEmpty{%
2273 \markdownRendererJekyllDataEmptyPrototype}%
2274 \ExplSyntaxOn
2275 \seq_gput_right:Nn
2276 \g_@@_renderers_seq
2277 { jekyllDataEmpty }
2278 \prop_gput:Nnn
2279 \g_@@_renderer_arities_prop
2280 { jekyllDataEmpty }
2281 { 1 }
2282 \ExplSyntaxOff

```

## 2.2.4 Token Renderer Prototypes

### 2.2.4.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes

for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the  $\LaTeX$ 3 kernel.

```
2283 \ExplSyntaxOn
2284 \keys_define:nn
2285 { markdown/jekyllData }
2286 { }
2287 \ExplSyntaxOff
```

The following  $\TeX$  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  $\LaTeX$  and  $\ConTeXt$  implementations (see sections 3.3 and 3.4).

```
2288 \ExplSyntaxOn
2289 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
2290 {
2291 \seq_map_function:NN
2292 \g_@@_renderers_seq
2293 \@@_plaintex_define_renderer_prototype:n
2294 \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
2295 \let\markdownRendererBlockHtmlCommentBegin=\iffalse
2296 \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
2297 \let\markdownRendererBlockHtmlCommentEnd=\fi
```

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

```
2298 \cs_undefine:N \markdownRendererFootnote
2299 \cs_undefine:N \markdownRendererFootnotePrototype
```

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

```
2300 \cs_undefine:N \markdownRendererHorizontalRule
2301 \cs_undefine:N \markdownRendererHorizontalRulePrototype
2302 }
2303 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
2304 {
2305 \@@_renderer_prototype_tl_to_csname:nN
2306 { #1 }
2307 \l_tmpa_tl
2308 \prop_get:NnN
2309 \g_@@_renderer_arities_prop
2310 { #1 }
2311 \l_tmpb_tl
2312 \@@_plaintex_define_renderer_prototype:cV
2313 { \l_tmpa_tl }
2314 \l_tmpb_tl
2315 }
```

```

2316 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
2317 {
2318 \tl_set:Nn
2319 \l_tmpa_tl
2320 { \str_uppercase:n { #1 } }
2321 \tl_set:Nx
2322 #2
2323 {
2324 markdownRenderer
2325 \tl_head:f { \l_tmpa_tl }
2326 \tl_tail:n { #1 }
2327 Prototype
2328 }
2329 }
2330 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
2331 {
2332 \cs_generate_from_arg_count:NNnn
2333 #1
2334 \cs_set:Npn
2335 { #2 }
2336 { }
2337 }
2338 \cs_generate_variant:Nn
2339 \@@_plaintex_define_renderer_prototype:Nn
2340 { cV }
2341 \@@_plaintex_define_renderer_prototypes:
2342 \ExplSyntaxOff

```

### 2.2.5 Logging Facilities

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

### 2.2.6 Miscellanea

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

```

2343 \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.

```
2344 \let\markdownReadAndConvert\relax
2345 \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*.

```
2346 \catcode`\|=0\catcode`\=12%
2347 |gdef|markdownBegin{%
2348 |markdownReadAndConvert{\markdownEnd}%
2349 {|\markdownEnd}}%
2350 |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 T<sub>E</sub>X 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 T<sub>E</sub>X 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.

```
2351 \ExplSyntaxOn
2352 \cs_if_exist:NF
2353 \markdownMode
2354 {
2355 \file_if_exist:nTF
2356 { lt3luabridge.tex }
2357 {
2358 \cs_new:Npn
```

```

2359 \markdownMode
2360 { 3 }
2361 }
2362 {
2363 \cs_if_exist:NTF
2364 \directlua
2365 {
2366 \cs_new:Npn
2367 \markdownMode
2368 { 2 }
2369 }
2370 {
2371 \cs_new:Npn
2372 \markdownMode
2373 { 0 }
2374 }
2375 }
2376 }
2377 \ExplSyntaxOff

```

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

```

2378 \def\markdownLuaRegisterIBCallback#1{\relax}%
2379 \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.

```

2380 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2381 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2382 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
2383 \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  $\LaTeX$  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  $\LaTeX 2_\epsilon$  parses package options.

### 2.3.1 Typesetting Markdown

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

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

```
2384 \newenvironment{markdown}\relax\relax
2385 \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`  $\LaTeX$  environment (and likewise for the starred version).

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

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

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

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  $\TeX$ . Unlike the `\markdownInput` macro provided by the plain  $\TeX$  interface, this macro also accepts  $\LaTeX$  interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

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

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

### 2.3.2 Options

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

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

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

```
2386 \ExplSyntaxOn
2387 \cs_new:Nn
2388 \@@_setup:n
2389 {
2390 \keys_set:nn
2391 { markdown/latex-options }
2392 { #1 }
2393 }
2394 \let\markdownSetup=\@@_setup:n
2395 \ExplSyntaxOff
```

**2.3.2.1  $\LaTeX$  snippets** We may also set up  $\LaTeX$  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:

```
2396 \ExplSyntaxOn
2397 \cs_new:Nn
2398 \@@_latex_setup_snippet:nn
2399 {
2400 \markdownIfSnippetExists
```

```

2401 { #1 }
2402 {
2403 \markdownWarning
2404 {Redefined~snippet~\markdownLaTeXThemeName#1}
2405 \csname markdownLaTeXSetupSnippet%
2406 \markdownLaTeXThemeName#1\endcsname={#2}
2407 }
2408 {
2409 \newtoks\next
2410 \next={#2}
2411 \expandafter\let\csname markdownLaTeXSetupSnippet%
2412 \markdownLaTeXThemeName#1\endcsname=\next
2413 }
2414 }
2415 \let\markdownSetupSnippet=\@@_latex_setup_snippet:nn
2416 \ExplSyntaxOff

```

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

```

2417 \newcommand\markdownIfSnippetExists[3]{%
2418 \ifundefined
2419 {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
2420 {#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.

```

2421 \ExplSyntaxOn
2422 \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.

```

2423 \prop_new:N \g_@@_latex_option_types_prop
2424 \prop_new:N \g_@@_default_latex_options_prop
2425 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
2426 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
2427 \cs_new:Nn
2428 \@@_add_latex_option:nnn
2429 {
2430 \@@_add_option:Vnnn
2431 \c_@@_option_layer_latex_tl
2432 { #1 }
2433 { #2 }
2434 { #3 }
2435 }

```

**2.3.2.2 No default token renderer prototypes** Default token renderer prototypes require  $\LaTeX$  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  $\TeX$  implementation (see Section 3.2.2) and prevent the soft  $\LaTeX$  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}
```

```
2436 \@_add_latex_option:nnn
2437 { plain }
2438 { boolean }
2439 { false }
2440 \ExplSyntaxOff
```

**2.3.2.3  $\LaTeX$  themes** User-defined  $\LaTeX$  themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to  $\LaTeX$  packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The  $\LaTeX$  option `import=<theme name>` loads a  $\LaTeX$  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  $\LaTeX$  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  $\LaTeX$  document class or for a single  $\LaTeX$  package. The preferred format of a theme name is `<theme author>/<target  $\LaTeX$  document class or package>/<private naming scheme>`, where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

Theme names are munged, because  $\LaTeX$  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  $\LaTeX$  package named `markdownthemewitiko_beamer_MU.sty`.

If the  $\LaTeX$  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  $\LaTeX$  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`  $\LaTeX$  package, and finally the `markdownthemewitiko_dot.sty`  $\LaTeX$  package:

```

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

2441 \newif\ifmarkdownLaTeXLoaded
2442 \markdownLaTeXLoadedfalse
2443 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
2444 \ExplSyntaxOn
2445 \tl_new:N \markdownLaTeXThemePackageName
2446 \cs_new:Nn
2447 \@@_set_latex_theme:n
2448 {
2449 \str_if_in:nnF
2450 { #1 }
2451 { / }
2452 {
2453 \markdownError
2454 { Won't-load-theme-with-unqualified-name-#1 }
2455 { Theme-names-must-contain-at-least-one-forward-slash }
2456 }
2457 \str_if_in:nnT
2458 { #1 }
2459 { _ }
2460 {
2461 \markdownError
2462 { Won't-load-theme-with-an-underscore-in-its-name-#1 }
2463 { Theme-names-must-not-contain-underscores-in-their-names }
2464 }
2465 \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
2466 \str_replace_all:Nnn
2467 \markdownLaTeXThemePackageName
2468 { / }
2469 { _ }
2470 \edef\markdownLaTeXThemePackageName{
2471 markdowntheme\markdownLaTeXThemePackageName}
2472 \expandafter\markdownLaTeXThemeLoad\expandafter{
2473 \markdownLaTeXThemePackageName}{#1/}
2474 }

```

```

2475 \keys_define:nn
2476 { markdown/latex-options }
2477 {
2478 import .code:n = {
2479 \tl_set:Nn
2480 \l_tmpa_tl
2481 { #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.

```

2482 \tl_replace_all:NnV
2483 \l_tmpa_tl
2484 { / }
2485 \c_backslash_str
2486 \keys_set:nV
2487 { markdown/latex-options/import }
2488 \l_tmpa_tl
2489 },
2490 }
2491 \cs_generate_variant:Nn
2492 \tl_replace_all:Nnn
2493 { NnV }

```

The  $\LaTeX$  option `theme` has been deprecated and will be removed in Markdown 3.0.0.

```

2494 \keys_define:nn
2495 { markdown/latex-options }
2496 {
2497 theme .code:n = { \@@_set_latex_theme:n { #1 } },
2498 }
2499 \ExplSyntaxOff

```

The  $\LaTeX$  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  $\langle theme\ name \rangle/$  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  $\LaTeX$ , themes may not be loaded after the beginning of a  $\LaTeX$  document.

```

2500 \ExplSyntaxOn
2501 \@onlypreamble
2502 \@@_set_latex_theme:n
2503 \ExplSyntaxOff

```

Example themes provided with the Markdown package include:



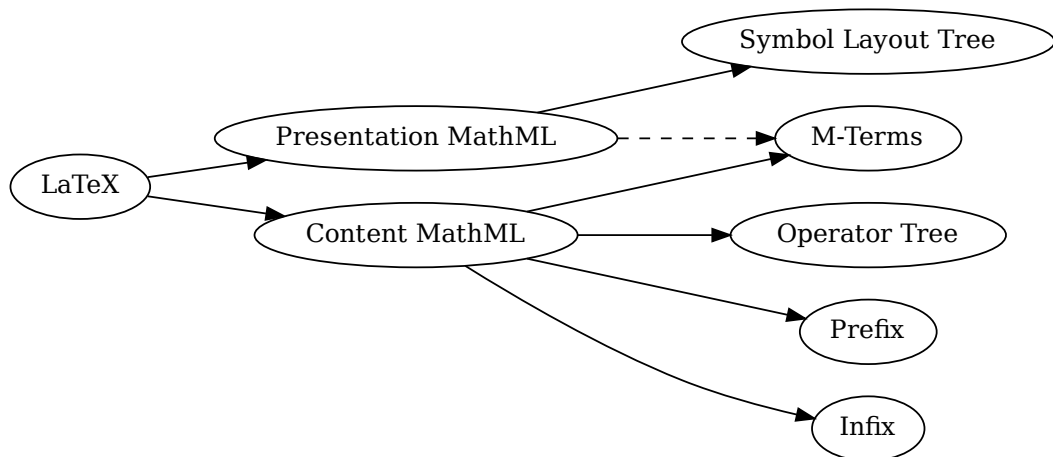
**witiko/dot** A theme that typesets fenced code blocks with the `dot ...` infostring as images of directed graphs rendered by the Graphviz tools. The right tail of the infostring is used as the image title.

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



**Figure 4: Various formats of mathematical formulae**

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

2504 `\ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%`

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

```

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

```

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

2505 `\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{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

```

\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”.

```
2506 \ProvidesPackage{markdownthemewitiko_tilde}[2021/03/22]%
```

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

**2.3.2.4  $\LaTeX$  snippets** The  $\LaTeX$  option with key `snippet` invokes a snippet named  $\langle value \rangle$ :

```

2507 \ExplSyntaxOn
2508 \keys_define:nn
2509 { markdown/latex-options }

```

```

2510 {
2511 snippet .code:n = {
2512 \markdownIfSnippetExists{#1}
2513 {
2514 \expandafter\markdownSetup\expandafter{
2515 \the\csname markdownLaTeXSetupSnippet
2516 \markdownLaTeXThemeName#1\endcsname}
2517 }{
2518 \markdownError
2519 {Can't~invoke~setup~snippet~#1}
2520 {The~setup~snippet~is~undefined}
2521 }
2522 }
2523 }
2524 \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 could import the `jdoe/lists` theme and use the qualified name `jdoe/lists/romanNumerals` to invoke the snippet:

```
\markdownSetup{import=jdoo/lists}
\begin{markdown*}{snippet=jdoo/lists/romanNumerals}
```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

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

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

```
\markdownSetup{
 import = {
 jdoo/lists = romanNumerals,
 },
}
\begin{markdown*}{snippet=romanNumerals}
```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

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

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

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

The following ordered list will be preceded by roman numerals:

```

3. tres
4. quattuor

\end{markdown*}

```

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

```

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

```

```

2525 \ExplSyntaxOn
2526 \tl_new:N
2527 \l_@@_latex_import_current_theme_tl
2528 \keys_define:nn
2529 { markdown/latex-options/import }
2530 {

```

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

```

2531 unknown .default:n = {},
2532 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.

```

2533 \tl_set_eq:NN
2534 \l_@@_latex_import_current_theme_tl
2535 \l_keys_key_str

```

```

2536 \tl_replace_all:NVn
2537 \l_@@_latex_import_current_theme_tl
2538 \c_backslash_str
2539 { / }

```

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

```

2540 \@@_set_latex_theme:V
2541 \l_@@_latex_import_current_theme_tl

```

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

```

2542 \clist_map_inline:nn
2543 { #1 }
2544 {
2545 \regex_extract_once:nnNTF
2546 { ^(.*)\s+as\s+(.*)$ }
2547 { ##1 }
2548 \l_tmpa_seq
2549 {
2550 \seq_pop:NN
2551 \l_tmpa_seq
2552 \l_tmpa_tl
2553 \seq_pop:NN
2554 \l_tmpa_seq
2555 \l_tmpa_tl
2556 \seq_pop:NN
2557 \l_tmpa_seq
2558 \l_tmpb_tl
2559 }
2560 {
2561 \tl_set:Nn
2562 \l_tmpa_tl
2563 { ##1 }
2564 \tl_set:Nn
2565 \l_tmpb_tl
2566 { ##1 }
2567 }
2568 \tl_put_left:Nn
2569 \l_tmpa_tl
2570 { / }
2571 \tl_put_left:NV
2572 \l_tmpa_tl
2573 \l_@@_latex_import_current_theme_tl
2574 \@@_latex_setup_snippet:Vx
2575 \l_tmpb_tl
2576 { snippet = { \l_tmpa_tl } }
2577 }
2578 },
2579 }

```

```

2580 \cs_generate_variant:Nn
2581 \tl_replace_all:Nnn
2582 { NVn }
2583 \cs_generate_variant:Nn
2584 \@@_set_latex_theme:n
2585 { V }
2586 \cs_generate_variant:Nn
2587 \@@_latex_setup_snippet:nn
2588 { Vx }
2589 \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.

```

2590 \ExplSyntaxOn
2591 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
2592 {
2593 \seq_map_inline:Nn
2594 \g_@@_latex_options_seq
2595 {
2596 \@@_plain_tex_define_option_command:n
2597 { ##1 }
2598 }

```

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.

```

2599 \seq_map_inline:Nn
2600 \g_@@_option_layers_seq
2601 {
2602 \seq_map_inline:cn
2603 { g_@@_ ##1 _options_seq }
2604 {

```

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.

```

2605 \@@_with_various_cases:nn
2606 { #####1 }
2607 {
2608 \@@_latex_define_option_keyval:nnn
2609 { ##1 }
2610 { #####1 }
2611 { #####1##1 }
2612 }
2613 }
2614 }
2615 }

```



```

2616 \cs_new:Nn \@@_latex_define_option_keyval:nnn
2617 {
2618 \prop_get:cnN
2619 { g_@@_ #1 _option_types_prop }
2620 { #2 }
2621 \l_tmpa_tl
2622 \keys_define:nn
2623 { markdown/latex-options }
2624 {
2625 #3 .code:n = {
2626 \@@_set_option_value:nn
2627 { #2 }
2628 { ##1 }
2629 },
2630 }
2631 \str_if_eq:VVT
2632 \l_tmpa_tl
2633 \c_@@_option_type_boolean_tl
2634 {
2635 \keys_define:nn
2636 { markdown/latex-options }
2637 {
2638 #3 .default:n = { true },
2639 }
2640 }

```

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.

```

2641 \str_if_eq:VVT
2642 \l_tmpa_tl
2643 \c_@@_option_type_clist_tl
2644 {
2645 \tl_set:Nn
2646 \l_tmpa_tl
2647 { #3 }
2648 \tl_reverse:N
2649 \l_tmpa_tl
2650 \str_if_eq:enF
2651 {
2652 \tl_head:V
2653 \l_tmpa_tl
2654 }
2655 { s }
2656 {

```

```

2657 \msg_error:nnn
2658 { markdown }
2659 { malformed-name-for-clist-option }
2660 { #3 }
2661 }
2662 \tl_set:Nx
2663 \l_tmpa_tl
2664 {
2665 \tl_tail:V
2666 \l_tmpa_tl
2667 }
2668 \tl_reverse:N
2669 \l_tmpa_tl
2670 \tl_put_right:Nn
2671 \l_tmpa_tl
2672 {
2673 .code:n = {
2674 \@@_get_option_value:nN
2675 { #2 }
2676 \l_tmpa_tl
2677 \clist_set:NV
2678 \l_tmpa_clist
2679 { \l_tmpa_tl, { ##1 } }
2680 \@@_set_option_value:nV
2681 { #2 }
2682 \l_tmpa_clist
2683 }
2684 }
2685 \keys_define:nV
2686 { markdown/latex-options }
2687 \l_tmpa_tl
2688 }
2689 }
2690 \cs_generate_variant:Nn
2691 \clist_set:Nn
2692 { NV }
2693 \cs_generate_variant:Nn
2694 \keys_define:nn
2695 { nV }
2696 \cs_generate_variant:Nn
2697 \@@_set_option_value:nn
2698 { nV }
2699 \prg_generate_conditional_variant:Nnn
2700 \str_if_eq:nn
2701 { en }
2702 { F }
2703 \msg_new:nnn

```

```

2704 { markdown }
2705 { malformed-name-for-clist-option }
2706 {
2707 Clist-option-name~#1~does~not~end~with~-s.
2708 }
2709 \@@_latex_define_option_commands_and_keyvals:
2710 \ExplSyntaxOff

```

The `finalizeCache` and `frozenCache` plain TeX options are exposed through L<sup>A</sup>T<sub>E</sub>X options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozencache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document class options. By passing `finalizcache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

```

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

```

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

```

2711 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
2712 \DeclareOption{frozencache}{\markdownSetup{frozenCache}}

```

The following example L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X interface recognizes an option with the `renderers` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer macros exposed by the plain TeX interface (see Section 2.2.3) and the values are new definitions of these token renderers.

```

2713 \ExplSyntaxOn
2714 \cs_new:Nn \@@_latex_define_renderers:

```

```

2715 {
2716 \seq_map_function:NN
2717 \g_@@_renderers_seq
2718 \@@_latex_define_renderer:n
2719 }
2720 \cs_new:Nn \@@_latex_define_renderer:n
2721 {
2722 \@@_renderer_tl_to_csname:nN
2723 { #1 }
2724 \l_tmpa_tl
2725 \prop_get:NnN
2726 \g_@@_renderer_arities_prop
2727 { #1 }
2728 \l_tmpb_tl
2729 \@@_latex_define_renderer:ncV
2730 { #1 }
2731 { \l_tmpa_tl }
2732 \l_tmpb_tl
2733 }
2734 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2735 {
2736 \tl_set:Nn
2737 \l_tmpa_tl
2738 { \str_uppercase:n { #1 } }
2739 \tl_set:Nx
2740 #2
2741 {
2742 markdownRenderer
2743 \tl_head:f { \l_tmpa_tl }
2744 \tl_tail:n { #1 }
2745 }
2746 }
2747 \cs_new:Nn \@@_latex_define_renderer:nNn
2748 {
2749 \@@_with_various_cases:nn
2750 { #1 }
2751 {
2752 \keys_define:nn
2753 { markdown/latex-options/renderers }
2754 {
2755 ##1 .code:n = {
2756 \cs_generate_from_arg_count:NNnn
2757 #2
2758 \cs_set:Npn
2759 { #3 }
2760 { #####1 }
2761 },

```

```

2762 }
2763 }
2764 }
2765 \cs_generate_variant:Nn
2766 \@@_latex_define_renderer:nNn
2767 { ncV }
2768 \ExplSyntaxOff

```

The following example L<sup>A</sup>T<sub>E</sub>X 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`.
 }
}

```

In addition to exact token renderer names, we also support wildcards that match multiple token renderer names.

```

2769 \ExplSyntaxOn
2770 \tl_new:N
2771 \l_@@_renderer_definition_tl
2772 \keys_define:nn
2773 { markdown/latex-options/renderers }
2774 {
2775 unknown .code:n = {
2776 \regex_match:nVTF
2777 { * }
2778 \l_keys_key_str
2779 {
2780 \tl_set:Nn
2781 \l_@@_renderer_definition_tl
2782 { #1 }
2783 \tl_set:NV
2784 \l_tmpa_tl
2785 \l_keys_key_str
2786 \regex_replace_all:nnN
2787 { * }
2788 { .* }
2789 \l_tmpa_tl
2790 \regex_set:NV
2791 \l_tmpa_regex
2792 \l_tmpa_tl
2793 \int_zero:N

```

```

2794 \l_tmpa_int
2795 \seq_map_inline:Nn
2796 \g_@@_renderers_seq
2797 {
2798 \@@_with_various_cases:nn
2799 { ##1 }
2800 {
2801 \regex_match:NnT
2802 \l_tmpa_regex
2803 { ####1 }
2804 {
2805 \@@_renderer_tl_to_csname:nN
2806 { ##1 }
2807 \l_tmpa_tl
2808 \prop_get:NnN
2809 \g_@@_renderer_arities_prop
2810 { ##1 }
2811 \l_tmpb_tl
2812 \cs_generate_from_arg_count:cNVV
2813 { \l_tmpa_tl }
2814 \cs_set:Npn
2815 \l_tmpb_tl
2816 \l_@@_renderer_definition_tl
2817 \int_incr:N
2818 \l_tmpa_int
2819 \@@_with_various_cases_break:
2820 }
2821 }
2822 }
2823 \int_compare:nNnT
2824 { \l_tmpa_int } = { 0 }
2825 {
2826 \msg_error:nnV
2827 { markdown }
2828 { nonmatched-renderer-wildcard }
2829 \l_keys_key_str
2830 }
2831 }
2832 {
2833 \msg_error:nnV
2834 { markdown }
2835 { undefined-renderer }
2836 \l_keys_key_str
2837 }
2838 },
2839 }
2840 \msg_new:nnn

```

```

2841 { markdown }
2842 { undefined-renderer }
2843 {
2844 Renderer~#1~is~undefined.
2845 }
2846 \msg_new:nnn
2847 { markdown }
2848 { nonmatched-renderer-wildcard }
2849 {
2850 Wildcard~#1~matches~no~renderers.
2851 }
2852 \cs_generate_variant:Nn
2853 \regex_set:Nn
2854 { NV }
2855 \cs_generate_variant:Nn
2856 \cs_generate_from_arg_count:NNnn
2857 { cNvV }
2858 \cs_generate_variant:Nn
2859 \msg_error:nnn
2860 { nnV }
2861 \prg_generate_conditional_variant:Nnn
2862 \regex_match:nn
2863 { nV }
2864 { TF }
2865 \ExplSyntaxOff

```

**2.3.2.7 Plain T<sub>E</sub>X Markdown Token Renderer Prototypes** The L<sup>A</sup>T<sub>E</sub>X interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer prototype macros exposed by the plain T<sub>E</sub>X interface (see Section 2.2.4) and the values are new definitions of these token renderer prototypes.

```

2866 \ExplSyntaxOn
2867 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2868 {
2869 \seq_map_function:NN
2870 \g_@@_renderers_seq
2871 \@@_latex_define_renderer_prototype:n
2872 }
2873 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2874 {
2875 \@@_renderer_prototype_tl_to_csname:nN
2876 { #1 }
2877 \l_tmpa_tl
2878 \prop_get:NnN
2879 \g_@@_renderer_arities_prop
2880 { #1 }

```

```

2881 \l_tmpb_tl
2882 \@@_latex_define_renderer_prototype:ncV
2883 { #1 }
2884 { \l_tmpa_tl }
2885 \l_tmpb_tl
2886 }
2887 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2888 {
2889 \@@_with_various_cases:nn
2890 { #1 }
2891 {
2892 \keys_define:nn
2893 { markdown/latex-options/renderer-prototypes }
2894 {
2895 ##1 .code:n = {
2896 \cs_generate_from_arg_count:NNnn
2897 #2
2898 \cs_set:Npn
2899 { #3 }
2900 { #####1 }
2901 },
2902 }
2903 }
2904 }
2905 \cs_generate_variant:Nn
2906 \@@_latex_define_renderer_prototype:nNn
2907 { ncV }
2908 \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`.
 }
}

```

In addition to exact token renderer prototype names, we also support wildcards that match multiple token renderer prototype names.

```

2909 \ExplSyntaxOn
2910 \tl_new:N
2911 \l_@@_renderer_prototype_definition_tl
2912 \keys_define:nn

```



```

2913 { markdown/latex-options/renderer-prototypes }
2914 {
2915 unknown .code:n = {
2916 \regex_match:nVTF
2917 { * }
2918 \l_keys_key_str
2919 {
2920 \tl_set:Nn
2921 \l_@@_renderer_prototype_definition_tl
2922 { #1 }
2923 \tl_set:NV
2924 \l_tmpa_tl
2925 \l_keys_key_str
2926 \regex_replace_all:nnN
2927 { * }
2928 { .* }
2929 \l_tmpa_tl
2930 \regex_set:NV
2931 \l_tmpa_regex
2932 \l_tmpa_tl
2933 \int_zero:N
2934 \l_tmpa_int
2935 \seq_map_inline:Nn
2936 \g_@@_renderers_seq
2937 {
2938 \@@_with_various_cases:nn
2939 { ##1 }
2940 {
2941 \regex_match:NnT
2942 \l_tmpa_regex
2943 { ####1 }
2944 {
2945 \@@_renderer_prototype_tl_to_csname:nN
2946 { ##1 }
2947 \l_tmpa_tl
2948 \prop_get:NnN
2949 \g_@@_renderer_arities_prop
2950 { ##1 }
2951 \l_tmpb_tl
2952 \cs_generate_from_arg_count:cNVV
2953 { \l_tmpa_tl }
2954 \cs_set:Npn
2955 \l_tmpb_tl
2956 \l_@@_renderer_prototype_definition_tl
2957 \int_incr:N
2958 \l_tmpa_int
2959 \@@_with_various_cases_break:

```

```

2960 }
2961 }
2962 }
2963 \int_compare:nNnT
2964 { \l_tmpa_int } = { 0 }
2965 {
2966 \msg_error:nnV
2967 { markdown }
2968 { nonmatched-renderer-prototype-wildcard }
2969 \l_keys_key_str
2970 }
2971 }
2972 {
2973 \msg_error:nnV
2974 { markdown }
2975 { undefined-renderer-prototype }
2976 \l_keys_key_str
2977 }
2978 },
2979 }
2980 \msg_new:nnn
2981 { markdown }
2982 { undefined-renderer-prototype }
2983 {
2984 Renderer~prototype~#1~is~undefined.
2985 }
2986 \msg_new:nnn
2987 { markdown }
2988 { nonmatched-renderer-prototype-wildcard }
2989 {
2990 Wildcard~#1~matches~no~renderer~prototypes.
2991 }
2992 \cs_generate_variant:Nn
2993 \regex_set:Nn
2994 { NV }
2995 \cs_generate_variant:Nn
2996 \cs_generate_from_arg_count:NNnn
2997 { cNVV }
2998 \cs_generate_variant:Nn
2999 \msg_error:nnn
3000 { nnV }
3001 \prg_generate_conditional_variant:Nnn
3002 \regex_match:nn
3003 { nV }
3004 { TF }
3005 \ExplSyntaxOff

```

## 2.4 ConTeXt Interface

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

```
3006 \writestatus{loading}{ConTeXt User Module / markdown}%
3007 \startmodule[markdown]
3008 \unprotect
```

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

```
3009 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
3010 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
3011 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3012 \do#\do\^\do_do\%do\~}%
3013 \input markdown/markdown
```

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

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

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

### 2.4.1 Typesetting Markdown

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

```
3014 \let\startmarkdown\relax
3015 \let\stopmarkdown\relax
3016 \let\inputmarkdown\relax
```

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

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

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

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

```

\startmarkdown
Hello world ...
\stopmarkdown
\stoptext

```

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

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

```

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

```

## 2.4.2 Options

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

Con $\TeX$ t options map directly to the options recognized by the plain  $\TeX$  interface (see Section 2.2.2).

The Con $\TeX$ t 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:

```

3017 \ExplSyntaxOn
3018 \cs_new:Nn
3019 \@@_setup:n
3020 {
3021 \keys_set:nn
3022 { markdown/context-options }
3023 { #1 }
3024 }
3025 \long\def\setupmarkdown[#1]
3026 {
3027 \@@_setup:n
3028 { #1 }
3029 }
3030 \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.

```

3031 \ExplSyntaxOn
3032 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
3033 {
3034 \seq_map_inline:Nn
3035 \g_@@_option_layers_seq
3036 {
3037 \seq_map_inline:cn
3038 { g_@@_ ##1 _options_seq }
3039 {

```

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.

```

3040 \@@_with_various_cases:nn
3041 { #####1 }
3042 {
3043 \@@_context_define_option_keyval:nnn
3044 { ##1 }
3045 { #####1 }
3046 { #####1##1 }
3047 }
3048 }
3049 }
3050 }

```

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

```

3051 \cs_new:Nn \@@_caseless:N
3052 {
3053 \regex_replace_all:nnN
3054 { ([a-z])([A-Z]) }
3055 { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3056 #1
3057 \tl_set:Nx
3058 #1
3059 { #1 }
3060 }
3061 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
3062 \cs_new:Nn \@@_context_define_option_keyval:nnn
3063 {
3064 \prop_get:cnN
3065 { g_@@_ #1 _option_types_prop }
3066 { #2 }
3067 \l_tmpa_tl

```

```

3068 \keys_define:nn
3069 { markdown/context-options }
3070 {
3071 #3 .code:n = {
3072 \tl_set:Nx
3073 \l_tmpa_tl
3074 {
3075 \str_case:nnF
3076 { ##1 }
3077 {
3078 { yes } { true }
3079 { no } { false }
3080 }
3081 { ##1 }
3082 }
3083 \@@_set_option_value:nV
3084 { #2 }
3085 \l_tmpa_tl
3086 },
3087 }
3088 \str_if_eq:VVT
3089 \l_tmpa_tl
3090 \c_@@_option_type_boolean_tl
3091 {
3092 \keys_define:nn
3093 { markdown/context-options }
3094 {
3095 #3 .default:n = { true },
3096 }
3097 }
3098 }
3099 \cs_generate_variant:Nn
3100 \@@_set_option_value:nn
3101 { nV }
3102 \@@_context_define_option_commands_and_keyvals:
3103 \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

ConT<sub>E</sub>Xt layers correct idiosyncrasies of the respective T<sub>E</sub>X formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain T<sub>E</sub>X, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

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

```
3104 local upper, format, length =
3105 string.upper, string.format, string.len
3106 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
3107 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3108 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.

```
3109 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.

```
3110 function util.err(msg, exit_code)
3111 io.stderr:write("markdown.lua: " .. msg .. "\n")
3112 os.exit(exit_code or 1)
3113 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.

```
3114 function util.cache(dir, string, salt, transform, suffix)
3115 local digest = md5.sumhexa(string .. (salt or ""))
3116 local name = util.pathname(dir, digest .. suffix)
3117 local file = io.open(name, "r")
3118 if file == nil then -- If no cache entry exists, then create a new one.
3119 file = assert(io.open(name, "w"),
3120 [[Could not open file]] .. name .. [[for writing]])
3121 local result = string
```

```

3122 if transform ~= nil then
3123 result = transform(result)
3124 end
3125 assert(file:write(result))
3126 assert(file:close())
3127 end
3128 return name
3129 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.

```

3130 function util.cache_verbatim(dir, string)
3131 local name = util.cache(dir, string, nil, nil, ".verbatim")
3132 return name
3133 end

```

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

```

3134 function util.table_copy(t)
3135 local u = { }
3136 for k, v in pairs(t) do u[k] = v end
3137 return setmetatable(u, getmetatable(t))
3138 end

```

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

```

3139 function util.encode_json_string(s)
3140 s = s:gsub([[\\]], [[\\]])
3141 s = s:gsub([[\"]], [[\"]])
3142 return [[\"]] .. s .. [[\"]]
3143 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]

```

3144 function util.lookup_files(f, options)
3145 return kpse.lookup(f, options)
3146 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].

```

3147 function util.expand_tabs_in_line(s, tabstop)
3148 local tab = tabstop or 4
3149 local corr = 0
3150 return (s:gsub(")\t", function(p)
3151 local sp = tab - (p - 1 + corr) % tab
3152 corr = corr - 1 + sp

```



```

3153 return string.rep(" ", sp)
3154 end))
3155 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.

```

3156 function util.walk(t, f)
3157 local typ = type(t)
3158 if typ == "string" then
3159 f(t)
3160 elseif typ == "table" then
3161 local i = 1
3162 local n
3163 n = t[i]
3164 while n do
3165 util.walk(n, f)
3166 i = i + 1
3167 n = t[i]
3168 end
3169 elseif typ == "function" then
3170 local ok, val = pcall(t)
3171 if ok then
3172 util.walk(val, f)
3173 end
3174 else
3175 f(tostring(t))
3176 end
3177 end

```

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

```

3178 function util.flatten(ary)
3179 local new = {}
3180 for _,v in ipairs(ary) do
3181 if type(v) == "table" then
3182 for _,w in ipairs(util.flatten(v)) do
3183 new[#new + 1] = w
3184 end
3185 else
3186 new[#new + 1] = v
3187 end
3188 end
3189 return new
3190 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.

```
3191 function util.rope_to_string(rope)
3192 local buffer = {}
3193 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3194 return table.concat(buffer)
3195 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.

```
3196 function util.rope_last(rope)
3197 if #rope == 0 then
3198 return nil
3199 else
3200 local l = rope[#rope]
3201 if type(l) == "table" then
3202 return util.rope_last(l)
3203 else
3204 return l
3205 end
3206 end
3207 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  $1 \leq i \leq \#ary$ .

```
3208 function util.intersperse(ary, x)
3209 local new = {}
3210 local l = #ary
3211 for i,v in ipairs(ary) do
3212 local n = #new
3213 new[n + 1] = v
3214 if i ~= l then
3215 new[n + 2] = x
3216 end
3217 end
3218 return new
3219 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  $1 \leq i \leq \#ary$ .

```
3220 function util.map(ary, f)
3221 local new = {}
3222 for i,v in ipairs(ary) do
3223 new[i] = f(v)
3224 end
3225 return new
3226 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.

```
3227 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3228 local char_escapes_list = ""
3229 for i,_ in pairs(char_escapes) do
3230 char_escapes_list = char_escapes_list .. i
3231 end
```

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

```
3232 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.

```
3233 if string_escapes then
3234 for k,v in pairs(string_escapes) do
3235 escapable = P(k) / v + escapable
3236 end
3237 end
```

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

```
3238 local escape_string = Cs((escapable + any)^0)
```

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

```
3239 return function(s)
3240 return lpeg.match(escape_string, s)
3241 end
3242 end
```

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

```
3243 function util.pathname(dir, file)
3244 if #dir == 0 then
3245 return file
```

```
3246 else
3247 return dir .. "/" .. file
3248 end
3249 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.

```
3250 local entities = {}
3251
3252 local character_entities = {
3253 ["Tab"] = 9,
3254 ["NewLine"] = 10,
3255 ["excl"] = 33,
3256 ["quot"] = 34,
3257 ["QUOT"] = 34,
3258 ["num"] = 35,
3259 ["dollar"] = 36,
3260 ["percent"] = 37,
3261 ["amp"] = 38,
3262 ["AMP"] = 38,
3263 ["apos"] = 39,
3264 ["lpar"] = 40,
3265 ["rpar"] = 41,
3266 ["ast"] = 42,
3267 ["midast"] = 42,
3268 ["plus"] = 43,
3269 ["comma"] = 44,
3270 ["period"] = 46,
3271 ["sol"] = 47,
3272 ["colon"] = 58,
3273 ["semi"] = 59,
3274 ["lt"] = 60,
3275 ["LT"] = 60,
3276 ["equals"] = 61,
3277 ["gt"] = 62,
3278 ["GT"] = 62,
3279 ["quest"] = 63,
3280 ["commat"] = 64,
3281 ["lsqb"] = 91,
3282 ["lbrack"] = 91,
3283 ["bsol"] = 92,
3284 ["rsqb"] = 93,
3285 ["rbrack"] = 93,
```

3286 ["Hat"] = 94,  
3287 ["lowbar"] = 95,  
3288 ["grave"] = 96,  
3289 ["DiacriticalGrave"] = 96,  
3290 ["lcub"] = 123,  
3291 ["lbrace"] = 123,  
3292 ["verbar"] = 124,  
3293 ["vert"] = 124,  
3294 ["VerticalLine"] = 124,  
3295 ["rcub"] = 125,  
3296 ["rbrace"] = 125,  
3297 ["nbsp"] = 160,  
3298 ["NonBreakingSpace"] = 160,  
3299 ["iexcl"] = 161,  
3300 ["cent"] = 162,  
3301 ["pound"] = 163,  
3302 ["curren"] = 164,  
3303 ["yen"] = 165,  
3304 ["brvbar"] = 166,  
3305 ["sect"] = 167,  
3306 ["Dot"] = 168,  
3307 ["die"] = 168,  
3308 ["DoubleDot"] = 168,  
3309 ["uml"] = 168,  
3310 ["copy"] = 169,  
3311 ["COPY"] = 169,  
3312 ["ordf"] = 170,  
3313 ["laquo"] = 171,  
3314 ["not"] = 172,  
3315 ["shy"] = 173,  
3316 ["reg"] = 174,  
3317 ["circledR"] = 174,  
3318 ["REG"] = 174,  
3319 ["macr"] = 175,  
3320 ["OverBar"] = 175,  
3321 ["strns"] = 175,  
3322 ["deg"] = 176,  
3323 ["plusmn"] = 177,  
3324 ["pm"] = 177,  
3325 ["PlusMinus"] = 177,  
3326 ["sup2"] = 178,  
3327 ["sup3"] = 179,  
3328 ["acute"] = 180,  
3329 ["DiacriticalAcute"] = 180,  
3330 ["micro"] = 181,  
3331 ["para"] = 182,  
3332 ["middot"] = 183,

3333 ["centerdot"] = 183,  
3334 ["CenterDot"] = 183,  
3335 ["cedil"] = 184,  
3336 ["Cedilla"] = 184,  
3337 ["sup1"] = 185,  
3338 ["ordm"] = 186,  
3339 ["raquo"] = 187,  
3340 ["frac14"] = 188,  
3341 ["frac12"] = 189,  
3342 ["half"] = 189,  
3343 ["frac34"] = 190,  
3344 ["iquest"] = 191,  
3345 ["Agrave"] = 192,  
3346 ["Aacute"] = 193,  
3347 ["Acirc"] = 194,  
3348 ["Atilde"] = 195,  
3349 ["Auml"] = 196,  
3350 ["Aring"] = 197,  
3351 ["AElig"] = 198,  
3352 ["Ccedil"] = 199,  
3353 ["Egrave"] = 200,  
3354 ["Eacute"] = 201,  
3355 ["Ecirc"] = 202,  
3356 ["Euml"] = 203,  
3357 ["Igrave"] = 204,  
3358 ["Iacute"] = 205,  
3359 ["Icirc"] = 206,  
3360 ["Iuml"] = 207,  
3361 ["ETH"] = 208,  
3362 ["Ntilde"] = 209,  
3363 ["Ograve"] = 210,  
3364 ["Oacute"] = 211,  
3365 ["Ocirc"] = 212,  
3366 ["Otilde"] = 213,  
3367 ["Ouml"] = 214,  
3368 ["times"] = 215,  
3369 ["Oslash"] = 216,  
3370 ["Ugrave"] = 217,  
3371 ["Uacute"] = 218,  
3372 ["Ucirc"] = 219,  
3373 ["Uuml"] = 220,  
3374 ["Yacute"] = 221,  
3375 ["THORN"] = 222,  
3376 ["szlig"] = 223,  
3377 ["agrave"] = 224,  
3378 ["aacute"] = 225,  
3379 ["acirc"] = 226,

3380 ["atilde"] = 227,  
3381 ["auml"] = 228,  
3382 ["aring"] = 229,  
3383 ["aelig"] = 230,  
3384 ["ccedil"] = 231,  
3385 ["egrave"] = 232,  
3386 ["eacute"] = 233,  
3387 ["ecirc"] = 234,  
3388 ["euml"] = 235,  
3389 ["igrave"] = 236,  
3390 ["iacute"] = 237,  
3391 ["icirc"] = 238,  
3392 ["iuml"] = 239,  
3393 ["eth"] = 240,  
3394 ["ntilde"] = 241,  
3395 ["ograve"] = 242,  
3396 ["oacute"] = 243,  
3397 ["ocirc"] = 244,  
3398 ["otilde"] = 245,  
3399 ["ouml"] = 246,  
3400 ["divide"] = 247,  
3401 ["div"] = 247,  
3402 ["oslash"] = 248,  
3403 ["ugrave"] = 249,  
3404 ["uacute"] = 250,  
3405 ["ucirc"] = 251,  
3406 ["uuml"] = 252,  
3407 ["yacute"] = 253,  
3408 ["thorn"] = 254,  
3409 ["yuml"] = 255,  
3410 ["Amacr"] = 256,  
3411 ["amacr"] = 257,  
3412 ["Abreve"] = 258,  
3413 ["abreve"] = 259,  
3414 ["Aogon"] = 260,  
3415 ["aogon"] = 261,  
3416 ["Cacute"] = 262,  
3417 ["cacute"] = 263,  
3418 ["Ccirc"] = 264,  
3419 ["ccirc"] = 265,  
3420 ["Cdot"] = 266,  
3421 ["cdot"] = 267,  
3422 ["Ccaron"] = 268,  
3423 ["ccaron"] = 269,  
3424 ["Dcaron"] = 270,  
3425 ["dcaron"] = 271,  
3426 ["Dstrok"] = 272,

3427 ["dstrok"] = 273,  
3428 ["Emacr"] = 274,  
3429 ["emacr"] = 275,  
3430 ["Edot"] = 278,  
3431 ["edot"] = 279,  
3432 ["Eogon"] = 280,  
3433 ["eogon"] = 281,  
3434 ["Ecaron"] = 282,  
3435 ["ecaron"] = 283,  
3436 ["Gcirc"] = 284,  
3437 ["gcirc"] = 285,  
3438 ["Gbreve"] = 286,  
3439 ["gbreve"] = 287,  
3440 ["Gdot"] = 288,  
3441 ["gdot"] = 289,  
3442 ["Gcedil"] = 290,  
3443 ["Hcirc"] = 292,  
3444 ["hcirc"] = 293,  
3445 ["Hstrook"] = 294,  
3446 ["hstrook"] = 295,  
3447 ["Itilde"] = 296,  
3448 ["itilde"] = 297,  
3449 ["Imacr"] = 298,  
3450 ["imacr"] = 299,  
3451 ["Iogon"] = 302,  
3452 ["iogon"] = 303,  
3453 ["Idot"] = 304,  
3454 ["imath"] = 305,  
3455 ["inodot"] = 305,  
3456 ["IJlig"] = 306,  
3457 ["ijlig"] = 307,  
3458 ["Jcirc"] = 308,  
3459 ["jcirc"] = 309,  
3460 ["Kcedil"] = 310,  
3461 ["kcedil"] = 311,  
3462 ["kgreen"] = 312,  
3463 ["Lacute"] = 313,  
3464 ["lacute"] = 314,  
3465 ["Lcedil"] = 315,  
3466 ["lcedil"] = 316,  
3467 ["Lcaron"] = 317,  
3468 ["lcaron"] = 318,  
3469 ["Lmidot"] = 319,  
3470 ["lmidot"] = 320,  
3471 ["Lstrook"] = 321,  
3472 ["lstrook"] = 322,  
3473 ["Nacute"] = 323,



3474 ["nacute"] = 324,  
3475 ["Ncedil"] = 325,  
3476 ["ncedil"] = 326,  
3477 ["Ncaron"] = 327,  
3478 ["ncaron"] = 328,  
3479 ["napos"] = 329,  
3480 ["ENG"] = 330,  
3481 ["eng"] = 331,  
3482 ["Omacr"] = 332,  
3483 ["omacr"] = 333,  
3484 ["Odblac"] = 336,  
3485 ["odblac"] = 337,  
3486 ["OElig"] = 338,  
3487 ["oelig"] = 339,  
3488 ["Racute"] = 340,  
3489 ["racute"] = 341,  
3490 ["Rcedil"] = 342,  
3491 ["rcedil"] = 343,  
3492 ["Rcaron"] = 344,  
3493 ["rcaron"] = 345,  
3494 ["Sacute"] = 346,  
3495 ["sacute"] = 347,  
3496 ["Scirc"] = 348,  
3497 ["scirc"] = 349,  
3498 ["Scedil"] = 350,  
3499 ["scedil"] = 351,  
3500 ["Scaron"] = 352,  
3501 ["scaron"] = 353,  
3502 ["Tcedil"] = 354,  
3503 ["tcedil"] = 355,  
3504 ["Tcaron"] = 356,  
3505 ["tcaron"] = 357,  
3506 ["Tstrok"] = 358,  
3507 ["tstrok"] = 359,  
3508 ["Utilde"] = 360,  
3509 ["utilde"] = 361,  
3510 ["Umacr"] = 362,  
3511 ["umacr"] = 363,  
3512 ["Ubreve"] = 364,  
3513 ["ubreve"] = 365,  
3514 ["Uring"] = 366,  
3515 ["uring"] = 367,  
3516 ["Udblac"] = 368,  
3517 ["udblac"] = 369,  
3518 ["Uogon"] = 370,  
3519 ["uogon"] = 371,  
3520 ["Wcirc"] = 372,

3521 ["wcirc"] = 373,  
3522 ["Ycirc"] = 374,  
3523 ["ycirc"] = 375,  
3524 ["Yuml"] = 376,  
3525 ["Zacute"] = 377,  
3526 ["zacute"] = 378,  
3527 ["Zdot"] = 379,  
3528 ["zdot"] = 380,  
3529 ["Zcaron"] = 381,  
3530 ["zcaron"] = 382,  
3531 ["fnof"] = 402,  
3532 ["imped"] = 437,  
3533 ["gacute"] = 501,  
3534 ["jmath"] = 567,  
3535 ["circ"] = 710,  
3536 ["caron"] = 711,  
3537 ["Hacek"] = 711,  
3538 ["breve"] = 728,  
3539 ["Breve"] = 728,  
3540 ["dot"] = 729,  
3541 ["DiacriticalDot"] = 729,  
3542 ["ring"] = 730,  
3543 ["ogon"] = 731,  
3544 ["tilde"] = 732,  
3545 ["DiacriticalTilde"] = 732,  
3546 ["dblac"] = 733,  
3547 ["DiacriticalDoubleAcute"] = 733,  
3548 ["DownBreve"] = 785,  
3549 ["UnderBar"] = 818,  
3550 ["Alpha"] = 913,  
3551 ["Beta"] = 914,  
3552 ["Gamma"] = 915,  
3553 ["Delta"] = 916,  
3554 ["Epsilon"] = 917,  
3555 ["Zeta"] = 918,  
3556 ["Eta"] = 919,  
3557 ["Theta"] = 920,  
3558 ["Iota"] = 921,  
3559 ["Kappa"] = 922,  
3560 ["Lambda"] = 923,  
3561 ["Mu"] = 924,  
3562 ["Nu"] = 925,  
3563 ["Xi"] = 926,  
3564 ["Omicron"] = 927,  
3565 ["Pi"] = 928,  
3566 ["Rho"] = 929,  
3567 ["Sigma"] = 931,

3568 ["Tau"] = 932,  
3569 ["Upsilon"] = 933,  
3570 ["Phi"] = 934,  
3571 ["Chi"] = 935,  
3572 ["Psi"] = 936,  
3573 ["Omega"] = 937,  
3574 ["alpha"] = 945,  
3575 ["beta"] = 946,  
3576 ["gamma"] = 947,  
3577 ["delta"] = 948,  
3578 ["epsiv"] = 949,  
3579 ["varepsilon"] = 949,  
3580 ["epsilon"] = 949,  
3581 ["zeta"] = 950,  
3582 ["eta"] = 951,  
3583 ["theta"] = 952,  
3584 ["iota"] = 953,  
3585 ["kappa"] = 954,  
3586 ["lambda"] = 955,  
3587 ["mu"] = 956,  
3588 ["nu"] = 957,  
3589 ["xi"] = 958,  
3590 ["omicron"] = 959,  
3591 ["pi"] = 960,  
3592 ["rho"] = 961,  
3593 ["sigmav"] = 962,  
3594 ["varsigma"] = 962,  
3595 ["sigmaf"] = 962,  
3596 ["sigma"] = 963,  
3597 ["tau"] = 964,  
3598 ["upsi"] = 965,  
3599 ["upsilon"] = 965,  
3600 ["phi"] = 966,  
3601 ["phiv"] = 966,  
3602 ["varphi"] = 966,  
3603 ["chi"] = 967,  
3604 ["psi"] = 968,  
3605 ["omega"] = 969,  
3606 ["thetav"] = 977,  
3607 ["vartheta"] = 977,  
3608 ["thetasym"] = 977,  
3609 ["Upsi"] = 978,  
3610 ["upsih"] = 978,  
3611 ["straightphi"] = 981,  
3612 ["piv"] = 982,  
3613 ["varpi"] = 982,  
3614 ["Gammad"] = 988,

3615 ["gammad"] = 989,  
3616 ["digamma"] = 989,  
3617 ["kappav"] = 1008,  
3618 ["varkappa"] = 1008,  
3619 ["rhov"] = 1009,  
3620 ["varrho"] = 1009,  
3621 ["epsi"] = 1013,  
3622 ["straightepsilon"] = 1013,  
3623 ["bepsi"] = 1014,  
3624 ["backepsilon"] = 1014,  
3625 ["IOcy"] = 1025,  
3626 ["DJcy"] = 1026,  
3627 ["GJcy"] = 1027,  
3628 ["Jukcy"] = 1028,  
3629 ["DScy"] = 1029,  
3630 ["Iukcy"] = 1030,  
3631 ["YIcy"] = 1031,  
3632 ["Jsercy"] = 1032,  
3633 ["LJcy"] = 1033,  
3634 ["NJcy"] = 1034,  
3635 ["TSHcy"] = 1035,  
3636 ["KJcy"] = 1036,  
3637 ["Ubrcy"] = 1038,  
3638 ["DZcy"] = 1039,  
3639 ["Acy"] = 1040,  
3640 ["Bcy"] = 1041,  
3641 ["Vcy"] = 1042,  
3642 ["Gcy"] = 1043,  
3643 ["Dcy"] = 1044,  
3644 ["IEcy"] = 1045,  
3645 ["ZHcy"] = 1046,  
3646 ["Zcy"] = 1047,  
3647 ["Icy"] = 1048,  
3648 ["Jcy"] = 1049,  
3649 ["Kcy"] = 1050,  
3650 ["Lcy"] = 1051,  
3651 ["Mcy"] = 1052,  
3652 ["Ncy"] = 1053,  
3653 ["Ocy"] = 1054,  
3654 ["Pcy"] = 1055,  
3655 ["Rcy"] = 1056,  
3656 ["Scy"] = 1057,  
3657 ["Tcy"] = 1058,  
3658 ["Ucy"] = 1059,  
3659 ["Fcy"] = 1060,  
3660 ["KHcy"] = 1061,  
3661 ["TScy"] = 1062,

3662 ["CHcy"] = 1063,  
3663 ["SHcy"] = 1064,  
3664 ["SHCHcy"] = 1065,  
3665 ["HARDcy"] = 1066,  
3666 ["Ycy"] = 1067,  
3667 ["SOFTcy"] = 1068,  
3668 ["Ecy"] = 1069,  
3669 ["YUcy"] = 1070,  
3670 ["YAcy"] = 1071,  
3671 ["acy"] = 1072,  
3672 ["bcy"] = 1073,  
3673 ["vcy"] = 1074,  
3674 ["gcy"] = 1075,  
3675 ["dcy"] = 1076,  
3676 ["iecy"] = 1077,  
3677 ["zhcy"] = 1078,  
3678 ["zcy"] = 1079,  
3679 ["icy"] = 1080,  
3680 ["jcy"] = 1081,  
3681 ["kcy"] = 1082,  
3682 ["lcy"] = 1083,  
3683 ["mcy"] = 1084,  
3684 ["ncy"] = 1085,  
3685 ["ocy"] = 1086,  
3686 ["pcy"] = 1087,  
3687 ["rcy"] = 1088,  
3688 ["scy"] = 1089,  
3689 ["tcy"] = 1090,  
3690 ["ucy"] = 1091,  
3691 ["fcy"] = 1092,  
3692 ["khcy"] = 1093,  
3693 ["tscy"] = 1094,  
3694 ["chcy"] = 1095,  
3695 ["shcy"] = 1096,  
3696 ["shchcy"] = 1097,  
3697 ["hardcy"] = 1098,  
3698 ["ycy"] = 1099,  
3699 ["softcy"] = 1100,  
3700 ["ecy"] = 1101,  
3701 ["yucy"] = 1102,  
3702 ["yacy"] = 1103,  
3703 ["iocy"] = 1105,  
3704 ["djcy"] = 1106,  
3705 ["gjcy"] = 1107,  
3706 ["jukcy"] = 1108,  
3707 ["dscy"] = 1109,  
3708 ["iukcy"] = 1110,

3709 ["yicy"] = 1111,  
3710 ["jsercy"] = 1112,  
3711 ["ljcy"] = 1113,  
3712 ["njcy"] = 1114,  
3713 ["tshcy"] = 1115,  
3714 ["kjcy"] = 1116,  
3715 ["ubrcy"] = 1118,  
3716 ["dzcy"] = 1119,  
3717 ["ensp"] = 8194,  
3718 ["emsp"] = 8195,  
3719 ["emsp13"] = 8196,  
3720 ["emsp14"] = 8197,  
3721 ["numsp"] = 8199,  
3722 ["puncsp"] = 8200,  
3723 ["thinsp"] = 8201,  
3724 ["ThinSpace"] = 8201,  
3725 ["hairsp"] = 8202,  
3726 ["VeryThinSpace"] = 8202,  
3727 ["ZeroWidthSpace"] = 8203,  
3728 ["NegativeVeryThinSpace"] = 8203,  
3729 ["NegativeThinSpace"] = 8203,  
3730 ["NegativeMediumSpace"] = 8203,  
3731 ["NegativeThickSpace"] = 8203,  
3732 ["zwnj"] = 8204,  
3733 ["zwj"] = 8205,  
3734 ["lrm"] = 8206,  
3735 ["rlm"] = 8207,  
3736 ["hyphen"] = 8208,  
3737 ["dash"] = 8208,  
3738 ["ndash"] = 8211,  
3739 ["mdash"] = 8212,  
3740 ["horbar"] = 8213,  
3741 ["Verbar"] = 8214,  
3742 ["Vert"] = 8214,  
3743 ["lsquo"] = 8216,  
3744 ["OpenCurlyQuote"] = 8216,  
3745 ["rsquo"] = 8217,  
3746 ["rsquor"] = 8217,  
3747 ["CloseCurlyQuote"] = 8217,  
3748 ["lsquor"] = 8218,  
3749 ["sbquo"] = 8218,  
3750 ["ldquo"] = 8220,  
3751 ["OpenCurlyDoubleQuote"] = 8220,  
3752 ["rdquo"] = 8221,  
3753 ["rdquor"] = 8221,  
3754 ["CloseCurlyDoubleQuote"] = 8221,  
3755 ["ldquor"] = 8222,

3756 ["bdquo"] = 8222,  
3757 ["dagger"] = 8224,  
3758 ["Dagger"] = 8225,  
3759 ["ddagger"] = 8225,  
3760 ["bull"] = 8226,  
3761 ["bullet"] = 8226,  
3762 ["nldr"] = 8229,  
3763 ["hellip"] = 8230,  
3764 ["mldr"] = 8230,  
3765 ["permil"] = 8240,  
3766 ["pertenk"] = 8241,  
3767 ["prime"] = 8242,  
3768 ["Prime"] = 8243,  
3769 ["tprime"] = 8244,  
3770 ["bprime"] = 8245,  
3771 ["backprime"] = 8245,  
3772 ["lsaquo"] = 8249,  
3773 ["rsaquo"] = 8250,  
3774 ["oline"] = 8254,  
3775 ["caret"] = 8257,  
3776 ["hybull"] = 8259,  
3777 ["fras1"] = 8260,  
3778 ["bsemi"] = 8271,  
3779 ["qprime"] = 8279,  
3780 ["MediumSpace"] = 8287,  
3781 ["NoBreak"] = 8288,  
3782 ["ApplyFunction"] = 8289,  
3783 ["af"] = 8289,  
3784 ["InvisibleTimes"] = 8290,  
3785 ["it"] = 8290,  
3786 ["InvisibleComma"] = 8291,  
3787 ["ic"] = 8291,  
3788 ["euro"] = 8364,  
3789 ["tdot"] = 8411,  
3790 ["TripleDot"] = 8411,  
3791 ["DotDot"] = 8412,  
3792 ["Copf"] = 8450,  
3793 ["complexes"] = 8450,  
3794 ["incare"] = 8453,  
3795 ["gscr"] = 8458,  
3796 ["hamilt"] = 8459,  
3797 ["HilbertSpace"] = 8459,  
3798 ["Hscr"] = 8459,  
3799 ["Hfr"] = 8460,  
3800 ["Poincareplane"] = 8460,  
3801 ["quaternions"] = 8461,  
3802 ["Hopf"] = 8461,

3803 ["planckh"] = 8462,  
3804 ["planck"] = 8463,  
3805 ["hbar"] = 8463,  
3806 ["plankv"] = 8463,  
3807 ["hslash"] = 8463,  
3808 ["Iscr"] = 8464,  
3809 ["imagline"] = 8464,  
3810 ["image"] = 8465,  
3811 ["Im"] = 8465,  
3812 ["imagpart"] = 8465,  
3813 ["Ifr"] = 8465,  
3814 ["Lscr"] = 8466,  
3815 ["lagran"] = 8466,  
3816 ["Laplacetrif"] = 8466,  
3817 ["ell"] = 8467,  
3818 ["Nopf"] = 8469,  
3819 ["naturals"] = 8469,  
3820 ["numero"] = 8470,  
3821 ["copysr"] = 8471,  
3822 ["weierp"] = 8472,  
3823 ["wp"] = 8472,  
3824 ["Popf"] = 8473,  
3825 ["primes"] = 8473,  
3826 ["rationals"] = 8474,  
3827 ["Qopf"] = 8474,  
3828 ["Rscr"] = 8475,  
3829 ["realine"] = 8475,  
3830 ["real"] = 8476,  
3831 ["Re"] = 8476,  
3832 ["realpart"] = 8476,  
3833 ["Rfr"] = 8476,  
3834 ["reals"] = 8477,  
3835 ["Ropf"] = 8477,  
3836 ["rx"] = 8478,  
3837 ["trade"] = 8482,  
3838 ["TRADE"] = 8482,  
3839 ["integers"] = 8484,  
3840 ["Zopf"] = 8484,  
3841 ["ohm"] = 8486,  
3842 ["mho"] = 8487,  
3843 ["Zfr"] = 8488,  
3844 ["zeetrif"] = 8488,  
3845 ["iiota"] = 8489,  
3846 ["angst"] = 8491,  
3847 ["bernou"] = 8492,  
3848 ["Bernoullis"] = 8492,  
3849 ["Bscr"] = 8492,



3850 ["Cfr"] = 8493,  
3851 ["Cayleys"] = 8493,  
3852 ["escr"] = 8495,  
3853 ["Escr"] = 8496,  
3854 ["expectation"] = 8496,  
3855 ["Fscr"] = 8497,  
3856 ["Fouriertrf"] = 8497,  
3857 ["phmmat"] = 8499,  
3858 ["Mellintrf"] = 8499,  
3859 ["Mscr"] = 8499,  
3860 ["order"] = 8500,  
3861 ["orderof"] = 8500,  
3862 ["oscr"] = 8500,  
3863 ["alefsym"] = 8501,  
3864 ["aleph"] = 8501,  
3865 ["beth"] = 8502,  
3866 ["gimel"] = 8503,  
3867 ["daleth"] = 8504,  
3868 ["CapitalDifferentialD"] = 8517,  
3869 ["DD"] = 8517,  
3870 ["DifferentialD"] = 8518,  
3871 ["dd"] = 8518,  
3872 ["ExponentialE"] = 8519,  
3873 ["exponentiale"] = 8519,  
3874 ["ee"] = 8519,  
3875 ["ImaginaryI"] = 8520,  
3876 ["ii"] = 8520,  
3877 ["frac13"] = 8531,  
3878 ["frac23"] = 8532,  
3879 ["frac15"] = 8533,  
3880 ["frac25"] = 8534,  
3881 ["frac35"] = 8535,  
3882 ["frac45"] = 8536,  
3883 ["frac16"] = 8537,  
3884 ["frac56"] = 8538,  
3885 ["frac18"] = 8539,  
3886 ["frac38"] = 8540,  
3887 ["frac58"] = 8541,  
3888 ["frac78"] = 8542,  
3889 ["larr"] = 8592,  
3890 ["leftarrow"] = 8592,  
3891 ["LeftArrow"] = 8592,  
3892 ["slarr"] = 8592,  
3893 ["ShortLeftArrow"] = 8592,  
3894 ["uarr"] = 8593,  
3895 ["uparrow"] = 8593,  
3896 ["UpArrow"] = 8593,

```

3897 ["ShortUpArrow"] = 8593,
3898 ["rarr"] = 8594,
3899 ["rightarrow"] = 8594,
3900 ["RightArrow"] = 8594,
3901 ["srarr"] = 8594,
3902 ["ShortRightArrow"] = 8594,
3903 ["darr"] = 8595,
3904 ["downarrow"] = 8595,
3905 ["DownArrow"] = 8595,
3906 ["ShortDownArrow"] = 8595,
3907 ["harr"] = 8596,
3908 ["leftrightarrow"] = 8596,
3909 ["LeftRightArrow"] = 8596,
3910 ["varr"] = 8597,
3911 ["updownarrow"] = 8597,
3912 ["UpDownArrow"] = 8597,
3913 ["nwarr"] = 8598,
3914 ["UpperLeftArrow"] = 8598,
3915 ["nwarrow"] = 8598,
3916 ["nearr"] = 8599,
3917 ["UpperRightArrow"] = 8599,
3918 ["nearrow"] = 8599,
3919 ["searr"] = 8600,
3920 ["searrow"] = 8600,
3921 ["LowerRightArrow"] = 8600,
3922 ["swarr"] = 8601,
3923 ["swarrow"] = 8601,
3924 ["LowerLeftArrow"] = 8601,
3925 ["nlarr"] = 8602,
3926 ["nleftarrow"] = 8602,
3927 ["nrarr"] = 8603,
3928 ["nrightarrow"] = 8603,
3929 ["rarrw"] = 8605,
3930 ["rightsquigarrow"] = 8605,
3931 ["Larr"] = 8606,
3932 ["twoheadleftarrow"] = 8606,
3933 ["Uarr"] = 8607,
3934 ["Rarr"] = 8608,
3935 ["twoheadrightarrow"] = 8608,
3936 ["Darr"] = 8609,
3937 ["larrtl"] = 8610,
3938 ["leftarrowtail"] = 8610,
3939 ["rarrtl"] = 8611,
3940 ["rightarrowtail"] = 8611,
3941 ["LeftTeeArrow"] = 8612,
3942 ["mapstoleft"] = 8612,
3943 ["UpTeeArrow"] = 8613,

```

3944 ["mapstoup"] = 8613,  
 3945 ["map"] = 8614,  
 3946 ["RightTeeArrow"] = 8614,  
 3947 ["mapsto"] = 8614,  
 3948 ["DownTeeArrow"] = 8615,  
 3949 ["mapstodown"] = 8615,  
 3950 ["larrhk"] = 8617,  
 3951 ["hookleftarrow"] = 8617,  
 3952 ["rarrhk"] = 8618,  
 3953 ["hookrightarrow"] = 8618,  
 3954 ["larrlp"] = 8619,  
 3955 ["looparrowleft"] = 8619,  
 3956 ["rarrlp"] = 8620,  
 3957 ["looparrowright"] = 8620,  
 3958 ["harrw"] = 8621,  
 3959 ["leftrightsquigarrow"] = 8621,  
 3960 ["nharr"] = 8622,  
 3961 ["nletrightarrow"] = 8622,  
 3962 ["lsh"] = 8624,  
 3963 ["Lsh"] = 8624,  
 3964 ["rsh"] = 8625,  
 3965 ["Rsh"] = 8625,  
 3966 ["ldsh"] = 8626,  
 3967 ["rdsh"] = 8627,  
 3968 ["crarr"] = 8629,  
 3969 ["cularr"] = 8630,  
 3970 ["curvearrowleft"] = 8630,  
 3971 ["curarr"] = 8631,  
 3972 ["curvearrowright"] = 8631,  
 3973 ["olarr"] = 8634,  
 3974 ["circlearrowleft"] = 8634,  
 3975 ["orarr"] = 8635,  
 3976 ["circlearrowright"] = 8635,  
 3977 ["lharu"] = 8636,  
 3978 ["LeftVector"] = 8636,  
 3979 ["leftharpoonup"] = 8636,  
 3980 ["lhard"] = 8637,  
 3981 ["leftharpoondown"] = 8637,  
 3982 ["DownLeftVector"] = 8637,  
 3983 ["uharr"] = 8638,  
 3984 ["upharpoonright"] = 8638,  
 3985 ["RightUpVector"] = 8638,  
 3986 ["uharl"] = 8639,  
 3987 ["upharpoonleft"] = 8639,  
 3988 ["LeftUpVector"] = 8639,  
 3989 ["rharu"] = 8640,  
 3990 ["RightVector"] = 8640,

3991 ["rightharpoonup"] = 8640,  
 3992 ["rhard"] = 8641,  
 3993 ["rightharpoondown"] = 8641,  
 3994 ["DownRightVector"] = 8641,  
 3995 ["dharr"] = 8642,  
 3996 ["RightDownVector"] = 8642,  
 3997 ["downharpoonright"] = 8642,  
 3998 ["dharl"] = 8643,  
 3999 ["LeftDownVector"] = 8643,  
 4000 ["downharpoonleft"] = 8643,  
 4001 ["rlarr"] = 8644,  
 4002 ["rightleftarrows"] = 8644,  
 4003 ["RightArrowLeftArrow"] = 8644,  
 4004 ["udarr"] = 8645,  
 4005 ["UpArrowDownArrow"] = 8645,  
 4006 ["lrarr"] = 8646,  
 4007 ["leftrightarrows"] = 8646,  
 4008 ["LeftArrowRightArrow"] = 8646,  
 4009 ["llarr"] = 8647,  
 4010 ["leftleftarrows"] = 8647,  
 4011 ["uuarr"] = 8648,  
 4012 ["upuparrows"] = 8648,  
 4013 ["rrarr"] = 8649,  
 4014 ["rightrightarrows"] = 8649,  
 4015 ["ddarr"] = 8650,  
 4016 ["downdownarrows"] = 8650,  
 4017 ["lrhar"] = 8651,  
 4018 ["ReverseEquilibrium"] = 8651,  
 4019 ["leftrightharpoons"] = 8651,  
 4020 ["rlhar"] = 8652,  
 4021 ["rightleftharpoons"] = 8652,  
 4022 ["Equilibrium"] = 8652,  
 4023 ["nlArr"] = 8653,  
 4024 ["nLeftarrow"] = 8653,  
 4025 ["nhArr"] = 8654,  
 4026 ["nLeftrightarrow"] = 8654,  
 4027 ["nrArr"] = 8655,  
 4028 ["nRightarrow"] = 8655,  
 4029 ["lArr"] = 8656,  
 4030 ["Leftarrow"] = 8656,  
 4031 ["DoubleLeftArrow"] = 8656,  
 4032 ["uArr"] = 8657,  
 4033 ["Uparrow"] = 8657,  
 4034 ["DoubleUpArrow"] = 8657,  
 4035 ["rArr"] = 8658,  
 4036 ["Rightarrow"] = 8658,  
 4037 ["Implies"] = 8658,

4038 ["DoubleRightArrow"] = 8658,  
 4039 ["dArr"] = 8659,  
 4040 ["Downarrow"] = 8659,  
 4041 ["DoubleDownArrow"] = 8659,  
 4042 ["hArr"] = 8660,  
 4043 ["Leftrightarrow"] = 8660,  
 4044 ["DoubleLeftRightArrow"] = 8660,  
 4045 ["iff"] = 8660,  
 4046 ["vArr"] = 8661,  
 4047 ["Updownarrow"] = 8661,  
 4048 ["DoubleUpDownArrow"] = 8661,  
 4049 ["nwArr"] = 8662,  
 4050 ["neArr"] = 8663,  
 4051 ["seArr"] = 8664,  
 4052 ["swArr"] = 8665,  
 4053 ["lAarr"] = 8666,  
 4054 ["Lleftarrow"] = 8666,  
 4055 ["rAarr"] = 8667,  
 4056 ["Rrightarrow"] = 8667,  
 4057 ["zigrarr"] = 8669,  
 4058 ["larrb"] = 8676,  
 4059 ["LeftArrowBar"] = 8676,  
 4060 ["rarrb"] = 8677,  
 4061 ["RightArrowBar"] = 8677,  
 4062 ["duarr"] = 8693,  
 4063 ["DownArrowUpArrow"] = 8693,  
 4064 ["loarr"] = 8701,  
 4065 ["roarr"] = 8702,  
 4066 ["hoarr"] = 8703,  
 4067 ["forall"] = 8704,  
 4068 ["ForAll"] = 8704,  
 4069 ["comp"] = 8705,  
 4070 ["complement"] = 8705,  
 4071 ["part"] = 8706,  
 4072 ["PartialD"] = 8706,  
 4073 ["exist"] = 8707,  
 4074 ["Exists"] = 8707,  
 4075 ["nexist"] = 8708,  
 4076 ["NotExists"] = 8708,  
 4077 ["nexists"] = 8708,  
 4078 ["empty"] = 8709,  
 4079 ["emptyset"] = 8709,  
 4080 ["emptyv"] = 8709,  
 4081 ["varnothing"] = 8709,  
 4082 ["nabla"] = 8711,  
 4083 ["Del"] = 8711,  
 4084 ["isin"] = 8712,

4085 ["isinv"] = 8712,  
 4086 ["Element"] = 8712,  
 4087 ["in"] = 8712,  
 4088 ["notin"] = 8713,  
 4089 ["NotElement"] = 8713,  
 4090 ["notinva"] = 8713,  
 4091 ["niv"] = 8715,  
 4092 ["ReverseElement"] = 8715,  
 4093 ["ni"] = 8715,  
 4094 ["SuchThat"] = 8715,  
 4095 ["notni"] = 8716,  
 4096 ["notniva"] = 8716,  
 4097 ["NotReverseElement"] = 8716,  
 4098 ["prod"] = 8719,  
 4099 ["Product"] = 8719,  
 4100 ["coprod"] = 8720,  
 4101 ["Coproduct"] = 8720,  
 4102 ["sum"] = 8721,  
 4103 ["Sum"] = 8721,  
 4104 ["minus"] = 8722,  
 4105 ["mnplus"] = 8723,  
 4106 ["mp"] = 8723,  
 4107 ["MinusPlus"] = 8723,  
 4108 ["plusdo"] = 8724,  
 4109 ["dotplus"] = 8724,  
 4110 ["setmn"] = 8726,  
 4111 ["setminus"] = 8726,  
 4112 ["Backslash"] = 8726,  
 4113 ["ssetmn"] = 8726,  
 4114 ["smallsetminus"] = 8726,  
 4115 ["lowast"] = 8727,  
 4116 ["compfn"] = 8728,  
 4117 ["SmallCircle"] = 8728,  
 4118 ["radic"] = 8730,  
 4119 ["Sqrt"] = 8730,  
 4120 ["prop"] = 8733,  
 4121 ["propto"] = 8733,  
 4122 ["Proportional"] = 8733,  
 4123 ["vprop"] = 8733,  
 4124 ["varpropto"] = 8733,  
 4125 ["infin"] = 8734,  
 4126 ["angrt"] = 8735,  
 4127 ["ang"] = 8736,  
 4128 ["angle"] = 8736,  
 4129 ["angmsd"] = 8737,  
 4130 ["measuredangle"] = 8737,  
 4131 ["angsph"] = 8738,

4132 ["mid"] = 8739,  
4133 ["VerticalBar"] = 8739,  
4134 ["smid"] = 8739,  
4135 ["shortmid"] = 8739,  
4136 ["nmid"] = 8740,  
4137 ["NotVerticalBar"] = 8740,  
4138 ["nsmid"] = 8740,  
4139 ["nshortmid"] = 8740,  
4140 ["par"] = 8741,  
4141 ["parallel"] = 8741,  
4142 ["DoubleVerticalBar"] = 8741,  
4143 ["spar"] = 8741,  
4144 ["shortparallel"] = 8741,  
4145 ["npar"] = 8742,  
4146 ["nparallel"] = 8742,  
4147 ["NotDoubleVerticalBar"] = 8742,  
4148 ["nspar"] = 8742,  
4149 ["nshortparallel"] = 8742,  
4150 ["and"] = 8743,  
4151 ["wedge"] = 8743,  
4152 ["or"] = 8744,  
4153 ["vee"] = 8744,  
4154 ["cap"] = 8745,  
4155 ["cup"] = 8746,  
4156 ["int"] = 8747,  
4157 ["Integral"] = 8747,  
4158 ["Int"] = 8748,  
4159 ["tint"] = 8749,  
4160 ["iiint"] = 8749,  
4161 ["conint"] = 8750,  
4162 ["oint"] = 8750,  
4163 ["ContourIntegral"] = 8750,  
4164 ["Conint"] = 8751,  
4165 ["DoubleContourIntegral"] = 8751,  
4166 ["Cconint"] = 8752,  
4167 ["cwint"] = 8753,  
4168 ["cwconint"] = 8754,  
4169 ["ClockwiseContourIntegral"] = 8754,  
4170 ["awconint"] = 8755,  
4171 ["CounterClockwiseContourIntegral"] = 8755,  
4172 ["there4"] = 8756,  
4173 ["therefore"] = 8756,  
4174 ["Therefore"] = 8756,  
4175 ["because"] = 8757,  
4176 ["because"] = 8757,  
4177 ["Because"] = 8757,  
4178 ["ratio"] = 8758,

4179 ["Colon"] = 8759,  
 4180 ["Proportion"] = 8759,  
 4181 ["minusd"] = 8760,  
 4182 ["dotminus"] = 8760,  
 4183 ["mDDot"] = 8762,  
 4184 ["homtht"] = 8763,  
 4185 ["sim"] = 8764,  
 4186 ["Tilde"] = 8764,  
 4187 ["thksim"] = 8764,  
 4188 ["thicksim"] = 8764,  
 4189 ["bsim"] = 8765,  
 4190 ["backsim"] = 8765,  
 4191 ["ac"] = 8766,  
 4192 ["mstpos"] = 8766,  
 4193 ["acd"] = 8767,  
 4194 ["wreath"] = 8768,  
 4195 ["VerticalTilde"] = 8768,  
 4196 ["wr"] = 8768,  
 4197 ["nsim"] = 8769,  
 4198 ["NotTilde"] = 8769,  
 4199 ["esim"] = 8770,  
 4200 ["EqualTilde"] = 8770,  
 4201 ["eqsim"] = 8770,  
 4202 ["sime"] = 8771,  
 4203 ["TildeEqual"] = 8771,  
 4204 ["simeq"] = 8771,  
 4205 ["nsime"] = 8772,  
 4206 ["nsimeq"] = 8772,  
 4207 ["NotTildeEqual"] = 8772,  
 4208 ["cong"] = 8773,  
 4209 ["TildeFullEqual"] = 8773,  
 4210 ["simne"] = 8774,  
 4211 ["ncong"] = 8775,  
 4212 ["NotTildeFullEqual"] = 8775,  
 4213 ["asymp"] = 8776,  
 4214 ["ap"] = 8776,  
 4215 ["TildeTilde"] = 8776,  
 4216 ["approx"] = 8776,  
 4217 ["thkap"] = 8776,  
 4218 ["thickapprox"] = 8776,  
 4219 ["nap"] = 8777,  
 4220 ["NotTildeTilde"] = 8777,  
 4221 ["napprox"] = 8777,  
 4222 ["ape"] = 8778,  
 4223 ["approxq"] = 8778,  
 4224 ["apid"] = 8779,  
 4225 ["bcong"] = 8780,



4226 ["backcong"] = 8780,  
 4227 ["asympeq"] = 8781,  
 4228 ["CupCap"] = 8781,  
 4229 ["bump"] = 8782,  
 4230 ["HumpDownHump"] = 8782,  
 4231 ["Bumpeq"] = 8782,  
 4232 ["bumpe"] = 8783,  
 4233 ["HumpEqual"] = 8783,  
 4234 ["bumpeq"] = 8783,  
 4235 ["esdot"] = 8784,  
 4236 ["DotEqual"] = 8784,  
 4237 ["doteq"] = 8784,  
 4238 ["eDot"] = 8785,  
 4239 ["doteqdot"] = 8785,  
 4240 ["efDot"] = 8786,  
 4241 ["fallingdotseq"] = 8786,  
 4242 ["erDot"] = 8787,  
 4243 ["risingdotseq"] = 8787,  
 4244 ["colone"] = 8788,  
 4245 ["coloneq"] = 8788,  
 4246 ["Assign"] = 8788,  
 4247 ["ecolon"] = 8789,  
 4248 ["eqcolon"] = 8789,  
 4249 ["ecir"] = 8790,  
 4250 ["eqcirc"] = 8790,  
 4251 ["cire"] = 8791,  
 4252 ["circeq"] = 8791,  
 4253 ["wedgeq"] = 8793,  
 4254 ["veeeq"] = 8794,  
 4255 ["trie"] = 8796,  
 4256 ["triangleq"] = 8796,  
 4257 ["equest"] = 8799,  
 4258 ["questeq"] = 8799,  
 4259 ["ne"] = 8800,  
 4260 ["NotEqual"] = 8800,  
 4261 ["equiv"] = 8801,  
 4262 ["Congruent"] = 8801,  
 4263 ["nequiv"] = 8802,  
 4264 ["NotCongruent"] = 8802,  
 4265 ["le"] = 8804,  
 4266 ["leq"] = 8804,  
 4267 ["ge"] = 8805,  
 4268 ["GreaterEqual"] = 8805,  
 4269 ["geq"] = 8805,  
 4270 ["lE"] = 8806,  
 4271 ["LessFullEqual"] = 8806,  
 4272 ["leqq"] = 8806,

4273 ["gE"] = 8807,  
 4274 ["GreaterFullEqual"] = 8807,  
 4275 ["geqq"] = 8807,  
 4276 ["lnE"] = 8808,  
 4277 ["lneqq"] = 8808,  
 4278 ["gnE"] = 8809,  
 4279 ["gneqq"] = 8809,  
 4280 ["Lt"] = 8810,  
 4281 ["NestedLessLess"] = 8810,  
 4282 ["ll"] = 8810,  
 4283 ["Gt"] = 8811,  
 4284 ["NestedGreaterGreater"] = 8811,  
 4285 ["gg"] = 8811,  
 4286 ["twixt"] = 8812,  
 4287 ["between"] = 8812,  
 4288 ["NotCupCap"] = 8813,  
 4289 ["nlt"] = 8814,  
 4290 ["NotLess"] = 8814,  
 4291 ["nless"] = 8814,  
 4292 ["ngt"] = 8815,  
 4293 ["NotGreater"] = 8815,  
 4294 ["ngtr"] = 8815,  
 4295 ["nle"] = 8816,  
 4296 ["NotLessEqual"] = 8816,  
 4297 ["nleq"] = 8816,  
 4298 ["nge"] = 8817,  
 4299 ["NotGreaterEqual"] = 8817,  
 4300 ["ngeq"] = 8817,  
 4301 ["lsim"] = 8818,  
 4302 ["LessTilde"] = 8818,  
 4303 ["lesssim"] = 8818,  
 4304 ["gsim"] = 8819,  
 4305 ["gtrsim"] = 8819,  
 4306 ["GreaterTilde"] = 8819,  
 4307 ["nlsim"] = 8820,  
 4308 ["NotLessTilde"] = 8820,  
 4309 ["ngsim"] = 8821,  
 4310 ["NotGreaterTilde"] = 8821,  
 4311 ["lg"] = 8822,  
 4312 ["lessgtr"] = 8822,  
 4313 ["LessGreater"] = 8822,  
 4314 ["gl"] = 8823,  
 4315 ["gtrless"] = 8823,  
 4316 ["GreaterLess"] = 8823,  
 4317 ["ntlg"] = 8824,  
 4318 ["NotLessGreater"] = 8824,  
 4319 ["ntgl"] = 8825,

4320 ["NotGreaterLess"] = 8825,  
 4321 ["pr"] = 8826,  
 4322 ["Precedes"] = 8826,  
 4323 ["prec"] = 8826,  
 4324 ["sc"] = 8827,  
 4325 ["Succeeds"] = 8827,  
 4326 ["succ"] = 8827,  
 4327 ["prcue"] = 8828,  
 4328 ["PrecedesSlantEqual"] = 8828,  
 4329 ["preccurlyeq"] = 8828,  
 4330 ["sccue"] = 8829,  
 4331 ["SucceedsSlantEqual"] = 8829,  
 4332 ["succurlyeq"] = 8829,  
 4333 ["prsim"] = 8830,  
 4334 ["precsim"] = 8830,  
 4335 ["PrecedesTilde"] = 8830,  
 4336 ["scsim"] = 8831,  
 4337 ["succsim"] = 8831,  
 4338 ["SucceedsTilde"] = 8831,  
 4339 ["npr"] = 8832,  
 4340 ["nprec"] = 8832,  
 4341 ["NotPrecedes"] = 8832,  
 4342 ["nsc"] = 8833,  
 4343 ["nsucc"] = 8833,  
 4344 ["NotSucceeds"] = 8833,  
 4345 ["sub"] = 8834,  
 4346 ["subset"] = 8834,  
 4347 ["sup"] = 8835,  
 4348 ["supset"] = 8835,  
 4349 ["Superset"] = 8835,  
 4350 ["nsub"] = 8836,  
 4351 ["nsup"] = 8837,  
 4352 ["sube"] = 8838,  
 4353 ["SubsetEqual"] = 8838,  
 4354 ["subseteq"] = 8838,  
 4355 ["supe"] = 8839,  
 4356 ["supseteq"] = 8839,  
 4357 ["SupersetEqual"] = 8839,  
 4358 ["nsube"] = 8840,  
 4359 ["nsubseteq"] = 8840,  
 4360 ["NotSubsetEqual"] = 8840,  
 4361 ["nsupe"] = 8841,  
 4362 ["nsupseteq"] = 8841,  
 4363 ["NotSupersetEqual"] = 8841,  
 4364 ["subne"] = 8842,  
 4365 ["subsetneq"] = 8842,  
 4366 ["supne"] = 8843,

4367 ["supsetneq"] = 8843,  
 4368 ["cupdot"] = 8845,  
 4369 ["uplus"] = 8846,  
 4370 ["UnionPlus"] = 8846,  
 4371 ["sqsub"] = 8847,  
 4372 ["SquareSubset"] = 8847,  
 4373 ["sqsubset"] = 8847,  
 4374 ["sqsup"] = 8848,  
 4375 ["SquareSuperset"] = 8848,  
 4376 ["sqsupset"] = 8848,  
 4377 ["sqsube"] = 8849,  
 4378 ["SquareSubsetEqual"] = 8849,  
 4379 ["sqsubseteq"] = 8849,  
 4380 ["sqsupe"] = 8850,  
 4381 ["SquareSupersetEqual"] = 8850,  
 4382 ["sqsupseteq"] = 8850,  
 4383 ["sqcap"] = 8851,  
 4384 ["SquareIntersection"] = 8851,  
 4385 ["sqcup"] = 8852,  
 4386 ["SquareUnion"] = 8852,  
 4387 ["oplus"] = 8853,  
 4388 ["CirclePlus"] = 8853,  
 4389 ["ominus"] = 8854,  
 4390 ["CircleMinus"] = 8854,  
 4391 ["otimes"] = 8855,  
 4392 ["CircleTimes"] = 8855,  
 4393 ["osol"] = 8856,  
 4394 ["odot"] = 8857,  
 4395 ["CircleDot"] = 8857,  
 4396 ["ocir"] = 8858,  
 4397 ["circledcirc"] = 8858,  
 4398 ["oast"] = 8859,  
 4399 ["circledast"] = 8859,  
 4400 ["odash"] = 8861,  
 4401 ["circleddash"] = 8861,  
 4402 ["plusb"] = 8862,  
 4403 ["boxplus"] = 8862,  
 4404 ["minusb"] = 8863,  
 4405 ["boxminus"] = 8863,  
 4406 ["timesb"] = 8864,  
 4407 ["boxtimes"] = 8864,  
 4408 ["sdotb"] = 8865,  
 4409 ["dotsquare"] = 8865,  
 4410 ["vdash"] = 8866,  
 4411 ["RightTee"] = 8866,  
 4412 ["dashv"] = 8867,  
 4413 ["LeftTee"] = 8867,

4414 ["top"] = 8868,  
 4415 ["DownTee"] = 8868,  
 4416 ["bottom"] = 8869,  
 4417 ["bot"] = 8869,  
 4418 ["perp"] = 8869,  
 4419 ["UpTee"] = 8869,  
 4420 ["models"] = 8871,  
 4421 ["vDash"] = 8872,  
 4422 ["DoubleRightTee"] = 8872,  
 4423 ["Vdash"] = 8873,  
 4424 ["Vvdash"] = 8874,  
 4425 ["VDash"] = 8875,  
 4426 ["nvdash"] = 8876,  
 4427 ["nvDash"] = 8877,  
 4428 ["nVdash"] = 8878,  
 4429 ["nVDash"] = 8879,  
 4430 ["prurel"] = 8880,  
 4431 ["vltri"] = 8882,  
 4432 ["vartriangleleft"] = 8882,  
 4433 ["LeftTriangle"] = 8882,  
 4434 ["vrtri"] = 8883,  
 4435 ["vartriangleright"] = 8883,  
 4436 ["RightTriangle"] = 8883,  
 4437 ["ltrie"] = 8884,  
 4438 ["trianglelefteq"] = 8884,  
 4439 ["LeftTriangleEqual"] = 8884,  
 4440 ["rtrie"] = 8885,  
 4441 ["trianglerighteq"] = 8885,  
 4442 ["RightTriangleEqual"] = 8885,  
 4443 ["origof"] = 8886,  
 4444 ["imof"] = 8887,  
 4445 ["mumap"] = 8888,  
 4446 ["multimap"] = 8888,  
 4447 ["hercon"] = 8889,  
 4448 ["intcal"] = 8890,  
 4449 ["intercal"] = 8890,  
 4450 ["veebar"] = 8891,  
 4451 ["barvee"] = 8893,  
 4452 ["angrtvb"] = 8894,  
 4453 ["ltri"] = 8895,  
 4454 ["xwedge"] = 8896,  
 4455 ["Wedge"] = 8896,  
 4456 ["bigwedge"] = 8896,  
 4457 ["xvee"] = 8897,  
 4458 ["Vee"] = 8897,  
 4459 ["bigvee"] = 8897,  
 4460 ["xcap"] = 8898,

4461 ["Intersection"] = 8898,  
 4462 ["bigcap"] = 8898,  
 4463 ["xcup"] = 8899,  
 4464 ["Union"] = 8899,  
 4465 ["bigcup"] = 8899,  
 4466 ["diam"] = 8900,  
 4467 ["diamond"] = 8900,  
 4468 ["Diamond"] = 8900,  
 4469 ["sdot"] = 8901,  
 4470 ["sstarf"] = 8902,  
 4471 ["Star"] = 8902,  
 4472 ["divonx"] = 8903,  
 4473 ["divideontimes"] = 8903,  
 4474 ["bowtie"] = 8904,  
 4475 ["ltimes"] = 8905,  
 4476 ["rtimes"] = 8906,  
 4477 ["lthree"] = 8907,  
 4478 ["leftthreetimes"] = 8907,  
 4479 ["rthree"] = 8908,  
 4480 ["rightthreetimes"] = 8908,  
 4481 ["bsime"] = 8909,  
 4482 ["backsimeq"] = 8909,  
 4483 ["cuvee"] = 8910,  
 4484 ["curlyvee"] = 8910,  
 4485 ["cuwed"] = 8911,  
 4486 ["curlywedge"] = 8911,  
 4487 ["Sub"] = 8912,  
 4488 ["Subset"] = 8912,  
 4489 ["Sup"] = 8913,  
 4490 ["Supset"] = 8913,  
 4491 ["Cap"] = 8914,  
 4492 ["Cup"] = 8915,  
 4493 ["fork"] = 8916,  
 4494 ["pitchfork"] = 8916,  
 4495 ["epar"] = 8917,  
 4496 ["ltdot"] = 8918,  
 4497 ["lessdot"] = 8918,  
 4498 ["gtdot"] = 8919,  
 4499 ["gtrdot"] = 8919,  
 4500 ["Ll"] = 8920,  
 4501 ["Gg"] = 8921,  
 4502 ["ggg"] = 8921,  
 4503 ["leg"] = 8922,  
 4504 ["LessEqualGreater"] = 8922,  
 4505 ["lesseqgtr"] = 8922,  
 4506 ["gel"] = 8923,  
 4507 ["gtreqless"] = 8923,

4508 ["GreaterEqualLess"] = 8923,  
 4509 ["cuepr"] = 8926,  
 4510 ["curlyeqprec"] = 8926,  
 4511 ["cuesc"] = 8927,  
 4512 ["curlyeqsucc"] = 8927,  
 4513 ["nprcue"] = 8928,  
 4514 ["NotPrecedesSlantEqual"] = 8928,  
 4515 ["nsccue"] = 8929,  
 4516 ["NotSucceedsSlantEqual"] = 8929,  
 4517 ["nsqsube"] = 8930,  
 4518 ["NotSquareSubsetEqual"] = 8930,  
 4519 ["nsqsupe"] = 8931,  
 4520 ["NotSquareSupersetEqual"] = 8931,  
 4521 ["lnsim"] = 8934,  
 4522 ["gnsim"] = 8935,  
 4523 ["prnsim"] = 8936,  
 4524 ["precnsim"] = 8936,  
 4525 ["scnsim"] = 8937,  
 4526 ["succnsim"] = 8937,  
 4527 ["nltri"] = 8938,  
 4528 ["ntriangleleft"] = 8938,  
 4529 ["NotLeftTriangle"] = 8938,  
 4530 ["nrtri"] = 8939,  
 4531 ["ntriangleright"] = 8939,  
 4532 ["NotRightTriangle"] = 8939,  
 4533 ["nltrie"] = 8940,  
 4534 ["ntrianglelefteq"] = 8940,  
 4535 ["NotLeftTriangleEqual"] = 8940,  
 4536 ["nrtrie"] = 8941,  
 4537 ["ntrianglerighteq"] = 8941,  
 4538 ["NotRightTriangleEqual"] = 8941,  
 4539 ["vellip"] = 8942,  
 4540 ["ctdot"] = 8943,  
 4541 ["utdot"] = 8944,  
 4542 ["dtdot"] = 8945,  
 4543 ["disin"] = 8946,  
 4544 ["isinsv"] = 8947,  
 4545 ["isins"] = 8948,  
 4546 ["isindot"] = 8949,  
 4547 ["notinvc"] = 8950,  
 4548 ["notinvb"] = 8951,  
 4549 ["isinE"] = 8953,  
 4550 ["nisd"] = 8954,  
 4551 ["xnis"] = 8955,  
 4552 ["nis"] = 8956,  
 4553 ["notnivc"] = 8957,  
 4554 ["notnivb"] = 8958,

4555 ["barwed"] = 8965,  
4556 ["barwedge"] = 8965,  
4557 ["Barwed"] = 8966,  
4558 ["doublebarwedge"] = 8966,  
4559 ["lceil"] = 8968,  
4560 ["LeftCeiling"] = 8968,  
4561 ["rceil"] = 8969,  
4562 ["RightCeiling"] = 8969,  
4563 ["lfloor"] = 8970,  
4564 ["LeftFloor"] = 8970,  
4565 ["rfloor"] = 8971,  
4566 ["RightFloor"] = 8971,  
4567 ["drcrop"] = 8972,  
4568 ["dlcrop"] = 8973,  
4569 ["urcrop"] = 8974,  
4570 ["ulcrop"] = 8975,  
4571 ["bnot"] = 8976,  
4572 ["proflines"] = 8978,  
4573 ["profsurf"] = 8979,  
4574 ["telrec"] = 8981,  
4575 ["target"] = 8982,  
4576 ["ulcorn"] = 8988,  
4577 ["ulcorner"] = 8988,  
4578 ["urcorn"] = 8989,  
4579 ["urcorner"] = 8989,  
4580 ["dlcorn"] = 8990,  
4581 ["llcorner"] = 8990,  
4582 ["drcorn"] = 8991,  
4583 ["lrcorn"] = 8991,  
4584 ["frown"] = 8994,  
4585 ["sfrown"] = 8994,  
4586 ["smile"] = 8995,  
4587 ["ssmile"] = 8995,  
4588 ["cylcty"] = 9005,  
4589 ["profalar"] = 9006,  
4590 ["topbot"] = 9014,  
4591 ["ovbar"] = 9021,  
4592 ["solbar"] = 9023,  
4593 ["angzarr"] = 9084,  
4594 ["lmoust"] = 9136,  
4595 ["lmoustache"] = 9136,  
4596 ["rmoust"] = 9137,  
4597 ["rmoustache"] = 9137,  
4598 ["tbrk"] = 9140,  
4599 ["OverBracket"] = 9140,  
4600 ["bbrk"] = 9141,  
4601 ["UnderBracket"] = 9141,



4602 ["bbrktbrk"] = 9142,  
4603 ["OverParenthesis"] = 9180,  
4604 ["UnderParenthesis"] = 9181,  
4605 ["OverBrace"] = 9182,  
4606 ["UnderBrace"] = 9183,  
4607 ["trpezium"] = 9186,  
4608 ["elinters"] = 9191,  
4609 ["blank"] = 9251,  
4610 ["oS"] = 9416,  
4611 ["circledS"] = 9416,  
4612 ["boxh"] = 9472,  
4613 ["HorizontalLine"] = 9472,  
4614 ["boxv"] = 9474,  
4615 ["boxdr"] = 9484,  
4616 ["boxdl"] = 9488,  
4617 ["boxur"] = 9492,  
4618 ["boxul"] = 9496,  
4619 ["boxvr"] = 9500,  
4620 ["boxvl"] = 9508,  
4621 ["boxhd"] = 9516,  
4622 ["boxhu"] = 9524,  
4623 ["boxvh"] = 9532,  
4624 ["boxH"] = 9552,  
4625 ["boxV"] = 9553,  
4626 ["boxdR"] = 9554,  
4627 ["boxDr"] = 9555,  
4628 ["boxDR"] = 9556,  
4629 ["boxdL"] = 9557,  
4630 ["boxDL"] = 9558,  
4631 ["boxDL"] = 9559,  
4632 ["boxuR"] = 9560,  
4633 ["boxUr"] = 9561,  
4634 ["boxUR"] = 9562,  
4635 ["boxuL"] = 9563,  
4636 ["boxUL"] = 9564,  
4637 ["boxUL"] = 9565,  
4638 ["boxvR"] = 9566,  
4639 ["boxVr"] = 9567,  
4640 ["boxVR"] = 9568,  
4641 ["boxvL"] = 9569,  
4642 ["boxVl"] = 9570,  
4643 ["boxVL"] = 9571,  
4644 ["boxHd"] = 9572,  
4645 ["boxhD"] = 9573,  
4646 ["boxHD"] = 9574,  
4647 ["boxHu"] = 9575,  
4648 ["boxhU"] = 9576,

4649 ["boxHU"] = 9577,  
4650 ["boxvH"] = 9578,  
4651 ["boxVh"] = 9579,  
4652 ["boxVH"] = 9580,  
4653 ["uhblk"] = 9600,  
4654 ["lhblk"] = 9604,  
4655 ["block"] = 9608,  
4656 ["blk14"] = 9617,  
4657 ["blk12"] = 9618,  
4658 ["blk34"] = 9619,  
4659 ["squ"] = 9633,  
4660 ["square"] = 9633,  
4661 ["Square"] = 9633,  
4662 ["squf"] = 9642,  
4663 ["suarf"] = 9642,  
4664 ["blacksquare"] = 9642,  
4665 ["FilledVerySmallSquare"] = 9642,  
4666 ["EmptyVerySmallSquare"] = 9643,  
4667 ["rect"] = 9645,  
4668 ["marker"] = 9646,  
4669 ["fltns"] = 9649,  
4670 ["xutri"] = 9651,  
4671 ["bigtriangleup"] = 9651,  
4672 ["utrif"] = 9652,  
4673 ["blacktriangle"] = 9652,  
4674 ["utri"] = 9653,  
4675 ["triangle"] = 9653,  
4676 ["rtrif"] = 9656,  
4677 ["blacktriangleright"] = 9656,  
4678 ["rtri"] = 9657,  
4679 ["triangleright"] = 9657,  
4680 ["xdtri"] = 9661,  
4681 ["bigtriangledown"] = 9661,  
4682 ["dtrif"] = 9662,  
4683 ["blacktriangledown"] = 9662,  
4684 ["dtri"] = 9663,  
4685 ["triangledown"] = 9663,  
4686 ["ltrif"] = 9666,  
4687 ["blacktriangleleft"] = 9666,  
4688 ["ltri"] = 9667,  
4689 ["triangleleft"] = 9667,  
4690 ["loz"] = 9674,  
4691 ["lozenge"] = 9674,  
4692 ["cir"] = 9675,  
4693 ["tridot"] = 9708,  
4694 ["xcirc"] = 9711,  
4695 ["bigcirc"] = 9711,

4696 ["ultri"] = 9720,  
4697 ["urtri"] = 9721,  
4698 ["lltri"] = 9722,  
4699 ["EmptySmallSquare"] = 9723,  
4700 ["FilledSmallSquare"] = 9724,  
4701 ["starf"] = 9733,  
4702 ["bigstar"] = 9733,  
4703 ["star"] = 9734,  
4704 ["phone"] = 9742,  
4705 ["female"] = 9792,  
4706 ["male"] = 9794,  
4707 ["spades"] = 9824,  
4708 ["spadesuit"] = 9824,  
4709 ["clubs"] = 9827,  
4710 ["clubsuit"] = 9827,  
4711 ["hearts"] = 9829,  
4712 ["heartsuit"] = 9829,  
4713 ["diams"] = 9830,  
4714 ["diamondsuit"] = 9830,  
4715 ["sung"] = 9834,  
4716 ["flat"] = 9837,  
4717 ["natur"] = 9838,  
4718 ["natural"] = 9838,  
4719 ["sharp"] = 9839,  
4720 ["check"] = 10003,  
4721 ["checkmark"] = 10003,  
4722 ["cross"] = 10007,  
4723 ["malt"] = 10016,  
4724 ["maltese"] = 10016,  
4725 ["sext"] = 10038,  
4726 ["VerticalSeparator"] = 10072,  
4727 ["lbrk"] = 10098,  
4728 ["rbrk"] = 10099,  
4729 ["lobrk"] = 10214,  
4730 ["LeftDoubleBracket"] = 10214,  
4731 ["robrk"] = 10215,  
4732 ["RightDoubleBracket"] = 10215,  
4733 ["lang"] = 10216,  
4734 ["LeftAngleBracket"] = 10216,  
4735 ["langle"] = 10216,  
4736 ["rang"] = 10217,  
4737 ["RightAngleBracket"] = 10217,  
4738 ["rangle"] = 10217,  
4739 ["Lang"] = 10218,  
4740 ["Rang"] = 10219,  
4741 ["loang"] = 10220,  
4742 ["roang"] = 10221,

4743 ["xlarr"] = 10229,  
 4744 ["longleftarrow"] = 10229,  
 4745 ["LongLeftArrow"] = 10229,  
 4746 ["xrarr"] = 10230,  
 4747 ["longrightarrow"] = 10230,  
 4748 ["LongRightArrow"] = 10230,  
 4749 ["xharr"] = 10231,  
 4750 ["longleftrightarrow"] = 10231,  
 4751 ["LongLeftRightArrow"] = 10231,  
 4752 ["xlArr"] = 10232,  
 4753 ["Longleftarrow"] = 10232,  
 4754 ["DoubleLongLeftArrow"] = 10232,  
 4755 ["xrArr"] = 10233,  
 4756 ["Longrightarrow"] = 10233,  
 4757 ["DoubleLongRightArrow"] = 10233,  
 4758 ["xhArr"] = 10234,  
 4759 ["Longleftrightarrow"] = 10234,  
 4760 ["DoubleLongLeftRightArrow"] = 10234,  
 4761 ["xmap"] = 10236,  
 4762 ["longmapsto"] = 10236,  
 4763 ["dzigrarr"] = 10239,  
 4764 ["nvlArr"] = 10498,  
 4765 ["nvrArr"] = 10499,  
 4766 ["nvHarr"] = 10500,  
 4767 ["Map"] = 10501,  
 4768 ["lbarr"] = 10508,  
 4769 ["rbarr"] = 10509,  
 4770 ["bkarow"] = 10509,  
 4771 ["lBarr"] = 10510,  
 4772 ["rBarr"] = 10511,  
 4773 ["dbkarow"] = 10511,  
 4774 ["RBarr"] = 10512,  
 4775 ["drbkarow"] = 10512,  
 4776 ["DDottrahd"] = 10513,  
 4777 ["UpArrowBar"] = 10514,  
 4778 ["DownArrowBar"] = 10515,  
 4779 ["Rarrtl"] = 10518,  
 4780 ["latail"] = 10521,  
 4781 ["ratail"] = 10522,  
 4782 ["lAtail"] = 10523,  
 4783 ["rAtail"] = 10524,  
 4784 ["larrfs"] = 10525,  
 4785 ["rarrfs"] = 10526,  
 4786 ["larrbfs"] = 10527,  
 4787 ["rarrbfs"] = 10528,  
 4788 ["nwarhk"] = 10531,  
 4789 ["nearhk"] = 10532,

4790 ["searhk"] = 10533,  
4791 ["hksearow"] = 10533,  
4792 ["swarhk"] = 10534,  
4793 ["hkswarow"] = 10534,  
4794 ["nwnear"] = 10535,  
4795 ["nesear"] = 10536,  
4796 ["toea"] = 10536,  
4797 ["seswar"] = 10537,  
4798 ["tosa"] = 10537,  
4799 ["swnwar"] = 10538,  
4800 ["rarrc"] = 10547,  
4801 ["cudarr"] = 10549,  
4802 ["ldca"] = 10550,  
4803 ["rdca"] = 10551,  
4804 ["cudarrr"] = 10552,  
4805 ["larrpl"] = 10553,  
4806 ["curarrm"] = 10556,  
4807 ["cularrp"] = 10557,  
4808 ["rarrpl"] = 10565,  
4809 ["harrcir"] = 10568,  
4810 ["Uarrocir"] = 10569,  
4811 ["lurdshar"] = 10570,  
4812 ["ldrushar"] = 10571,  
4813 ["LeftRightVector"] = 10574,  
4814 ["RightUpDownVector"] = 10575,  
4815 ["DownLeftRightVector"] = 10576,  
4816 ["LeftUpDownVector"] = 10577,  
4817 ["LeftVectorBar"] = 10578,  
4818 ["RightVectorBar"] = 10579,  
4819 ["RightUpVectorBar"] = 10580,  
4820 ["RightDownVectorBar"] = 10581,  
4821 ["DownLeftVectorBar"] = 10582,  
4822 ["DownRightVectorBar"] = 10583,  
4823 ["LeftUpVectorBar"] = 10584,  
4824 ["LeftDownVectorBar"] = 10585,  
4825 ["LeftTeeVector"] = 10586,  
4826 ["RightTeeVector"] = 10587,  
4827 ["RightUpTeeVector"] = 10588,  
4828 ["RightDownTeeVector"] = 10589,  
4829 ["DownLeftTeeVector"] = 10590,  
4830 ["DownRightTeeVector"] = 10591,  
4831 ["LeftUpTeeVector"] = 10592,  
4832 ["LeftDownTeeVector"] = 10593,  
4833 ["lHar"] = 10594,  
4834 ["uHar"] = 10595,  
4835 ["rHar"] = 10596,  
4836 ["dHar"] = 10597,

4837 ["luruhar"] = 10598,  
4838 ["ldrdhar"] = 10599,  
4839 ["ruluhar"] = 10600,  
4840 ["rdldhar"] = 10601,  
4841 ["lharul"] = 10602,  
4842 ["llhard"] = 10603,  
4843 ["rharul"] = 10604,  
4844 ["lrhard"] = 10605,  
4845 ["udhar"] = 10606,  
4846 ["UpEquilibrium"] = 10606,  
4847 ["duhar"] = 10607,  
4848 ["ReverseUpEquilibrium"] = 10607,  
4849 ["RoundImplies"] = 10608,  
4850 ["erarr"] = 10609,  
4851 ["simrarr"] = 10610,  
4852 ["larrsim"] = 10611,  
4853 ["rarrsim"] = 10612,  
4854 ["rarrap"] = 10613,  
4855 ["ltlarr"] = 10614,  
4856 ["gtrarr"] = 10616,  
4857 ["subrarr"] = 10617,  
4858 ["suplarr"] = 10619,  
4859 ["lfisht"] = 10620,  
4860 ["rfisht"] = 10621,  
4861 ["ufisht"] = 10622,  
4862 ["dfisht"] = 10623,  
4863 ["lopar"] = 10629,  
4864 ["ropar"] = 10630,  
4865 ["lbrke"] = 10635,  
4866 ["rbrke"] = 10636,  
4867 ["lbrkslu"] = 10637,  
4868 ["rbrksld"] = 10638,  
4869 ["lbrksld"] = 10639,  
4870 ["rbrkslu"] = 10640,  
4871 ["langd"] = 10641,  
4872 ["rangd"] = 10642,  
4873 ["lparlt"] = 10643,  
4874 ["rpargt"] = 10644,  
4875 ["gtlPar"] = 10645,  
4876 ["ltrPar"] = 10646,  
4877 ["vzigzag"] = 10650,  
4878 ["vangrt"] = 10652,  
4879 ["angrtvbd"] = 10653,  
4880 ["ange"] = 10660,  
4881 ["range"] = 10661,  
4882 ["dwangle"] = 10662,  
4883 ["uwangle"] = 10663,

4884 ["angmsdaa"] = 10664,  
 4885 ["angmsdab"] = 10665,  
 4886 ["angmsdac"] = 10666,  
 4887 ["angmsdad"] = 10667,  
 4888 ["angmsdae"] = 10668,  
 4889 ["angmsdaf"] = 10669,  
 4890 ["angmsdag"] = 10670,  
 4891 ["angmsdah"] = 10671,  
 4892 ["bemptyv"] = 10672,  
 4893 ["demptyv"] = 10673,  
 4894 ["cemptyv"] = 10674,  
 4895 ["raemptyv"] = 10675,  
 4896 ["laemptyv"] = 10676,  
 4897 ["ohbar"] = 10677,  
 4898 ["omid"] = 10678,  
 4899 ["opar"] = 10679,  
 4900 ["operp"] = 10681,  
 4901 ["olcross"] = 10683,  
 4902 ["odsold"] = 10684,  
 4903 ["olcir"] = 10686,  
 4904 ["ofcir"] = 10687,  
 4905 ["olt"] = 10688,  
 4906 ["ogt"] = 10689,  
 4907 ["cirscir"] = 10690,  
 4908 ["cirE"] = 10691,  
 4909 ["solb"] = 10692,  
 4910 ["bsolb"] = 10693,  
 4911 ["boxbox"] = 10697,  
 4912 ["trisb"] = 10701,  
 4913 ["rtriltri"] = 10702,  
 4914 ["LeftTriangleBar"] = 10703,  
 4915 ["RightTriangleBar"] = 10704,  
 4916 ["race"] = 10714,  
 4917 ["iinfin"] = 10716,  
 4918 ["infintie"] = 10717,  
 4919 ["nvinfin"] = 10718,  
 4920 ["eparsl"] = 10723,  
 4921 ["smeparsl"] = 10724,  
 4922 ["eqvparsl"] = 10725,  
 4923 ["lozf"] = 10731,  
 4924 ["blacklozenge"] = 10731,  
 4925 ["RuleDelayed"] = 10740,  
 4926 ["dsol"] = 10742,  
 4927 ["xodot"] = 10752,  
 4928 ["bigodot"] = 10752,  
 4929 ["xoplus"] = 10753,  
 4930 ["bigoplus"] = 10753,

4931 ["xotime"] = 10754,  
4932 ["bigotimes"] = 10754,  
4933 ["xuplus"] = 10756,  
4934 ["biguplus"] = 10756,  
4935 ["xsqcup"] = 10758,  
4936 ["bigsqcup"] = 10758,  
4937 ["qint"] = 10764,  
4938 ["iiiint"] = 10764,  
4939 ["fpartint"] = 10765,  
4940 ["cirfnint"] = 10768,  
4941 ["awint"] = 10769,  
4942 ["rppolint"] = 10770,  
4943 ["scpolint"] = 10771,  
4944 ["npolint"] = 10772,  
4945 ["pointint"] = 10773,  
4946 ["quatint"] = 10774,  
4947 ["intlarhk"] = 10775,  
4948 ["pluscir"] = 10786,  
4949 ["plusacir"] = 10787,  
4950 ["simplus"] = 10788,  
4951 ["plusdu"] = 10789,  
4952 ["plussim"] = 10790,  
4953 ["plustwo"] = 10791,  
4954 ["mcomma"] = 10793,  
4955 ["minusdu"] = 10794,  
4956 ["loplus"] = 10797,  
4957 ["roplus"] = 10798,  
4958 ["Cross"] = 10799,  
4959 ["timesd"] = 10800,  
4960 ["timesbar"] = 10801,  
4961 ["smashp"] = 10803,  
4962 ["lotimes"] = 10804,  
4963 ["rotimes"] = 10805,  
4964 ["otimesas"] = 10806,  
4965 ["Otimes"] = 10807,  
4966 ["odiv"] = 10808,  
4967 ["triplus"] = 10809,  
4968 ["triminus"] = 10810,  
4969 ["tritime"] = 10811,  
4970 ["iproduct"] = 10812,  
4971 ["intprod"] = 10812,  
4972 ["amalg"] = 10815,  
4973 ["capdot"] = 10816,  
4974 ["ncup"] = 10818,  
4975 ["ncap"] = 10819,  
4976 ["capand"] = 10820,  
4977 ["cupor"] = 10821,



4978 ["cupcap"] = 10822,  
 4979 ["capcup"] = 10823,  
 4980 ["cupbrcap"] = 10824,  
 4981 ["capbrcup"] = 10825,  
 4982 ["cupcup"] = 10826,  
 4983 ["capcap"] = 10827,  
 4984 ["ccups"] = 10828,  
 4985 ["ccaps"] = 10829,  
 4986 ["ccupssm"] = 10832,  
 4987 ["And"] = 10835,  
 4988 ["Or"] = 10836,  
 4989 ["andand"] = 10837,  
 4990 ["oror"] = 10838,  
 4991 ["orslope"] = 10839,  
 4992 ["andslope"] = 10840,  
 4993 ["andv"] = 10842,  
 4994 ["orv"] = 10843,  
 4995 ["andd"] = 10844,  
 4996 ["ord"] = 10845,  
 4997 ["wedbar"] = 10847,  
 4998 ["sdote"] = 10854,  
 4999 ["simdot"] = 10858,  
 5000 ["congdote"] = 10861,  
 5001 ["easter"] = 10862,  
 5002 ["apacir"] = 10863,  
 5003 ["apE"] = 10864,  
 5004 ["eplus"] = 10865,  
 5005 ["pluse"] = 10866,  
 5006 ["Esim"] = 10867,  
 5007 ["Colone"] = 10868,  
 5008 ["Equal"] = 10869,  
 5009 ["eDDot"] = 10871,  
 5010 ["ddotseq"] = 10871,  
 5011 ["equivDD"] = 10872,  
 5012 ["ltcir"] = 10873,  
 5013 ["gtcir"] = 10874,  
 5014 ["ltquest"] = 10875,  
 5015 ["gtquest"] = 10876,  
 5016 ["les"] = 10877,  
 5017 ["LessSlantEqual"] = 10877,  
 5018 ["leqslant"] = 10877,  
 5019 ["ges"] = 10878,  
 5020 ["GreaterSlantEqual"] = 10878,  
 5021 ["geqslant"] = 10878,  
 5022 ["lesdot"] = 10879,  
 5023 ["gesdot"] = 10880,  
 5024 ["lesdoto"] = 10881,

5025 ["gesdoto"] = 10882,  
5026 ["lesdotor"] = 10883,  
5027 ["gesdotol"] = 10884,  
5028 ["lap"] = 10885,  
5029 ["lessapprox"] = 10885,  
5030 ["gap"] = 10886,  
5031 ["gtrapprox"] = 10886,  
5032 ["lne"] = 10887,  
5033 ["lneq"] = 10887,  
5034 ["gne"] = 10888,  
5035 ["gneq"] = 10888,  
5036 ["lnap"] = 10889,  
5037 ["lnapprox"] = 10889,  
5038 ["gnap"] = 10890,  
5039 ["gnapprox"] = 10890,  
5040 ["lEg"] = 10891,  
5041 ["lesseqqgtr"] = 10891,  
5042 ["gE1"] = 10892,  
5043 ["gtreqqless"] = 10892,  
5044 ["lsime"] = 10893,  
5045 ["gsime"] = 10894,  
5046 ["lsimg"] = 10895,  
5047 ["gsiml"] = 10896,  
5048 ["lgE"] = 10897,  
5049 ["glE"] = 10898,  
5050 ["lesges"] = 10899,  
5051 ["gesles"] = 10900,  
5052 ["els"] = 10901,  
5053 ["eqslantless"] = 10901,  
5054 ["egs"] = 10902,  
5055 ["eqslantgtr"] = 10902,  
5056 ["elsdot"] = 10903,  
5057 ["egsdot"] = 10904,  
5058 ["el"] = 10905,  
5059 ["eg"] = 10906,  
5060 ["siml"] = 10909,  
5061 ["simg"] = 10910,  
5062 ["simlE"] = 10911,  
5063 ["simgE"] = 10912,  
5064 ["LessLess"] = 10913,  
5065 ["GreaterGreater"] = 10914,  
5066 ["glj"] = 10916,  
5067 ["gla"] = 10917,  
5068 ["ltcc"] = 10918,  
5069 ["gtcc"] = 10919,  
5070 ["lescc"] = 10920,  
5071 ["gescc"] = 10921,

5072 ["smt"] = 10922,  
5073 ["lat"] = 10923,  
5074 ["smte"] = 10924,  
5075 ["late"] = 10925,  
5076 ["bumpE"] = 10926,  
5077 ["pre"] = 10927,  
5078 ["preceq"] = 10927,  
5079 ["PrecedesEqual"] = 10927,  
5080 ["sce"] = 10928,  
5081 ["succeq"] = 10928,  
5082 ["SucceedsEqual"] = 10928,  
5083 ["prE"] = 10931,  
5084 ["scE"] = 10932,  
5085 ["prnE"] = 10933,  
5086 ["precneqq"] = 10933,  
5087 ["scnE"] = 10934,  
5088 ["succneqq"] = 10934,  
5089 ["prap"] = 10935,  
5090 ["precapprox"] = 10935,  
5091 ["scap"] = 10936,  
5092 ["succapprox"] = 10936,  
5093 ["prnap"] = 10937,  
5094 ["precnapprox"] = 10937,  
5095 ["scnap"] = 10938,  
5096 ["succnapprox"] = 10938,  
5097 ["Pr"] = 10939,  
5098 ["Sc"] = 10940,  
5099 ["subdot"] = 10941,  
5100 ["supdot"] = 10942,  
5101 ["subplus"] = 10943,  
5102 ["supplus"] = 10944,  
5103 ["submult"] = 10945,  
5104 ["supmult"] = 10946,  
5105 ["subedot"] = 10947,  
5106 ["supedot"] = 10948,  
5107 ["subE"] = 10949,  
5108 ["subseteqq"] = 10949,  
5109 ["supE"] = 10950,  
5110 ["supseteqq"] = 10950,  
5111 ["subsim"] = 10951,  
5112 ["supsim"] = 10952,  
5113 ["subnE"] = 10955,  
5114 ["subsetneqq"] = 10955,  
5115 ["supnE"] = 10956,  
5116 ["supsetneqq"] = 10956,  
5117 ["csub"] = 10959,  
5118 ["csup"] = 10960,

5119 ["csube"] = 10961,  
 5120 ["csupe"] = 10962,  
 5121 ["subsup"] = 10963,  
 5122 ["supsub"] = 10964,  
 5123 ["subsub"] = 10965,  
 5124 ["supsup"] = 10966,  
 5125 ["suphsub"] = 10967,  
 5126 ["supdsub"] = 10968,  
 5127 ["forkv"] = 10969,  
 5128 ["topfork"] = 10970,  
 5129 ["mlcp"] = 10971,  
 5130 ["Dashv"] = 10980,  
 5131 ["DoubleLeftTee"] = 10980,  
 5132 ["Vdashl"] = 10982,  
 5133 ["Barv"] = 10983,  
 5134 ["vBar"] = 10984,  
 5135 ["vBarv"] = 10985,  
 5136 ["Vbar"] = 10987,  
 5137 ["Not"] = 10988,  
 5138 ["bNot"] = 10989,  
 5139 ["rnmid"] = 10990,  
 5140 ["cirmid"] = 10991,  
 5141 ["midcir"] = 10992,  
 5142 ["topcir"] = 10993,  
 5143 ["nhpar"] = 10994,  
 5144 ["parsim"] = 10995,  
 5145 ["parsl"] = 11005,  
 5146 ["fflig"] = 64256,  
 5147 ["filig"] = 64257,  
 5148 ["fllig"] = 64258,  
 5149 ["ffilig"] = 64259,  
 5150 ["ffllig"] = 64260,  
 5151 ["Ascr"] = 119964,  
 5152 ["Cscr"] = 119966,  
 5153 ["Dscr"] = 119967,  
 5154 ["Gscr"] = 119970,  
 5155 ["Jscr"] = 119973,  
 5156 ["Kscr"] = 119974,  
 5157 ["Nscr"] = 119977,  
 5158 ["Oscr"] = 119978,  
 5159 ["Pscr"] = 119979,  
 5160 ["Qscr"] = 119980,  
 5161 ["Sscr"] = 119982,  
 5162 ["Tscr"] = 119983,  
 5163 ["Uscr"] = 119984,  
 5164 ["Vscr"] = 119985,  
 5165 ["Wscr"] = 119986,

5166 ["Xscr"] = 119987,  
5167 ["Yscr"] = 119988,  
5168 ["Zscr"] = 119989,  
5169 ["ascr"] = 119990,  
5170 ["bscr"] = 119991,  
5171 ["cscr"] = 119992,  
5172 ["dscr"] = 119993,  
5173 ["fscr"] = 119995,  
5174 ["hscr"] = 119997,  
5175 ["iscr"] = 119998,  
5176 ["jscr"] = 119999,  
5177 ["kscr"] = 120000,  
5178 ["lscr"] = 120001,  
5179 ["mscr"] = 120002,  
5180 ["nscr"] = 120003,  
5181 ["pscr"] = 120005,  
5182 ["qscr"] = 120006,  
5183 ["rscr"] = 120007,  
5184 ["sscr"] = 120008,  
5185 ["tscr"] = 120009,  
5186 ["uscr"] = 120010,  
5187 ["vscr"] = 120011,  
5188 ["wscr"] = 120012,  
5189 ["xscr"] = 120013,  
5190 ["yscr"] = 120014,  
5191 ["zscr"] = 120015,  
5192 ["Afr"] = 120068,  
5193 ["Bfr"] = 120069,  
5194 ["Dfr"] = 120071,  
5195 ["Efr"] = 120072,  
5196 ["Ffr"] = 120073,  
5197 ["Gfr"] = 120074,  
5198 ["Jfr"] = 120077,  
5199 ["Kfr"] = 120078,  
5200 ["Lfr"] = 120079,  
5201 ["Mfr"] = 120080,  
5202 ["Nfr"] = 120081,  
5203 ["Ofr"] = 120082,  
5204 ["Pfr"] = 120083,  
5205 ["Qfr"] = 120084,  
5206 ["Sfr"] = 120086,  
5207 ["Tfr"] = 120087,  
5208 ["Ufr"] = 120088,  
5209 ["Vfr"] = 120089,  
5210 ["Wfr"] = 120090,  
5211 ["Xfr"] = 120091,  
5212 ["Yfr"] = 120092,

5213 ["afr"] = 120094,  
5214 ["bfr"] = 120095,  
5215 ["cfr"] = 120096,  
5216 ["dfr"] = 120097,  
5217 ["efr"] = 120098,  
5218 ["ffr"] = 120099,  
5219 ["gfr"] = 120100,  
5220 ["hfr"] = 120101,  
5221 ["ifr"] = 120102,  
5222 ["jfr"] = 120103,  
5223 ["kfr"] = 120104,  
5224 ["lfr"] = 120105,  
5225 ["mfr"] = 120106,  
5226 ["nfr"] = 120107,  
5227 ["ofr"] = 120108,  
5228 ["pfr"] = 120109,  
5229 ["qfr"] = 120110,  
5230 ["rfr"] = 120111,  
5231 ["sfr"] = 120112,  
5232 ["tfr"] = 120113,  
5233 ["ufr"] = 120114,  
5234 ["vfr"] = 120115,  
5235 ["wfr"] = 120116,  
5236 ["xfr"] = 120117,  
5237 ["yfr"] = 120118,  
5238 ["zfr"] = 120119,  
5239 ["Aopf "] = 120120,  
5240 ["Bopf "] = 120121,  
5241 ["Dopf "] = 120123,  
5242 ["Eopf "] = 120124,  
5243 ["Fopf "] = 120125,  
5244 ["Gopf "] = 120126,  
5245 ["Iopf "] = 120128,  
5246 ["Jopf "] = 120129,  
5247 ["Kopf "] = 120130,  
5248 ["Lopf "] = 120131,  
5249 ["Mopf "] = 120132,  
5250 ["Oopf "] = 120134,  
5251 ["Sopf "] = 120138,  
5252 ["Topf "] = 120139,  
5253 ["Uopf "] = 120140,  
5254 ["Vopf "] = 120141,  
5255 ["Wopf "] = 120142,  
5256 ["Xopf "] = 120143,  
5257 ["Yopf "] = 120144,  
5258 ["aopf "] = 120146,  
5259 ["bopf "] = 120147,

```

5260 ["copf"] = 120148,
5261 ["dopf"] = 120149,
5262 ["eopf"] = 120150,
5263 ["fopf"] = 120151,
5264 ["gopf"] = 120152,
5265 ["hopf"] = 120153,
5266 ["iopf"] = 120154,
5267 ["jopf"] = 120155,
5268 ["kopf"] = 120156,
5269 ["lopf"] = 120157,
5270 ["mopf"] = 120158,
5271 ["nopf"] = 120159,
5272 ["oopf"] = 120160,
5273 ["popf"] = 120161,
5274 ["qopf"] = 120162,
5275 ["ropf"] = 120163,
5276 ["sopf"] = 120164,
5277 ["topf"] = 120165,
5278 ["uopf"] = 120166,
5279 ["vopf"] = 120167,
5280 ["wopf"] = 120168,
5281 ["xopf"] = 120169,
5282 ["yopf"] = 120170,
5283 ["zopf"] = 120171,
5284 }

```

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

```

5285 function entities.dec_entity(s)
5286 return unicode.utf8.char(tonumber(s))
5287 end

```

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

```

5288 function entities.hex_entity(s)
5289 return unicode.utf8.char(tonumber("0x"..s))
5290 end

```

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

```

5291 function entities.char_entity(s)
5292 local n = character_entities[s]
5293 if n == nil then
5294 return "&" .. s .. ";"
5295 end
5296 return unicode.utf8.char(n)
5297 end

```

### 3.1.3 Plain TeX Writer

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

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

```
5298 M.writer = {}
```

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

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

```
5299 function M.writer.new(options)
5300 local self = {}
```

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

```
5301 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.

```
5302 local slice_specifiers = {}
5303 for specifier in options.slice:gmatch("[^%s]+") do
5304 table.insert(slice_specifiers, specifier)
5305 end
5306
5307 if #slice_specifiers == 2 then
5308 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5309 local slice_begin_type = self.slice_begin:sub(1, 1)
5310 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5311 self.slice_begin = "^" .. self.slice_begin
5312 end
5313 local slice_end_type = self.slice_end:sub(1, 1)
5314 if slice_end_type ~= "^" and slice_end_type ~= "$" then
5315 self.slice_end = "$" .. self.slice_end
5316 end
5317 elseif #slice_specifiers == 1 then
5318 self.slice_begin = "^" .. slice_specifiers[1]
```



```

5319 self.slice_end = "$" .. slice_specifiers[1]
5320 end
5321
5322 self.slice_begin_type = self.slice_begin:sub(1, 1)
5323 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5324 self.slice_end_type = self.slice_end:sub(1, 1)
5325 self.slice_end_identifier = self.slice_end:sub(2) or ""
5326
5327 if self.slice_begin == "^" and self.slice_end ~= "^" then
5328 self.is_writing = true
5329 else
5330 self.is_writing = false
5331 end

```

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

```
5332 self.suffix = ".tex"
```

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

```
5333 self.space = " "
```

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

```
5334 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5335 function self.plain(s)
5336 return s
5337 end

```

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

```
5338 function self.paragraph(s)
5339 if not self.is_writing then return "" end
5340 return s
5341 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.

```
5342 function self.pack(name)
5343 return [[\input{]] .. name .. [}\relax]]
5344 end

```

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

```
5345 function self.interblocksep()
5346 if not self.is_writing then return "" end
5347 return "\\markdownRendererInterblockSeparator\n{}"
5348 end

```

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

```
5349 self.hard_line_break = "\\markdownRendererHardLineBreak\n{}"
```

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

```
5350 self.ellipsis = "\\markdownRendererEllipsis{}
```

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

```
5351 function self.thematic_break()
5352 if not self.is_writing then return "" end
5353 return "\\markdownRendererThematicBreak{}"
5354 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.

```
5355 self.escaped_uri_chars = {
5356 [{""] = "\\markdownRendererLeftBrace{]",
5357 ["}"] = "\\markdownRendererRightBrace{]",
5358 ["\\"] = "\\markdownRendererBackslash{]",
5359 }
5360 self.escaped_minimal_strings = {
5361 ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
5362 ["☒"] = "\\markdownRendererTickedBox{]",
5363 ["☐"] = "\\markdownRendererHalfTickedBox{]",
5364 ["□"] = "\\markdownRendererUntickedBox{]",
5365 [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{]",
5366 }
```

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

```
5367 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5368 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.

```
5369 self.escaped_chars = {
5370 [{""] = "\\markdownRendererLeftBrace{]",
5371 ["}"] = "\\markdownRendererRightBrace{]",
5372 ["%"] = "\\markdownRendererPercentSign{]",
5373 ["\\"] = "\\markdownRendererBackslash{]",
5374 ["#"] = "\\markdownRendererHash{]",
5375 ["$"] = "\\markdownRendererDollarSign{]",
5376 ["&"] = "\\markdownRendererAmpersand{]",
5377 ["_"] = "\\markdownRendererUnderscore{]",
5378 ["^"] = "\\markdownRendererCircumflex{]",
5379 ["~"] = "\\markdownRendererTilde{]",
5380 ["|"] = "\\markdownRendererPipe{]",
5381 [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{]",
5382 }
```

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

```
5383 local escape_typographic_text = util.escaper(
5384 self.escaped_chars, self.escaped_strings)
5385 local escape_programmatic_text = util.escaper(
5386 self.escaped_uri_chars, self.escaped_minimal_strings)
5387 local escape_minimal = util.escaper(
5388 {}, 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.

```
5389 self.escape = escape_typographic_text
5390 self.math = escape_minimal
5391 if options.hybrid then
5392 self.identifier = escape_minimal
5393 self.string = escape_minimal
5394 self.uri = escape_minimal
5395 else
5396 self.identifier = escape_programmatic_text
5397 self.string = escape_typographic_text
5398 self.uri = escape_programmatic_text
5399 end
```

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

```
5400 function self.code(s, attributes)
5401 local buf = {}
5402 if attributes ~= nil then
5403 table.insert(buf,
5404 "\\markdownRendererCodeSpanAttributeContextBegin\n")
5405 table.insert(buf, self.attributes(attributes))
5406 end
5407 table.insert(buf,
5408 {"\\markdownRendererCodeSpan{", self.escape(s), "}")})
5409 if attributes ~= nil then
5410 table.insert(buf,
5411 "\\markdownRendererCodeSpanAttributeContextEnd{")
5412 end
```

```

5413 return buf
5414 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.

```

5415 function self.link(lab, src, tit, attributes)
5416 local buf = {}
5417 if attributes ~= nil then
5418 table.insert(buf,
5419 "\\markdownRendererLinkAttributeContextBegin\n")
5420 table.insert(buf, self.attributes(attributes))
5421 end
5422 table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ",
5423 "{",self.escape(src),"}",
5424 "{",self.uri(src),"}",
5425 "{",self.string(tit or ""),"}"}))
5426 if attributes ~= nil then
5427 table.insert(buf,
5428 "\\markdownRendererLinkAttributeContextEnd{")
5429 end
5430 return buf
5431 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.

```

5432 function self.image(lab, src, tit, attributes)
5433 local buf = {}
5434 if attributes ~= nil then
5435 table.insert(buf,
5436 "\\markdownRendererImageAttributeContextBegin\n")
5437 table.insert(buf, self.attributes(attributes))
5438 end
5439 table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
5440 "{",self.string(src),"}",
5441 "{",self.uri(src),"}",
5442 "{",self.string(tit or ""),"}"}))
5443 if attributes ~= nil then
5444 table.insert(buf,
5445 "\\markdownRendererImageAttributeContextEnd{")
5446 end
5447 return buf
5448 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.

```

5449 function self.bulletlist(items,tight)
5450 if not self.is_writing then return "" end
5451 local buffer = {}
5452 for _,item in ipairs(items) do
5453 buffer[#buffer + 1] = self.bulletitem(item)
5454 end
5455 local contents = util.intersperse(buffer,"\n")
5456 if tight and options.tightLists then
5457 return {"\\markdownRendererUlBeginTight\n",contents,
5458 "\n\\markdownRendererUlEndTight "}
5459 else
5460 return {"\\markdownRendererUlBegin\n",contents,
5461 "\n\\markdownRendererUlEnd "}
5462 end
5463 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.

```

5464 function self.bulletitem(s)
5465 return {"\\markdownRendererUlItem ",s,
5466 "\\markdownRendererUlItemEnd "}
5467 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.

```

5468 function self.orderedlist(items,tight,startnum)
5469 if not self.is_writing then return "" end
5470 local buffer = {}
5471 local num = startnum
5472 for _,item in ipairs(items) do
5473 buffer[#buffer + 1] = self.ordereditem(item,num)
5474 if num ~= nil then
5475 num = num + 1
5476 end
5477 end
5478 local contents = util.intersperse(buffer,"\n")
5479 if tight and options.tightLists then
5480 return {"\\markdownRendererOlBeginTight\n",contents,
5481 "\n\\markdownRendererOlEndTight "}
5482 else
5483 return {"\\markdownRendererOlBegin\n",contents,
5484 "\n\\markdownRendererOlEnd "}
5485 end
5486 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.

```
5487 function self.ordereditem(s,num)
5488 if num ~= nil then
5489 return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
5490 "\\markdownRendererOlItemEnd "}
5491 else
5492 return {"\\markdownRendererOlItem ",s,
5493 "\\markdownRendererOlItemEnd "}
5494 end
5495 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.

```
5496 function self.inline_html_comment(contents)
5497 return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5498 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.

```
5499 function self.block_html_comment(contents)
5500 if not self.is_writing then return "" end
5501 return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
5502 "\n\\markdownRendererBlockHtmlCommentEnd "}
5503 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.

```
5504 function self.inline_html_tag(contents)
5505 return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5506 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.

```
5507 function self.block_html_element(s)
5508 if not self.is_writing then return "" end
5509 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5510 return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5511 end
```

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

```

5512 function self.emphasis(s)
5513 return {"\\markdownRendererEmphasis{" ,s,"}"}
5514 end

```

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

```

5515 function self.checkbox(f)
5516 if f == 1.0 then
5517 return "☒ "
5518 elseif f == 0.0 then
5519 return "☐ "
5520 else
5521 return "◻ "
5522 end
5523 end

```

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

```

5524 function self.strong(s)
5525 return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
5526 end

```

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

```

5527 function self.blockquote(s)
5528 if #util.rope_to_string(s) == 0 then return "" end
5529 return {"\\markdownRendererBlockQuoteBegin\n",s,
5530 "\n\\markdownRendererBlockQuoteEnd "}
5531 end

```

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

```

5532 function self.verbatim(s)
5533 if not self.is_writing then return "" end
5534 s = s:gsub("\\n$", "")
5535 local name = util.cache_verbatim(options.cacheDir, s)
5536 return {"\\markdownRendererInputVerbatim{" ,name,"}"}
5537 end

```

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

```

5538 function self.document(d)
5539 local buf = {"\\markdownRendererDocumentBegin\n", d}
5540
5541 -- pop all attributes
5542 table.insert(buf, self.pop_attributes())
5543
5544 table.insert(buf, "\\markdownRendererDocumentEnd")

```

```

5545
5546 return buf
5547 end

```

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

```

5548 function self.attributes(attributes)
5549 local expanded_attributes = {}
5550 local key_value_regex = "([^\s=]+)%s*=%s*(.*)"
5551 local key, value
5552 for _, attribute in ipairs(attributes) do
5553 if attribute:sub(1, 1) == "#" or attribute:sub(1, 1) == "." then
5554 table.insert(expanded_attributes, attribute)
5555 else
5556 key, value = attribute:match(key_value_regex)
5557 if key:lower() == "id" then
5558 table.insert(expanded_attributes, "#" .. value)
5559 elseif key:lower() == "class" then
5560 local classes = {}
5561 for class in value:gmatch("%S+") do
5562 table.insert(classes, class)
5563 end
5564 table.sort(classes)
5565 for _, class in ipairs(classes) do
5566 table.insert(expanded_attributes, "." .. class)
5567 end
5568 else
5569 table.insert(expanded_attributes, attribute)
5570 end
5571 end
5572 end
5573 table.sort(expanded_attributes)
5574
5575 local buf = {}
5576 local seen = {}
5577 for _, attribute in ipairs(expanded_attributes) do
5578 if seen[attribute] ~= nil then
5579 goto continue -- prevent duplicate attributes
5580 else
5581 seen[attribute] = true
5582 end
5583 if attribute:sub(1, 1) == "#" then
5584 table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
5585 attribute:sub(2), "}"})
5586 elseif attribute:sub(1, 1) == "." then
5587 table.insert(buf, {"\\markdownRendererAttributeName{" ,
5588 attribute:sub(2), "}"})

```



```

5589 else
5590 key, value = attribute:match(key_value_regex)
5591 table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5592 key, "}{" , value, "}"}))
5593 end
5594 ::continue::
5595 end
5596
5597 return buf
5598 end

```

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

```

5599 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`.

```

5600 local function apply_attributes()
5601 local buf = {}
5602 for i = 1, #self.active_attributes do
5603 local start_output = self.active_attributes[i][3]
5604 if start_output ~= nil then
5605 table.insert(buf, start_output)
5606 end
5607 end
5608 return buf
5609 end
5610
5611 local function tear_down_attributes()
5612 local buf = {}
5613 for i = #self.active_attributes, 1, -1 do
5614 local end_output = self.active_attributes[i][4]
5615 if end_output ~= nil then
5616 table.insert(buf, end_output)
5617 end
5618 end
5619 return buf
5620 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.

```

5621 function self.push_attributes(attribute_type, attributes,
5622 start_output, end_output)

```

```

5623 -- index attributes in a hash table for easy lookup
5624 attributes = attributes or {}
5625 for i = 1, #attributes do
5626 attributes[attributes[i]] = true
5627 end
5628
5629 local buf = {}
5630 -- handle slicing
5631 if attributes["#" .. self.slice_end_identifrier] ~= nil and
5632 self.slice_end_type == "^" then
5633 if self.is_writing then
5634 table.insert(buf, tear_down_attributes())
5635 end
5636 self.is_writing = false
5637 end
5638 if attributes["#" .. self.slice_begin_identifrier] ~= nil and
5639 self.slice_begin_type == "^" then
5640 self.is_writing = true
5641 table.insert(buf, apply_attributes())
5642 self.is_writing = true
5643 end
5644 if self.is_writing and start_output ~= nil then
5645 table.insert(buf, start_output)
5646 end
5647 table.insert(self.active_attributes,
5648 {attribute_type, attributes,
5649 start_output, end_output})
5650 return buf
5651 end
5652

```

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`).

```

5653 function self.pop_attributes(attribute_type)
5654 local buf = {}
5655 -- pop attributes until we find attributes of correct type
5656 -- or until no attributes remain
5657 local current_attribute_type = false
5658 while current_attribute_type ~= attribute_type and
5659 #self.active_attributes > 0 do
5660 local attributes, _, end_output
5661 current_attribute_type, attributes, _, end_output = table.unpack(
5662 self.active_attributes[#self.active_attributes])

```

```

5663 if self.is_writing and end_output ~= nil then
5664 table.insert(buf, end_output)
5665 end
5666 table.remove(self.active_attributes, #self.active_attributes)
5667 -- handle slicing
5668 if attributes["#" .. self.slice_end_identifier] ~= nil
5669 and self.slice_end_type == "$" then
5670 if self.is_writing then
5671 table.insert(buf, tear_down_attributes())
5672 end
5673 self.is_writing = false
5674 end
5675 if attributes["#" .. self.slice_begin_identifier] ~= nil and
5676 self.slice_begin_type == "$" then
5677 self.is_writing = true
5678 table.insert(buf, apply_attributes())
5679 end
5680 end
5681 return buf
5682 end

```

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

```

5683 local current_heading_level = 0
5684 function self.heading(s, level, attributes)
5685 local buf = {}
5686
5687 -- push empty attributes for implied sections
5688 while current_heading_level < level - 1 do
5689 table.insert(buf,
5690 self.push_attributes("heading",
5691 nil,
5692 "\\markdownRendererSectionBegin\n",
5693 "\n\\markdownRendererSectionEnd ")
5694)
5695 current_heading_level = current_heading_level + 1
5696 end
5697
5698 -- pop attributes for sections that have ended
5699 while current_heading_level >= level do
5700 table.insert(buf, self.pop_attributes("heading"))
5701 current_heading_level = current_heading_level - 1
5702 end
5703
5704 -- push attributes for the new section
5705 local start_output = {}
5706 local end_output = {}
5707 table.insert(start_output, "\\markdownRendererSectionBegin\n")

```

```

5707 if options.headerAttributes and attributes ~= nil and #attributes > 0 then
5708 table.insert(start_output,
5709 "\\markdownRendererHeaderAttributeContextBegin\n")
5710 table.insert(start_output, self.attributes(attributes))
5711 table.insert(end_output,
5712 "\n\\markdownRendererHeaderAttributeContextEnd ")
5713 end
5714 table.insert(end_output, "\n\\markdownRendererSectionEnd ")
5715
5716 table.insert(buf, self.push_attributes("heading",
5717 attributes,
5718 start_output,
5719 end_output))
5720 current_heading_level = current_heading_level + 1
5721 assert(current_heading_level == level)
5722
5723 -- produce the renderer
5724 local cmd
5725 level = level + options.shiftHeadings
5726 if level <= 1 then
5727 cmd = "\\markdownRendererHeadingOne"
5728 elseif level == 2 then
5729 cmd = "\\markdownRendererHeadingTwo"
5730 elseif level == 3 then
5731 cmd = "\\markdownRendererHeadingThree"
5732 elseif level == 4 then
5733 cmd = "\\markdownRendererHeadingFour"
5734 elseif level == 5 then
5735 cmd = "\\markdownRendererHeadingFive"
5736 elseif level >= 6 then
5737 cmd = "\\markdownRendererHeadingSix"
5738 else
5739 cmd = ""
5740 end
5741 if self.is_writing then
5742 table.insert(buf, {cmd, "{", s, "}"})
5743 end
5744
5745 return buf
5746 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.

```

5747 function self.get_state()
5748 return {
5749 is_writing=self.is_writing,
5750 active_attributes={table.unpack(self.active_attributes)},

```

```

5751 }
5752 end

```

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

```

5753 function self.set_state(s)
5754 local previous_state = self.get_state()
5755 for key, value in pairs(s) do
5756 self[key] = value
5757 end
5758 return previous_state
5759 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`.

```

5760 function self.defer_call(f)
5761 local previous_state = self.get_state()
5762 return function(...)
5763 local state = self.set_state(previous_state)
5764 local return_value = f(...)
5765 self.set_state(state)
5766 return return_value
5767 end
5768 end
5769
5770 return self
5771 end

```

### 3.1.4 Parsers

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

```

5772 local parsers = {}

```

#### 3.1.4.1 Basic Parsers

```

5773 parsers.percent = P("%")
5774 parsers.at = P("@")
5775 parsers.comma = P(",")
5776 parsers.asterisk = P("*")
5777 parsers.dash = P("-")
5778 parsers.plus = P("+")
5779 parsers.underscore = P("_")
5780 parsers.period = P(".")
5781 parsers.hash = P("#")
5782 parsers.dollar = P("$")

```

```

5783 parsers.ampersand = P("&")
5784 parsers.backtick = P("`")
5785 parsers.less = P("<")
5786 parsers.more = P(">")
5787 parsers.space = P(" ")
5788 parsers.squote = P("'")
5789 parsers.dquote = P('"')
5790 parsers.lparent = P("(")
5791 parsers.rparent = P(")")
5792 parsers.lbracket = P("[")
5793 parsers.rbracket = P("]")
5794 parsers.lbrace = P("{")
5795 parsers.rbrace = P("}")
5796 parsers.circumflex = P("^")
5797 parsers.slash = P("/")
5798 parsers.equal = P("=")
5799 parsers.colon = P(":")
5800 parsers.semicolon = P(";")
5801 parsers.exclamation = P("!")
5802 parsers.pipe = P("|")
5803 parsers.tilde = P("~")
5804 parsers.backslash = P("\\")
5805 parsers.tab = P("\t")
5806 parsers.newline = P("\n")
5807 parsers.tightblocksep = P("\001")
5808
5809 parsers.digit = R("09")
5810 parsers.hexdigit = R("09", "af", "AF")
5811 parsers.letter = R("AZ", "az")
5812 parsers.alphanumeric = R("AZ", "az", "09")
5813 parsers.keyword = parsers.letter
5814 * parsers.alphanumeric^0
5815 parsers.internal_punctuation = S(":,.,.?")
5816
5817 parsers.doubleasterisks = P("**")
5818 parsers.doubleunderscores = P("__")
5819 parsers.doubletildes = P("~~")
5820 parsers.fourspace = P(" ")
5821
5822 parsers.any = P(1)
5823 parsers.succeed = P(true)
5824 parsers.fail = P(false)
5825
5826 parsers.escapable = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
5827 parsers.anyescaped = parsers.backslash / " " * parsers.escapable
5828 + parsers.any
5829

```

```

5830 parsers.spacechar = S("\t ")
5831 parsers.spacing = S(" \n\r\t")
5832 parsers.nospacechar = parsers.any - parsers.spacing
5833 parsers.optionalspace = parsers.spacechar^0
5834
5835 parsers.normalchar = parsers.any - (V("SpecialChar")
5836 + parsers.spacing
5837 + parsers.tightblocksep)
5838 parsers.eof = -parsers.any
5839 parsers.nonindentspace = parsers.space^-3 * - parsers.spacechar
5840 parsers.indent = parsers.space^-3 * parsers.tab
5841 + parsers.fourspaces / ""
5842 parsers.linechar = P(1 - parsers.newline)
5843
5844 parsers.blankline = parsers.optionalspace
5845 * parsers.newline / "\n"
5846 parsers.blanklines = parsers.blankline^0
5847 parsers.skipblanklines = (parsers.optionalspace * parsers.newline)^0
5848 parsers.indentedline = parsers.indent / ""
5849 * C(parsers.linechar^1 * parsers.newline^-
5850 1)
5851 parsers.optionallyindentedline = parsers.indent^-1 / ""
5852 * C(parsers.linechar^1 * parsers.newline^-
5853 1)
5854 parsers.sp = parsers.spacing^0
5855 parsers.spnl = parsers.optionalspace
5856 * (parsers.newline * parsers.optionalspace)^-
5857 1
5858 parsers.line = parsers.linechar^0 * parsers.newline
5859 parsers.nonemptyline = parsers.line - parsers.blankline

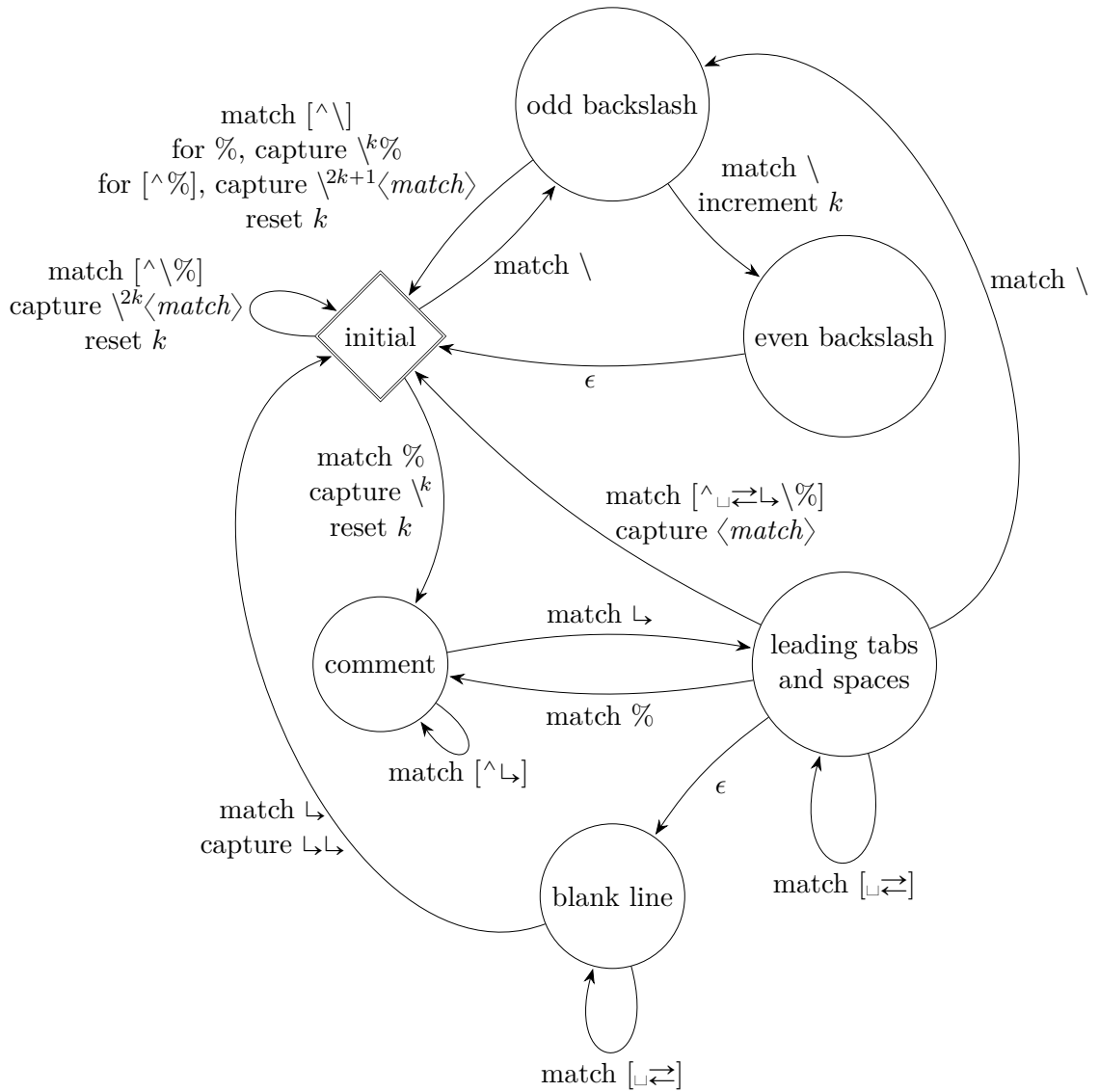
```

The `parsers.commented_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 6.

```

5857 parsers.commented_line_letter = parsers.linechar
5858 + parsers.newline
5859 - parsers.backslash
5860 - parsers.percent
5861 parsers.commented_line = Cg(Cc(""), "backslashes")
5862 * ((#(parsers.commented_line_letter
5863 - parsers.newline)
5864 * Cb("backslashes")
5865 * Cs(parsers.commented_line_letter
5866 - parsers.newline)^1 -- initial
5867 * Cg(Cc(""), "backslashes"))
5868 + #(parsers.backslash * parsers.backslash)
5869 * Cg((parsers.backslash -- even backslash
5870 * parsers.backslash)^1, "backslashes")

```



**Figure 6: A pushdown automaton that recognizes TeX comments**



```

5871 + (parsers.backslash
5872 * (#parsers.percent
5873 * Cb("backslashes")
5874 / function(backslashes)
5875 return string.rep("\\", #backslashes / 2)
5876 end
5877 * C(parsers.percent)
5878 + #parsers.commented_line_letter
5879 * Cb("backslashes")
5880 * Cc("\\")
5881 * C(parsers.commented_line_letter))
5882 * Cg(Cc(""), "backslashes"))^0
5883 * (#parsers.percent
5884 * Cb("backslashes")
5885 / function(backslashes)
5886 return string.rep("\\", #backslashes / 2)
5887 end
5888 * ((parsers.percent -- comment
5889 * parsers.line
5890 * #parsers.blankline) -- blank line
5891 / "\n"
5892 + parsers.percent -- comment
5893 * parsers.line
5894 * parsers.optionalspace) -- leading tabs and space
5895 + #parsers.newline)
5896 * Cb("backslashes")
5897 * C(parsers.newline))
5898
5899 parsers.chunk = parsers.line * (parsers.optionallyindentedline
5900 - parsers.blankline)^0
5901
5902 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
5903 parsers.attribute_key = (parsers.attribute_key_char
5904 - parsers.dash - parsers.digit)
5905 * parsers.attribute_key_char^0
5906 parsers.attribute_value = ((parsers.dquote / "")
5907 * (parsers.anyescaped - parsers.dquote)^0
5908 * (parsers.dquote / ""))
5909 + ((parsers.squote / "")
5910 * (parsers.anyescaped - parsers.squote)^0
5911 * (parsers.squote / ""))
5912 + (parsers.anyescaped - parsers.dquote - parsers.rbra
5913 - parsers.space)^0
5914
5915 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
5916 + C((parsers.hash + parsers.period)
5917 * parsers.attribute_key)

```

```

5918 + Cs(parsers.attribute_key
5919 * parsers.optionalspace * parsers.equal * parsers.optionalspace
5920 * parsers.attribute_value)
5921 parsers.attributes = parsers.lbrace
5922 * parsers.optionalspace
5923 * parsers.attribute
5924 * (parsers.spacechar^1
5925 * parsers.attribute)^0
5926 * parsers.optionalspace
5927 * parsers.rbrace
5928
5929
5930 parsers.raw_attribute = parsers.lbrace
5931 * parsers.optionalspace
5932 * parsers.equal
5933 * C(parsers.attribute_key)
5934 * parsers.optionalspace
5935 * parsers.rbrace
5936
5937 -- block followed by 0 or more optionally
5938 -- indented blocks with first line indented.
5939 parsers.indented_blocks = function(bl)
5940 return Cs(bl
5941 * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
5942 * (parsers.blankline^1 + parsers.eof))
5943 end

```

### 3.1.4.2 Parsers Used for Markdown Lists

```

5944 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
5945
5946 parsers.bullet = (parsers.bulletchar * #parsers.spacing
5947 * (parsers.tab + parsers.space^-
5948 3)
5949 + parsers.space * parsers.bulletchar * #parsers.spacing
5950 * (parsers.tab + parsers.space^-2)
5951 + parsers.space * parsers.space * parsers.bulletchar
5952 * #parsers.spacing
5953 * (parsers.tab + parsers.space^-1)
5954 + parsers.space * parsers.space * parsers.space
5955 * parsers.bulletchar * #parsers.spacing
5956)
5957 local function tickbox(interior)
5958 return parsers.optionalspace * parsers.lbracket
5959 * interior * parsers.rbracket * parsers.spacechar^1
5960 end

```

```

5961
5962 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
5963 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
5964 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
5965

```

### 3.1.4.3 Parsers Used for Markdown Code Spans

```

5966 parsers.openticks = Cg(parsers.backtick^1, "ticks")
5967
5968 local function captures_equal_length(_,i,a,b)
5969 return #a == #b and i
5970 end
5971
5972 parsers.closeticks = parsers.space^-1
5973 * Cmt(C(parsers.backtick^1)
5974 * Cb("ticks"), captures_equal_length)
5975
5976 parsers.intickschar = (parsers.any - S("\n\r`"))
5977 + (parsers.newline * -parsers.blankline)
5978 + (parsers.space - parsers.closeticks)
5979 + (parsers.backtick^1 - parsers.closeticks)
5980
5981 parsers.inticks = parsers.openticks * parsers.space^-1
5982 * C(parsers.intickschar^0) * parsers.closeticks

```

### 3.1.4.4 Parsers Used for Markdown Tags and Links

```

5983 parsers.leader = parsers.space^-3
5984
5985 -- content in balanced brackets, parentheses, or quotes:
5986 parsers.bracketed = P{ parsers.lbracket
5987 * ((parsers.backslash / "\"" * parsers.rbracket
5988 + parsers.any - (parsers.lbracket
5989 + parsers.rbracket
5990 + parsers.blankline^2)
5991) + V(1))^0
5992 * parsers.rbracket }
5993
5994 parsers.inparens = P{ parsers.lparent
5995 * ((parsers.anyescaped - (parsers.lparent
5996 + parsers.rparent
5997 + parsers.blankline^2)
5998) + V(1))^0
5999 * parsers.rparent }
6000
6001 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
6002 * ((parsers.anyescaped - (parsers.squote

```

```

6003 + parsers.blankline^2)
6004) + V(1))^0
6005 * parsers.squote }
6006
6007 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
6008 * ((parsers.anyescaped - (parsers.dquote
6009 + parsers.blankline^2)
6010) + V(1))^0
6011 * parsers.dquote }
6012
6013 -- bracketed tag for markdown links, allowing nested brackets:
6014 parsers.tag = parsers.lbracket
6015 * Cs((parsers.alphanumeric^1
6016 + parsers.bracketed
6017 + parsers.inticks
6018 + (parsers.backslash / "" * parsers.rbracket
6019 + parsers.any
6020 - (parsers.rbracket + parsers.blankline^2)))^0)
6021 * parsers.rbracket
6022
6023 -- url for markdown links, allowing nested brackets:
6024 parsers.url = parsers.less * Cs((parsers.anyescaped
6025 - parsers.more)^0)
6026 * parsers.more
6027 + Cs((parsers.inparens + (parsers.anyescaped
6028 - parsers.spacing
6029 - parsers.rparent))^1)
6030
6031 -- quoted text, possibly with nested quotes:
6032 parsers.title_s = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
6033 + parsers.squoted)^0)
6034 * parsers.squote
6035
6036 parsers.title_d = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
6037 + parsers.dquoted)^0)
6038 * parsers.dquote
6039
6040 parsers.title_p = parsers.lparent
6041 * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
6042 * parsers.rparent
6043
6044 parsers.title = parsers.title_d + parsers.title_s + parsers.title_p
6045
6046 parsers.optionaltitle
6047 = parsers.spnl * parsers.title * parsers.spacechar^0
6048 + Cc("")
6049

```

```

6050 parsers.indirect_link
6051 = parsers.tag
6052 * (C(parsers.spnl) * parsers.tag
6053 + Cc(nil) * Cc(nil) -- always produce exactly two captures
6054)
6055
6056 parsers.indirect_image
6057 = parsers.exclamation * parsers.indirect_link

```

### 3.1.4.5 Parsers Used for HTML

```

6058 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
6059 parsers.keyword_exact = function(s)
6060 local parser = P(0)
6061 for i=1,#s do
6062 local c = s:sub(i,i)
6063 local m = c .. upper(c)
6064 parser = parser * S(m)
6065 end
6066 return parser
6067 end
6068
6069 parsers.block_keyword =
6070 parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
6071 parsers.keyword_exact("center") + parsers.keyword_exact("del") +
6072 parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
6073 parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
6074 parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
6075 parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
6076 parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
6077 parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
6078 parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
6079 parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
6080 parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
6081 parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
6082 parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
6083 parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
6084 parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
6085 parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
6086 parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
6087 parsers.keyword_exact("td") + parsers.keyword_exact("tr")
6088
6089 -- There is no reason to support bad html, so we expect quoted attributes
6090 parsers.htmlattributevalue
6091 = parsers.squote * (parsers.any - (parsers.blankline
6092 + parsers.squote))^0
6093 * parsers.squote

```

```

6094 + parsers.dquote * (parsers.any - (parsers.blankline
6095 + parsers.dquote))^0
6096 * parsers.dquote
6097
6098 parsers.htmlattribute = parsers.spacing^1
6099 * (parsers.alphanumeric + S("_-"))^1
6100 * parsers.sp * parsers.equal * parsers.sp
6101 * parsers.htmlattributevalue
6102
6103 parsers.htmlcomment = P("<!--")
6104 * parsers.optionalspace
6105 * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
6106 * parsers.optionalspace
6107 * P("-->")
6108
6109 parsers.htmlinstruction = P("<?") * (parsers.any - P("?>"))^0 * P("?>")
6110
6111 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
6112 * parsers.sp * parsers.more
6113
6114 parsers.openelt_exact = function(s)
6115 return parsers.less * parsers.sp * parsers.keyword_exact(s)
6116 * parsers.htmlattribute^0 * parsers.sp * parsers.more
6117 end
6118
6119 parsers.openelt_block = parsers.sp * parsers.block_keyword
6120 * parsers.htmlattribute^0 * parsers.sp * parsers.more
6121
6122 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
6123 * parsers.keyword * parsers.sp * parsers.more
6124
6125 parsers.closeelt_exact = function(s)
6126 return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
6127 * parsers.sp * parsers.more
6128 end
6129
6130 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
6131 * parsers.htmlattribute^0 * parsers.sp * parsers.slash
6132 * parsers.more
6133
6134 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
6135 * parsers.htmlattribute^0 * parsers.sp * parsers.slash
6136 * parsers.more
6137
6138 parsers.displaytext = (parsers.any - parsers.less)^1
6139
6140 -- return content between two matched HTML tags

```

```

6141 parsers.in_matched = function(s)
6142 return { parsers.openelt_exact(s)
6143 * (V(1) + parsers.displaytext
6144 + (parsers.less - parsers.closeelt_exact(s))^0
6145 * parsers.closeelt_exact(s) }
6146 end
6147
6148 local function parse_matched_tags(s,pos)
6149 local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
6150 return lpeg.match(parsers.in_matched(t),s,pos-1)
6151 end
6152
6153 parsers.in_matched_block_tags = parsers.less
6154 * Cmt(#parsers.openelt_block, parse_matched_tags)
6155

```

### 3.1.4.6 Parsers Used for HTML Entities

```

6156 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
6157 * C(parsers.hexdigit^1) * parsers.semicolon
6158 parsers.decentity = parsers.ampersand * parsers.hash
6159 * C(parsers.digit^1) * parsers.semicolon
6160 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
6161 * parsers.semicolon

```

### 3.1.4.7 Helpers for Link Reference Definitions

```

6162 -- parse a reference definition: [foo]: /bar "title"
6163 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
6164 * parsers.spacechar^0 * parsers.url
6165 * parsers.optionaltitle

```

### 3.1.4.8 Inline Elements

```

6166 parsers.Inline = V("Inline")
6167 parsers.IndentedInline = V("IndentedInline")
6168
6169 -- parse many p between starter and ender
6170 parsers.between = function(p, starter, ender)
6171 local ender2 = B(parsers.nonspacechar) * ender
6172 return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
6173 end
6174
6175 parsers.urlchar = parsers.anyescaped
6176 - parsers.newline
6177 - parsers.more
6178
6179 parsers.auto_link_url = parsers.less

```

```

6180 * C(parsers.alphanumeric^1 * P("://")
6181 * parsers.urlchar^1)
6182 * parsers.more
6183
6184 parsers.auto_link_email
6185 = parsers.less
6186 * C((parsers.alphanumeric + S("-._+"))^1
6187 * P("@") * parsers.urlchar^1)
6188 * parsers.more
6189
6190 parsers.auto_link_relative_reference
6191 = parsers.less
6192 * C(parsers.urlchar^1)
6193 * parsers.more
6194

```

### 3.1.4.9 Block Elements

```

6195 parsers.lineof = function(c)
6196 return (parsers.leader * (P(c) * parsers.optionalspace)^3
6197 * (parsers.newline * parsers.blankline^1
6198 + parsers.newline^-1 * parsers.eof))
6199 end

```

### 3.1.4.10 Headings

```

6200 -- parse Atx heading start and return level
6201 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
6202 * -parsers.hash / length
6203
6204 -- parse setext header ending and return level
6205 parsers.heading_level = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
6206
6207 local function strip_atx_end(s)
6208 return s:gsub("#%s*\n$", "")
6209 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`.

```
6210 M.reader = {}
6211 function M.reader.new(writer, options)
6212 local self = {}
```

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

```
6213 self.writer = writer
6214 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.

```
6215 self.parsers = {}
6216 (function(parsers)
6217 setmetatable(self.parsers, {
6218 __index = function (_, key)
6219 return parsers[key]
6220 end
6221 })
6222 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.

```
6223 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.

```
6224 function self.normalize_tag(tag)
6225 tag = util.ropetostring(tag)
6226 tag = tag:gsub("[\n\r\t]+", " ")
6227 tag = tag:gsub("^ ", ""):gsub(" $", "")
6228 tag = uni_case.casefold(tag, true, false)
6229 return tag
6230 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.

```
6231 local function iterlines(s, f)
6232 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
6233 return util.ropetostring(rope)
6234 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.

```
6235 if options.preserveTabs then
6236 self.expandtabs = function(s) return s end
6237 else
6238 self.expandtabs = function(s)
6239 if s:find("\t") then
6240 return iterlines(s, util.expand_tabs_in_line)
6241 else
6242 return s
6243 end
6244 end
6245 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.

```
6246 self.parser_functions = {}
6247 self.create_parser = function(name, grammar, toplevel)
6248 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.

```
6249 if toplevel and options.stripIndent then
6250 local min_prefix_length, min_prefix = nil, ''
6251 str = iterlines(str, function(line)
6252 if lpeg.match(parsers.nonemptyline, line) == nil then
6253 return line
6254 end
6255 line = util.expand_tabs_in_line(line)
6256 local prefix = lpeg.match(C(parsers.optionalspace), line)
6257 local prefix_length = #prefix
6258 local is_shorter = min_prefix_length == nil
6259 is_shorter = is_shorter or prefix_length < min_prefix_length
6260 if is_shorter then
6261 min_prefix_length, min_prefix = prefix_length, prefix
6262 end
6263 return line
6264 end)
6265 str = str:gsub('^' .. min_prefix, '')
6266 end
```

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

```
6267 if toplevel and (options.texComments or options.hybrid) then
6268 str = lpeg.match(Ct(parsers.commented_line^1), str)
6269 str = util.ropo_to_string(str)
6270 end
6271 local res = lpeg.match(grammar(), str)
6272 if res == nil then
6273 error(format("%s failed on:\n%s", name, str:sub(1,20)))
6274 else
6275 return res
6276 end
6277 end
6278 end
6279
6280 self.create_parser("parse_blocks",
6281 function()
6282 return parsers.blocks
6283 end, true)
6284
6285 self.create_parser("parse_blocks_nested",
6286 function()
6287 return parsers.blocks_nested
6288 end, false)
6289
6290 self.create_parser("parse_inlines",
6291 function()
6292 return parsers.inlines
6293 end, false)
6294
6295 self.create_parser("parse_inlines_no_link",
6296 function()
6297 return parsers.inlines_no_link
6298 end, false)
6299
6300 self.create_parser("parse_inlines_no_inline_note",
6301 function()
6302 return parsers.inlines_no_inline_note
6303 end, false)
6304
6305 self.create_parser("parse_inlines_no_html",
6306 function()
6307 return parsers.inlines_no_html
6308 end, false)
6309
6310 self.create_parser("parse_inlines_nbsp",
```

```

6311 function()
6312 return parsers.inlines_nbsp
6313 end, false)

```

### 3.1.5.3 Parsers Used for Markdown Lists (local)

```

6314 if options.hashEnumerators then
6315 parsers.dig = parsers.digit + parsers.hash
6316 else
6317 parsers.dig = parsers.digit
6318 end
6319
6320 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
6321 + C(parsers.dig^2 * parsers.period) * #parsers.spacing
6322 * (parsers.tab + parsers.space^1)
6323 + C(parsers.dig * parsers.period) * #parsers.spacing
6324 * (parsers.tab + parsers.space^-2)
6325 + parsers.space * C(parsers.dig^2 * parsers.period)
6326 * #parsers.spacing
6327 + parsers.space * C(parsers.dig * parsers.period)
6328 * #parsers.spacing
6329 * (parsers.tab + parsers.space^-1)
6330 + parsers.space * parsers.space * C(parsers.dig^1
6331 * parsers.period) * #parsers.spacing

```

### 3.1.5.4 Parsers Used for Blockquotes (local)

```

6332 -- strip off leading > and indents, and run through blocks
6333 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
6334 1)/""
6335 * parsers.linechar^0 * parsers.newline)^1
6336 * (-V("BlockquoteExceptions") * parsers.linechar^1
6337 * parsers.newline)^0
6338
6339 if not options.breakableBlockquotes then
6340 parsers.blockquote_body = parsers.blockquote_body
6341 * (parsers.blankline^0 / "")
6342 end

```

### 3.1.5.5 Helpers for Links and Link Reference Definitions (local)

```

6342 -- List of references defined in the document
6343 local references
6344

```

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.

```

6345 function self.register_link(tag, url, title,
6346 attributes)
6347 tag = self.normalize_tag(tag)
6348 references[tag] = {
6349 url = url,
6350 title = title,
6351 attributes = attributes,
6352 }
6353 return ""
6354 end
6355

```

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.

```

6356 function self.lookup_reference(label, sps, tag,
6357 attributes)
6358 local tagpart
6359 if not tag then
6360 tag = label
6361 tagpart = ""
6362 elseif tag == "" then
6363 tag = label
6364 tagpart = "[]"
6365 else
6366 tagpart = {
6367 "[",
6368 self.parser_functions.parse_inlines(tag),
6369 "]"
6370 }
6371 end
6372 if sps then
6373 tagpart = {sps, tagpart}
6374 end
6375 tag = self.normalize_tag(tag)
6376 local r = references[tag]
6377 if r then
6378 local merged_attributes = {}
6379 for _, attribute in ipairs(r.attributes or {}) do
6380 table.insert(merged_attributes, attribute)
6381 end
6382 for _, attribute in ipairs(attributes or {}) do
6383 table.insert(merged_attributes, attribute)
6384 end

```

```

6385 if #merged_attributes == 0 then
6386 merged_attributes = nil
6387 end
6388 return {
6389 url = r.url,
6390 title = r.title,
6391 attributes = merged_attributes,
6392 }
6393 else
6394 return nil, {
6395 "[",
6396 self.parser_functions.parse_inlines(label),
6397 "]",
6398 tagpart
6399 }
6400 end
6401 end
6402
6403 -- lookup link reference and return a link, if the reference is found,
6404 -- or a bracketed label otherwise.
6405 local function indirect_link(label, sps, tag)
6406 return writer.defer_call(function()
6407 local r, fallback = self.lookup_reference(label, sps, tag)
6408 if r then
6409 return writer.link(
6410 self.parser_functions.parse_inlines_no_link(label),
6411 r.url, r.title)
6412 else
6413 return fallback
6414 end
6415 end)
6416 end
6417
6418 -- lookup image reference and return an image, if the reference is found,
6419 -- or a bracketed label otherwise.
6420 local function indirect_image(label, sps, tag)
6421 return writer.defer_call(function()
6422 local r, fallback = self.lookup_reference(label, sps, tag)
6423 if r then
6424 return writer.image(writer.string(label), r.url, r.title)
6425 else
6426 return {"!", fallback}
6427 end
6428 end)
6429 end
6430
6431 parsers.direct_link_tail = parsers.spnl

```

```

6432 * parsers.lparent
6433 * (parsers.url + Cc("")) -- link can be empty [foo]()
6434 * parsers.optionaltitle
6435 * parsers.rparent
6436
6437 parsers.direct_link = (parsers.tag / self.parser_functions.parse_inlines_no_link)
6438 * parsers.direct_link_tail
6439
6440 parsers.direct_image = parsers.exclamation
6441 * (parsers.tag / self.parser_functions.parse_inlines)
6442 * parsers.direct_link_tail

```

### 3.1.5.6 Inline Elements (local)

```

6443 parsers.Str = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
6444 / writer.string
6445
6446 parsers.Symbol = (V("SpecialChar") - parsers.tightblocksep)
6447 / writer.string
6448
6449 parsers.Ellipsis = P("...") / writer.ellipsis
6450
6451 parsers.Smart = parsers.Ellipsis
6452
6453 parsers.Code = parsers.inticks / writer.code
6454
6455 if options.blankBeforeBlockquote then
6456 parsers.bqstart = parsers.fail
6457 else
6458 parsers.bqstart = parsers.more
6459 end
6460
6461 if options.blankBeforeHeading then
6462 parsers.headerstart = parsers.fail
6463 else
6464 parsers.headerstart = parsers.hash
6465 + (parsers.line * (parsers.equal^1 + parsers.dash^1)
6466 * parsers.optionalspace * parsers.newline)
6467 end
6468
6469 parsers.EndlineExceptions
6470 = parsers.blankline -- paragraph break
6471 + parsers.tightblocksep -- nested list
6472 + parsers.eof -- end of document
6473 + parsers.bqstart
6474 + parsers.headerstart
6475

```

```

6476 parsers.Endline = parsers.newline
6477 * -V("EndlineExceptions")
6478 * parsers.spacechar^0
6479 / (options.hardLineBreaks and writer.hard_line_break
6480 or writer.space)
6481
6482 parsers.OptionalIndent
6483 = parsers.spacechar^1 / writer.space
6484
6485 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6486 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6487 + parsers.spacechar^1 * parsers.Endline
6488 * parsers.optionalspace
6489 / (options.hardLineBreaks
6490 and writer.hard_line_break
6491 or writer.space)
6492 + parsers.spacechar^1 * parsers.optionalspace
6493 / writer.space
6494
6495 parsers.NonbreakingEndline
6496 = parsers.newline
6497 * -V("EndlineExceptions")
6498 * parsers.spacechar^0
6499 / (options.hardLineBreaks and writer.hard_line_break
6500 or writer.nbsp)
6501
6502 parsers.NonbreakingSpace
6503 = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6504 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6505 + parsers.spacechar^1 * parsers.Endline
6506 * parsers.optionalspace
6507 / (options.hardLineBreaks
6508 and writer.hard_line_break
6509 or writer.nbsp)
6510 + parsers.spacechar^1 * parsers.optionalspace
6511 / writer.nbsp
6512
6513 if options.underscores then
6514 parsers.Strong = (parsers.between(parsers.Inline, parsers.doubleasterisks,
6515 parsers.doubleasterisks)
6516 + parsers.between(parsers.Inline, parsers.doubleunderscores,
6517 parsers.doubleunderscores)
6518) / writer.strong
6519
6520 parsers.Emph = (parsers.between(parsers.Inline, parsers.asterisk,
6521 parsers.asterisk)
6522 + parsers.between(parsers.Inline, parsers.underscore,

```



```

6523 parsers.underscore)
6524) / writer.emphasis
6525 else
6526 parsers.Strong = (parsers.between(parsers.Inline, parsers.doubleasterisks,
6527 parsers.doubleasterisks)
6528) / writer.strong
6529
6530 parsers.Emph = (parsers.between(parsers.Inline, parsers.asterisk,
6531 parsers.asterisk)
6532) / writer.emphasis
6533 end
6534

```

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.

```

6535 function self.auto_link_url(url, attributes)
6536 return writer.link(writer.escape(url),
6537 url, nil, attributes)
6538 end
6539

```

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.

```

6540 function self.auto_link_email(email, attributes)
6541 return writer.link(writer.escape(email),
6542 "mailto:".email,
6543 nil, attributes)
6544 end
6545
6546 parsers.AutoLinkUrl = parsers.auto_link_url
6547 / self.auto_link_url
6548
6549 parsers.AutoLinkEmail
6550 = parsers.auto_link_email
6551 / self.auto_link_email
6552
6553 parsers.AutoLinkRelativeReference
6554 = parsers.auto_link_relative_reference
6555 / self.auto_link_url
6556
6557 parsers.DirectLink = parsers.direct_link
6558 / writer.link
6559
6560 parsers.IndirectLink = parsers.indirect_link
6561 / indirect_link

```

```

6562
6563 -- parse a link or image (direct or indirect)
6564 parsers.Link = parsers.DirectLink + parsers.IndirectLink
6565
6566 parsers.DirectImage = parsers.direct_image
6567 / writer.image
6568
6569 parsers.IndirectImage = parsers.indirect_image
6570 / indirect_image
6571
6572 parsers.Image = parsers.DirectImage + parsers.IndirectImage
6573
6574 -- avoid parsing long strings of * or _ as emph/strong
6575 parsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
6576 / writer.string
6577
6578 parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
6579
6580 parsers.InlineHtml = parsers.emptyelt_any / writer.inline_html_tag
6581 + (parsers.htmlcomment / self.parser_functions.parse_inlines
6582 / writer.inline_html_comment
6583 + parsers.htmlinstruction
6584 + parsers.openelt_any / writer.inline_html_tag
6585 + parsers.closeelt_any / writer.inline_html_tag
6586
6587 parsers.HtmlEntity = parsers.hexentity / entities.hex_entity / writer.string
6588 + parsers.decentity / entities.dec_entity / writer.string
6589 + parsers.tagentity / entities.char_entity / writer.string

```

### 3.1.5.7 Block Elements (local)

```

6590 parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_ne
6591 / writer.block_html_comment
6592 + parsers.emptyelt_block / writer.block_html_element
6593 + parsers.openelt_exact("hr") / writer.block_html_element
6594 + parsers.in_matched_block_tags / writer.block_html_element
6595 + parsers.htmlinstruction
6596
6597 parsers.Verbatim = Cs((parsers.blanklines
6598 * ((parsers.indentedline - parsers.blankline))^1)^1
6599) / self.expandtabs / writer.verbatim
6600
6601 parsers.BlockquoteExceptions = parsers.leader * parsers.more
6602 + parsers.blankline
6603
6604 parsers.Blockquote = Cs(parsers.blockquote_body^1)
6605 / self.parser_functions.parse_blocks_nested

```

```

6606 / writer.blockquote
6607
6608 parsers.ThematicBreak = (parsers.lineof(parsers.asterisk)
6609 + parsers.lineof(parsers.dash)
6610 + parsers.lineof(parsers.underscore)
6611) / writer.thematic_break
6612
6613 parsers.Reference = parsers.define_reference_parser
6614 * parsers.blankline^1
6615 / self.register_link
6616
6617 parsers.Paragraph = parsers.nonindentspace * Ct(parsers.Inline^1)
6618 * (parsers.newline
6619 * (parsers.blankline^1
6620 + #V("EndlineExceptions")
6621)
6622 + parsers.eof)
6623 / writer.paragraph
6624
6625 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
6626 / writer.plain

```

### 3.1.5.8 Lists (local)

```

6627 parsers.starter = parsers.bullet + parsers.enumerator
6628
6629 if options.taskLists then
6630 parsers.tickbox = (parsers.ticked_box
6631 + parsers.halfticked_box
6632 + parsers.unticked_box
6633) / writer.tickbox
6634 else
6635 parsers.tickbox = parsers.fail
6636 end
6637
6638 -- we use \001 as a separator between a tight list item and a
6639 -- nested list under it.
6640 parsers.NestedList = Cs((parsers.optionallyindentedline
6641 - parsers.starter)^1)
6642 / function(a) return "\001"..a end
6643
6644 parsers.ListBlockLine = parsers.optionallyindentedline
6645 - parsers.blankline - (parsers.indent^1
6646 * parsers.starter)
6647
6648 parsers.ListBlock = parsers.line * parsers.ListBlockLine^0

```

```

6649
6650 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6651 * parsers.ListBlock
6652
6653 parsers.TightListItem = function(starter)
6654 return -parsers.ThematicBreak
6655 * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Ne
1)
6656 / self.parser_functions.parse_blocks_nested)
6657 * -(parsers.blanklines * parsers.indent)
6658 end
6659
6660 parsers.LooseListItem = function(starter)
6661 return -parsers.ThematicBreak
6662 * Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * Cc("\n")
6663 * (parsers.NestedList + parsers.ListContinuationBlock^0)
6664 * (parsers.blanklines / "\n\n")
6665) / self.parser_functions.parse_blocks_nested
6666 end
6667
6668 parsers.BulletList = (Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
6669 * parsers.skipblanklines * -parsers.bullet
6670 + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
6671 * parsers.skipblanklines)
6672 / writer.bulletlist
6673
6674 local function ordered_list(items,tight,startnum)
6675 if options.startNumber then
6676 startnum = tonumber(startnum) or 1 -- fallback for '#'
6677 if startnum ~= nil then
6678 startnum = math.floor(startnum)
6679 end
6680 else
6681 startnum = nil
6682 end
6683 return writer.orderedlist(items,tight,startnum)
6684 end
6685
6686 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
6687 (Ct(parsers.TightListItem(Cb("listtype")))
6688 * parsers.TightListItem(parsers.enumerator)^0)
6689 * Cc(true) * parsers.skipblanklines * -parsers.enumerator
6690 + Ct(parsers.LooseListItem(Cb("listtype")))
6691 * parsers.LooseListItem(parsers.enumerator)^0)
6692 * Cc(false) * parsers.skipblanklines
6693) * Cb("listtype") / ordered_list

```

### 3.1.5.9 Blank (local)

```
6694 parsers.Blank = parsers.blankline / ""
6695 + V("Reference")
6696 + (parsers.tightblocksep / "\n")
```

### 3.1.5.10 Headings (local)

```
6697 -- parse atx header
6698 parsers.AtxHeading = Cg(parsers.heading_start, "level")
6699 * parsers.optionalspace
6700 * (C(parsers.line)
6701 / strip_atx_end
6702 / self.parser_functions.parse_inlines)
6703 * Cb("level")
6704 / writer.heading
6705
6706 parsers.SetextHeading = #(parsers.line * S("--"))
6707 * Ct(parsers.linechar~1
6708 / self.parser_functions.parse_inlines)
6709 * parsers.newline
6710 * parsers.heading_level
6711 * parsers.optionalspace
6712 * parsers.newline
6713 / writer.heading
6714
6715 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  $\text{\TeX}$  output.

```
6716 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.

```
6717 local walkable_syntax = (function(global_walkable_syntax)
6718 local local_walkable_syntax = {}
6719 for lhs, rule in pairs(global_walkable_syntax) do
6720 local_walkable_syntax[lhs] = util.table_copy(rule)
6721 end
6722 return local_walkable_syntax
6723 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.

```

6724 local current_extension_name = nil
6725 self.insert_pattern = function(selector, pattern, pattern_name)
6726 assert(pattern_name == nil or type(pattern_name) == "string")
6727 local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
6728 assert(lhs ~= nil,
6729 [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
6730 .. selector .. ["])
6731 assert(walkable_syntax[lhs] ~= nil,
6732 [[Rule]] .. lhs .. [[-> ... does not exist in markdown grammar]])
6733 assert(pos == "before" or pos == "after" or pos == "instead of",
6734 [[Expected positional specifier "before", "after", or "instead of", not "]]
6735 .. pos .. ["])
6736 local rule = walkable_syntax[lhs]
6737 local index = nil
6738 for current_index, current_rhs in ipairs(rule) do
6739 if type(current_rhs) == "string" and current_rhs == rhs then
6740 index = current_index
6741 if pos == "after" then
6742 index = index + 1
6743 end
6744 break
6745 end
6746 end
6747 assert(index ~= nil,
6748 [[Rule]] .. lhs .. [[->]] .. rhs
6749 .. [[does not exist in markdown grammar]])
6750 local accountable_pattern
6751 if current_extension_name then
6752 accountable_pattern = { pattern, current_extension_name, pattern_name }
6753 else
6754 assert(type(pattern) == "string",
6755 [[reader->insert_pattern() was called outside an extension with]]
6756 .. [[a PEG pattern instead of a rule name]])
6757 accountable_pattern = pattern
6758 end
6759 if pos == "instead of" then
6760 rule[index] = accountable_pattern
6761 else
6762 table.insert(rule, index, accountable_pattern)
6763 end
6764 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.

```

6765 local syntax =
6766 { "Blocks",
6767
6768 Blocks = V("InitializeState")
6769 * (V("ExpectedJekyllData")
6770 * (V("Blank")^0 / writer.interblocksep))^~
1
6771 * V("Blank")^0
6772 * V("Block")^-1
6773 * (V("Blank")^0 / writer.interblocksep
6774 * V("Block"))^0
6775 * V("Blank")^0 * parsers.eof,
6776
6777 ExpectedJekyllData = parsers.fail,
6778
6779 Blank = parsers.Blank,
6780 Reference = parsers.Reference,
6781
6782 Blockquote = parsers.Blockquote,
6783 Verbatim = parsers.Verbatim,
6784 ThematicBreak = parsers.ThematicBreak,
6785 BulletList = parsers.BulletList,
6786 OrderedList = parsers.OrderedList,
6787 Heading = parsers.Heading,
6788 DisplayHtml = parsers.DisplayHtml,
6789 Paragraph = parsers.Paragraph,
6790 Plain = parsers.Plain,
6791
6792 EndlineExceptions = parsers.EndlineExceptions,
6793 BlockquoteExceptions = parsers.BlockquoteExceptions,
6794
6795 Str = parsers.Str,
6796 Space = parsers.Space,
6797 OptionalIndent = parsers.OptionalIndent,
6798 Endline = parsers.Endline,
6799 U1OrStarLine = parsers.U1OrStarLine,
6800 Strong = parsers.Strong,
6801 Emph = parsers.Emph,
6802 Link = parsers.Link,
6803 Image = parsers.Image,
6804 Code = parsers.Code,
6805 AutoLinkUrl = parsers.AutoLinkUrl,
6806 AutoLinkEmail = parsers.AutoLinkEmail,
6807 AutoLinkRelativeReference
6808 = parsers.AutoLinkRelativeReference,
6809 InlineHtml = parsers.InlineHtml,
6810 HtmlEntity = parsers.HtmlEntity,

```

```

6811 EscapedChar = parsers.EscapedChar,
6812 Smart = parsers.Smart,
6813 Symbol = parsers.Symbol,
6814 SpecialChar = parsers.fail,
6815 InitializeState = parsers.succeed,
6816 }

```

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]`.

```

6817 self.update_rule = function(rule_name, get_pattern)
6818 assert(current_extension_name ~= nil)
6819 assert(syntax[rule_name] ~= nil,
6820 [[Rule]] .. rule_name .. [[-> ... does not exist in markdown grammar]])
6821 local previous_pattern
6822 local extension_name
6823 if walkable_syntax[rule_name] then
6824 local previous_accountable_pattern = walkable_syntax[rule_name][1]
6825 previous_pattern = previous_accountable_pattern[1]
6826 extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
6827 else
6828 previous_pattern = nil
6829 extension_name = current_extension_name
6830 end
6831 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

```

```

6832 if type(get_pattern) == "function" then
6833 pattern = get_pattern(previous_pattern)
6834 else
6835 assert(previous_pattern == nil,
6836 [[Rule]] .. rule_name ..
6837 [[has already been updated by]] .. extension_name)
6838 pattern = get_pattern
6839 end

```



```

6840 local accountable_pattern = { pattern, extension_name, rule_name }
6841 walkable_syntax[rule_name] = { accountable_pattern }
6842 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.

```

6843 local special_characters = {}
6844 self.add_special_character = function(c)
6845 table.insert(special_characters, c)
6846 syntax.SpecialChar = S(table.concat(special_characters, ""))
6847 end
6848
6849 self.add_special_character("*")
6850 self.add_special_character("[")
6851 self.add_special_character("]")
6852 self.add_special_character("<")
6853 self.add_special_character("!")
6854 self.add_special_character("\\")

```

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

```

6855 self.initialize_named_group = function(name, value)
6856 syntax.InitializeState = syntax.InitializeState
6857 * Cg(Ct("") / value, name)
6858 end

```

Apply syntax extensions.

```

6859 for _, extension in ipairs(extensions) do
6860 current_extension_name = extension.name
6861 extension.extend_writer(writer)
6862 extension.extend_reader(self)
6863 end
6864 current_extension_name = nil

```

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

```

6865 if options.debugExtensions then
6866 local sorted_lhs = {}
6867 for lhs, _ in pairs(walkable_syntax) do
6868 table.insert(sorted_lhs, lhs)
6869 end
6870 table.sort(sorted_lhs)
6871
6872 local output_lines = {"{"}
6873 for lhs_index, lhs in ipairs(sorted_lhs) do
6874 local encoded_lhs = util.encode_json_string(lhs)
6875 table.insert(output_lines, [[]] .. encoded_lhs .. [[:]])

```

```

6876 local rule = walkable_syntax[lhs]
6877 for rhs_index, rhs in ipairs(rule) do
6878 local human_readable_rhs
6879 if type(rhs) == "string" then
6880 human_readable_rhs = rhs
6881 else
6882 local pattern_name
6883 if rhs[3] then
6884 pattern_name = rhs[3]
6885 else
6886 pattern_name = "Anonymous Pattern"
6887 end
6888 local extension_name = rhs[2]
6889 human_readable_rhs = pattern_name .. [[(]] .. extension_name .. [[]]
6890 end
6891 local encoded_rhs = util.encode_json_string(human_readable_rhs)
6892 local output_line = [[]] .. encoded_rhs
6893 if rhs_index < #rule then
6894 output_line = output_line .. ", "
6895 end
6896 table.insert(output_lines, output_line)
6897 end
6898 local output_line = "]"
6899 if lhs_index < #sorted_lhs then
6900 output_line = output_line .. ", "
6901 end
6902 table.insert(output_lines, output_line)
6903 end
6904 table.insert(output_lines, "}")
6905
6906 local output = table.concat(output_lines, "\n")
6907 local output_filename = options.debugExtensionsFileName
6908 local output_file = assert(io.open(output_filename, "w"),
6909 [[Could not open file]] .. output_filename .. [[for writing]])
6910 assert(output_file:write(output))
6911 assert(output_file:close())
6912 end

```

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

```

6913 walkable_syntax["IndentedInline"] = util.table_copy(
6914 walkable_syntax["Inline"])
6915 self.insert_pattern(
6916 "IndentedInline instead of Space",
6917 "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`.

```
6918 for lhs, rule in pairs(walkable_syntax) do
6919 syntax[lhs] = parsers.fail
6920 for _, rhs in ipairs(rule) do
6921 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.

```
6922 if type(rhs) == "string" then
6923 pattern = V(rhs)
6924 else
6925 pattern = rhs[1]
6926 if type(pattern) == "string" then
6927 pattern = V(pattern)
6928 end
6929 end
6930 syntax[lhs] = syntax[lhs] + pattern
6931 end
6932 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.

```
6933 if options.underscores then
6934 self.add_special_character("_")
6935 end
6936
6937 if not options.codeSpans then
6938 syntax.Code = parsers.fail
6939 else
6940 self.add_special_character("`")
6941 end
6942
6943 if not options.html then
6944 syntax.DisplayHtml = parsers.fail
6945 syntax.InlineHtml = parsers.fail
6946 syntax.HtmlEntity = parsers.fail
6947 else
6948 self.add_special_character("&")
6949 end
6950
6951 if options.preserveTabs then
6952 options.stripIndent = false
6953 end
```

```

6954
6955 if not options.smartEllipses then
6956 syntax.Smart = parsers.fail
6957 else
6958 self.add_special_character(".")
6959 end
6960
6961 if not options.relativeReferences then
6962 syntax.AutoLinkRelativeReference = parsers.fail
6963 end
6964
6965 local blocks_nested_t = util.table_copy(syntax)
6966 blocks_nested_t.ExpectedJekyllData = parsers.fail
6967 parsers.blocks_nested = Ct(blocks_nested_t)
6968
6969 parsers.blocks = Ct(syntax)
6970
6971 local inlines_t = util.table_copy(syntax)
6972 inlines_t[1] = "Inlines"
6973 inlines_t.Inlines = V("InitializeState")
6974 * parsers.Inline^0
6975 * (parsers.spacing^0
6976 * parsers.eof / "")
6977 parsers.inlines = Ct(inlines_t)
6978
6979 local inlines_no_link_t = util.table_copy(inlines_t)
6980 inlines_no_link_t.Link = parsers.fail
6981 parsers.inlines_no_link = Ct(inlines_no_link_t)
6982
6983 local inlines_no_inline_note_t = util.table_copy(inlines_t)
6984 inlines_no_inline_note_t.InlineNote = parsers.fail
6985 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6986
6987 local inlines_no_html_t = util.table_copy(inlines_t)
6988 inlines_no_html_t.DisplayHtml = parsers.fail
6989 inlines_no_html_t.InlineHtml = parsers.fail
6990 inlines_no_html_t.HtmlEntity = parsers.fail
6991 parsers.inlines_no_html = Ct(inlines_no_html_t)
6992
6993 local inlines_nbsp_t = util.table_copy(inlines_t)
6994 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6995 inlines_nbsp_t.Space = parsers.NonbreakingSpace
6996 parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

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

```

6997 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.

```
6998 input = input:gsub("\r\n?", "\n")
6999 if input:sub(-1) ~= "\n" then
7000 input = input .. "\n"
7001 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.

```
7002 references = {}
7003 local opt_string = {}
7004 for k, _ in pairs(defaultOptions) do
7005 local v = options[k]
7006 if type(v) == "table" then
7007 for _, i in ipairs(v) do
7008 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
7009 end
7010 elseif k ~= "cacheDir" then
7011 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
7012 end
7013 end
7014 table.sort(opt_string)
7015 local salt = table.concat(opt_string, ",") .. "," .. metadata.version
7016 local output
```

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

```
7017 local function convert(input)
7018 local document = self.parser_functions.parse_blocks(input)
7019 return util.rope_to_string(writer.document(document))
7020 end
7021 if options.eagerCache or options.finalizeCache then
7022 local name = util.cache(options.cacheDir, input, salt, convert,
7023 ".md" .. writer.suffix)
7024 output = writer.pack(name)
```

Otherwise, return the result of the conversion directly.

```
7025 else
7026 output = convert(input)
7027 end
```

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

```
7028 if options.finalizeCache then
7029 local file, mode
7030 if options.frozenCacheCounter > 0 then
```

```

7031 mode = "a"
7032 else
7033 mode = "w"
7034 end
7035 file = assert(io.open(options.frozenCacheFileName, mode),
7036 [[Could not open file "]] .. options.frozenCacheFileName
7037 .. [{" for writing}])
7038 assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname]]
7039 .. [{"markdownFrozenCache}] .. options.frozenCacheCounter
7040 .. [{"\\endcsname{}}] .. output .. [{"}]]) .. "\\n"))
7041 assert(file:close())
7042 end
7043 return output
7044 end
7045 end
7046 return self
7047 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.

```
7048 M.extensions = {}
```

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

```

7049 M.extensions.bracketed_spans = function()
7050 return {
7051 name = "built-in bracketed_spans syntax extension",
7052 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.

```

7053 function self.span(s, attr)
7054 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
7055 self.attributes(attr),
7056 s,
7057 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
7058 end
7059 end, extend_reader = function(self)
7060 local parsers = self.parsers
7061 local writer = self.writer
7062

```

```

7063 local Span = parsers.between(parsers.Inline,
7064 parsers.lbracket,
7065 parsers.rbracket)
7066 * Ct(parsers.attributes)
7067 / writer.span
7068
7069 self.insert_pattern("Inline after Emph",
7070 Span, "Span")
7071 end
7072 }
7073 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.

```

7074 M.extensions.citations = function(citation_nbsps)
7075 return {
7076 name = "built-in citations syntax extension",
7077 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.

```

7078 function self.citations(text_cites, cites)
7079 local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
7080 "{", #cites, "}"}
7081 for _,cite in ipairs(cites) do
7082 buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
7083 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
7084 end
7085 return buffer
7086 end
7087 end, extend_reader = function(self)

```

```

7088 local parsers = self.parsers
7089 local writer = self.writer
7090
7091 local citation_chars
7092 = parsers.alphanumeric
7093 + S("#$%&-+<>~/_")
7094
7095 local citation_name
7096 = Cs(parsers.dash^-1) * parsers.at
7097 * Cs(citation_chars
7098 * (((citation_chars + parsers.internal_punctuation
7099 - parsers.comma - parsers.semicolon)
7100 * -#((parsers.internal_punctuation - parsers.comma
7101 - parsers.semicolon)^0
7102 * -(citation_chars + parsers.internal_punctuation
7103 - parsers.comma - parsers.semicolon)))^0
7104 * citation_chars)^-1)
7105
7106 local citation_body_prenote
7107 = Cs((parsers.alphanumeric^1
7108 + parsers.bracketed
7109 + parsers.inticks
7110 + (parsers.anyescaped
7111 - (parsers.rbracket + parsers.blankline^2))
7112 - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
7113
7114 local citation_body_postnote
7115 = Cs((parsers.alphanumeric^1
7116 + parsers.bracketed
7117 + parsers.inticks
7118 + (parsers.anyescaped
7119 - (parsers.rbracket + parsers.semicolon
7120 + parsers.blankline^2))
7121 - (parsers.spnl * parsers.rbracket))^0)
7122
7123 local citation_body_chunk
7124 = citation_body_prenote
7125 * parsers.spnl * citation_name
7126 * (parsers.internal_punctuation - parsers.semicolon)^-
1
7127 * parsers.spnl * citation_body_postnote
7128
7129 local citation_body
7130 = citation_body_chunk
7131 * (parsers.semicolon * parsers.spnl
7132 * citation_body_chunk)^0
7133

```



```

7134 local citation_headless_body_postnote
7135 = Cs((parsers.alphanumeric^1
7136 + parsers.bracketed
7137 + parsers.inticks
7138 + (parsers.anyescaped
7139 - (parsers.rbracket + parsers.at
7140 + parsers.semicolon + parsers.blankline^2))
7141 - (parsers.spnl * parsers.rbracket))^0)
7142
7143 local citation_headless_body
7144 = citation_headless_body_postnote
7145 * (parsers.sp * parsers.semicolon * parsers.spnl
7146 * citation_body_chunk)^0
7147
7148 local citations
7149 = function(text_cites, raw_cites)
7150 local function normalize(str)
7151 if str == "" then
7152 str = nil
7153 else
7154 str = (citation_nbsps and
7155 self.parser_functions.parse_inlines_nbsp or
7156 self.parser_functions.parse_inlines)(str)
7157 end
7158 return str
7159 end
7160
7161 local cites = {}
7162 for i = 1,#raw_cites,4 do
7163 cites[#cites+1] = {
7164 prenote = normalize(raw_cites[i]),
7165 suppress_author = raw_cites[i+1] == "-",
7166 name = writer.identifier(raw_cites[i+2]),
7167 postnote = normalize(raw_cites[i+3]),
7168 }
7169 end
7170 return writer.citations(text_cites, cites)
7171 end
7172
7173 local TextCitations
7174 = Ct((parsers.spnl
7175 * Cc("")
7176 * citation_name
7177 * ((parsers.spnl
7178 * parsers.lbracket
7179 * citation_headless_body
7180 * parsers.rbracket) + Cc("")))^1)

```

```

7181 / function(raw_cites)
7182 return citations(true, raw_cites)
7183 end
7184
7185 local ParenthesizedCitations
7186 = Ct((parsers.spnl
7187 * parsers.lbracket
7188 * citation_body
7189 * parsers.rbracket)^1)
7190 / function(raw_cites)
7191 return citations(false, raw_cites)
7192 end
7193
7194 local Citations = TextCitations + ParenthesizedCitations
7195
7196 self.insert_pattern("Inline after Emph",
7197 Citations, "Citations")
7198
7199 self.add_special_character("@")
7200 self.add_special_character("-")
7201 end
7202 }
7203 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.

```

7204 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.

```

7205 local languages_json = (function()
7206 local base, prev, curr
7207 for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
7208 local file = io.open(pathname, "r")
7209 if not file then goto continue end
7210 local input = assert(file:read("*a"))
7211 assert(file:close())
7212 local json = input:gsub('("[^\\n]-"):','[%1]=')
7213 curr = load("_ENV = {}; return "..json")()
7214 if type(curr) == "table" then
7215 if base == nil then
7216 base = curr

```

```

7217 else
7218 setmetatable(prev, { __index = curr })
7219 end
7220 prev = curr
7221 end
7222 ::continue::
7223 end
7224 return base or {}
7225 end)()
7226
7227 return {
7228 name = "built-in content_blocks syntax extension",
7229 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.

```

7230 function self.contentblock(src,suf,type,tit)
7231 if not self.is_writing then return "" end
7232 src = src.." "..suf
7233 suf = suf:lower()
7234 if type == "onlineimage" then
7235 return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
7236 {"",self.string(src),"} ",
7237 {"",self.uri(src),"} ",
7238 {"",self.string(tit or ""),"}"}
7239 elseif languages_json[suf] then
7240 return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
7241 {"",self.string(languages_json[suf]),"} ",
7242 {"",self.string(src),"} ",
7243 {"",self.uri(src),"} ",
7244 {"",self.string(tit or ""),"}"}
7245 else
7246 return {"\\markdownRendererContentBlock{" ,suf,"} ",
7247 {"",self.string(src),"} ",
7248 {"",self.uri(src),"} ",
7249 {"",self.string(tit or ""),"}"}
7250 end
7251 end
7252 end, extend_reader = function(self)
7253 local parsers = self.parsers
7254 local writer = self.writer
7255
7256 local contentblock_tail
7257 = parsers.optionaltitle
7258 * (parsers.newline + parsers.eof)

```

```

7259
7260 -- case insensitive online image suffix:
7261 local onlineimagesuffix
7262 = (function(...)
7263 local parser = nil
7264 for _, suffix in ipairs({...}) do
7265 local pattern=nil
7266 for i=1,#suffix do
7267 local char=suffix:sub(i,i)
7268 char = S(char:lower()..char:upper())
7269 if pattern == nil then
7270 pattern = char
7271 else
7272 pattern = pattern * char
7273 end
7274 end
7275 if parser == nil then
7276 parser = pattern
7277 else
7278 parser = parser + pattern
7279 end
7280 end
7281 return parser
7282 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
7283
7284 -- online image url for iA Writer content blocks with mandatory suffix,
7285 -- allowing nested brackets:
7286 local onlineimageurl
7287 = (parsers.less
7288 * Cs((parsers.anyescaped
7289 - parsers.more
7290 - #(parsers.period
7291 * onlineimagesuffix
7292 * parsers.more
7293 * contentblock_tail))^0)
7294 * parsers.period
7295 * Cs(onlineimagesuffix)
7296 * parsers.more
7297 + (Cs((parsers.inparens
7298 + (parsers.anyescaped
7299 - parsers.spacing
7300 - parsers.rparent
7301 - #(parsers.period
7302 * onlineimagesuffix
7303 * contentblock_tail))))^0)
7304 * parsers.period
7305 * Cs(onlineimagesuffix))

```

```

7306) * Cc("onlineimage")
7307
7308 -- filename for iA Writer content blocks with mandatory suffix:
7309 local localfilepath
7310 = parsers.slash
7311 * Cs((parsers.anyescaped
7312 - parsers.tab
7313 - parsers.newline
7314 - #(parsers.period
7315 * parsers.alphanumeric^1
7316 * contentblock_tail))^1)
7317 * parsers.period
7318 * Cs(parsers.alphanumeric^1)
7319 * Cc("localfile")
7320
7321 local ContentBlock
7322 = parsers.leader
7323 * (localfilepath + onlineimageurl)
7324 * contentblock_tail
7325 / writer.contentblock
7326
7327 self.insert_pattern("Block before Blockquote",
7328 ContentBlock, "ContentBlock")
7329 end
7330 }
7331 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.

```

7332 M.extensions.definition_lists = function(tight_lists)
7333 return {
7334 name = "built-in definition_lists syntax extension",
7335 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.

```

7336 local function dlitem(term, defs)
7337 local retVal = {"\\markdownRendererDlItem{",term,""}
7338 for _, def in ipairs(defs) do
7339 retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
7340 "\\markdownRendererDlDefinitionEnd "}
7341 end
7342 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "

```

```

7343 return retVal
7344 end
7345
7346 function self.definitionlist(items,tight)
7347 if not self.is_writing then return "" end
7348 local buffer = {}
7349 for _,item in ipairs(items) do
7350 buffer[#buffer + 1] = dlitem(item.term, item.definitions)
7351 end
7352 if tight and tight_lists then
7353 return {"\\markdownRendererDlBeginTight\n", buffer,
7354 "\\markdownRendererDlEndTight"}
7355 else
7356 return {"\\markdownRendererDlBegin\n", buffer,
7357 "\\markdownRendererDlEnd"}
7358 end
7359 end
7360 end, extend_reader = function(self)
7361 local parsers = self.parsers
7362 local writer = self.writer
7363
7364 local defstartchar = S("~:")
7365
7366 local defstart = (defstartchar * #parsers.spacing
7367 * (parsers.tab + parsers.space^-
3)
7368 + parsers.space * defstartchar * #parsers.spacing
7369 * (parsers.tab + parsers.space^-
2)
7370 + parsers.space * parsers.space * defstartchar
7371 * #parsers.spacing
7372 * (parsers.tab + parsers.space^-
1)
7373 + parsers.space * parsers.space * parsers.space
7374 * defstartchar * #parsers.spacing
7375)
7376
7377 local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
7378
7379 local function definition_list_item(term, defs, _)
7380 return { term = self.parser_functions.parse_inlines(term),
7381 definitions = defs }
7382 end
7383
7384 local DefinitionListItemLoose
7385 = C(parsers.line) * parsers.skipblanklines
7386 * Ct((defstart

```

```

7387 * parsers.indented_blocks(dlchunk)
7388 / self.parser_functions.parse_blocks_nested)^1)
7389 * Cc(false) / definition_list_item
7390
7391 local DefinitionListItemTight
7392 = C(parsers.line)
7393 * Ct((defstart * dlchunk
7394 / self.parser_functions.parse_blocks_nested)^1)
7395 * Cc(true) / definition_list_item
7396
7397 local DefinitionList
7398 = (Ct(DefinitionListItemLoose^1) * Cc(false)
7399 + Ct(DefinitionListItemTight^1)
7400 * (parsers.skipblanklines
7401 * -DefinitionListItemLoose * Cc(true))
7402) / writer.definitionlist
7403
7404 self.insert_pattern("Block after Heading",
7405 DefinitionList, "DefinitionList")
7406 end
7407 }
7408 end

```

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

```

7409 M.extensions.fancy_lists = function()
7410 return {
7411 name = "built-in fancy_lists syntax extension",
7412 extend_writer = function(self)
7413 local options = self.options
7414

```

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.

```

7415 function self.fancylist(items,tight,startnum,numstyle,numdelim)
7416 if not self.is_writing then return "" end
7417 local buffer = {}
7418 local num = startnum
7419 for _,item in ipairs(items) do
7420 buffer[#buffer + 1] = self.fancyitem(item,num)
7421 if num ~= nil then
7422 num = num + 1
7423 end
7424 end
7425 local contents = util.intersperse(buffer,"\n")
7426 if tight and options.tightLists then
7427 return {"\\markdownRendererFancyOlBeginTight{",
7428 numstyle,"}{",numdelim,"}",contents,
7429 "\\n\\markdownRendererFancyOlEndTight "}
7430 else
7431 return {"\\markdownRendererFancyOlBegin{",
7432 numstyle,"}{",numdelim,"}",contents,
7433 "\\n\\markdownRendererFancyOlEnd "}
7434 end
7435 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.

```

7436 function self.fancyitem(s,num)
7437 if num ~= nil then
7438 return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
7439 "\\n\\markdownRendererFancyOlItemEnd "}
7440 else
7441 return {"\\markdownRendererFancyOlItem ",s,"\\n\\markdownRendererFancyOlItemEnd "}
7442 end
7443 end
7444 end, extend_reader = function(self)
7445 local parsers = self.parsers
7446 local options = self.options
7447 local writer = self.writer
7448
7449 local label = parsers.dig + parsers.letter
7450 local numdelim = parsers.period + parsers.rparent

```



```

7451 local enumerator = C(label^3 * numdelim) * #parsers.spacing
7452 + C(label^2 * numdelim) * #parsers.spacing
7453 * (parsers.tab + parsers.space^1)
7454 + C(label * numdelim) * #parsers.spacing
7455 * (parsers.tab + parsers.space^-
2)
7456 + parsers.space * C(label^2 * numdelim)
7457 * #parsers.spacing
7458 + parsers.space * C(label * numdelim)
7459 * #parsers.spacing
7460 * (parsers.tab + parsers.space^-
1)
7461 + parsers.space * parsers.space * C(label^1
7462 * numdelim) * #parsers.spacing
7463 local starter = parsers.bullet + enumerator
7464
7465 local NestedList = Cs((parsers.optionallyindentedline
7466 - starter)^1)
7467 / function(a) return "\001"..a end
7468
7469 local ListBlockLine = parsers.optionallyindentedline
7470 - parsers.blankline - (parsers.indent^-1
7471 * starter)
7472
7473 local ListBlock = parsers.line * ListBlockLine^0
7474
7475 local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
7476 * ListBlock
7477
7478 local TightListItem = function(starter)
7479 return -parsers.ThematicBreak
7480 * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
7481 / self.parser_functions.parse_blocks_nested)
7482 * -(parsers.blanklines * parsers.indent)
7483 end
7484
7485 local LooseListItem = function(starter)
7486 return -parsers.ThematicBreak
7487 * Cs(starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
7488 * (NestedList + ListContinuationBlock^0)
7489 * (parsers.blanklines / "\n\n")
7490) / self.parser_functions.parse_blocks_nested
7491 end
7492
7493 local function roman2number(roman)
7494 local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }

```

```

7495 local numeral = 0
7496
7497 local i = 1
7498 local len = string.len(roman)
7499 while i < len do
7500 local z1, z2 = romans[string.sub(roman, i, i)], romans[string.sub(roman,
7501 if z1 < z2 then
7502 numeral = numeral + (z2 - z1)
7503 i = i + 2
7504 else
7505 numeral = numeral + z1
7506 i = i + 1
7507 end
7508 end
7509 if i <= len then numeral = numeral + romans[string.sub(roman,i,i)] end
7510 return numeral
7511 end
7512
7513 local function sniffstyle(itemprefix)
7514 local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])*")
7515 local numdelim
7516 if delimend == ")" then
7517 numdelim = "OneParen"
7518 elseif delimend == "." then
7519 numdelim = "Period"
7520 else
7521 numdelim = "Default"
7522 end
7523 numstr = numstr or itemprefix
7524
7525 local num
7526 num = numstr:match("^([IVXL]+)")
7527 if num then
7528 return roman2number(num), "UpperRoman", numdelim
7529 end
7530 num = numstr:match("^([ivxl]+)")
7531 if num then
7532 return roman2number(string.upper(num)), "LowerRoman", numdelim
7533 end
7534 num = numstr:match("^([A-Z])")
7535 if num then
7536 return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
7537 end
7538 num = numstr:match("^([a-z])")
7539 if num then
7540 return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
7541 end

```

```

7542 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
7543 end
7544
7545 local function fancylist(items,tight,start)
7546 local startnum, numstyle, numdelim = sniffstyle(start)
7547 return writer.fancylist(items,tight,
7548 options.startNumber and startnum,
7549 numstyle or "Decimal",
7550 numdelim or "Default")
7551 end
7552
7553 local FancyList = Cg(enumerator, "listtype") *
7554 (Ct(TightListItem(Cb("listtype")))
7555 * TightListItem(enumerator)^0
7556 * Cc(true) * parsers.skipblanklines * -enumerator
7557 + Ct(LooseListItem(Cb("listtype")))
7558 * LooseListItem(enumerator)^0
7559 * Cc(false) * parsers.skipblanklines
7560) * Cb("listtype") / fancylist
7561
7562 self.update_rule("OrderedList", FancyList)
7563 end
7564 }
7565 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.

```

7566 M.extensions.fenced_code = function(blank_before_code_fence,
7567 allow_attributes,
7568 allow_raw_blocks)
7569 return {
7570 name = "built-in fenced_code syntax extension",
7571 extend_writer = function(self)
7572 local options = self.options
7573

```

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.

```

7574 function self.fencedCode(s, i, attr)
7575 if not self.is_writing then return "" end

```

```

7576 s = s:gsub("\n$", "")
7577 local buf = {}
7578 if attr ~= nil then
7579 table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
7580 self.attributes(attr)})
7581 end
7582 local name = util.cache_verbatim(options.cacheDir, s)
7583 table.insert(buf, {"\\markdownRendererInputFencedCode{" ,
7584 name,"}{" ,self.string(i),"}"}))
7585 if attr ~= nil then
7586 table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd")
7587 end
7588 return buf
7589 end
7590

```

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

```

7591 if allow_raw_blocks then
7592 function self.rawBlock(s, attr)
7593 if not self.is_writing then return "" end
7594 s = s:gsub("\n$", "")
7595 local name = util.cache_verbatim(options.cacheDir, s)
7596 return {"\\markdownRendererInputRawBlock{" ,
7597 name,"}{" , self.string(attr),"}"}
7598 end
7599 end
7600 end, extend_reader = function(self)
7601 local parsers = self.parsers
7602 local writer = self.writer
7603
7604 local function captures_geq_length(_,i,a,b)
7605 return #a >= #b and i
7606 end
7607
7608 local tilde_infostring
7609 = C((parsers.linechar
7610 - (parsers.spacechar^1 * parsers.newline))^0)
7611
7612 local backtick_infostring
7613 = C((parsers.linechar
7614 - (parsers.backtick
7615 + parsers.spacechar^1 * parsers.newline))^0)
7616
7617 local fenceindent
7618 local fencehead = function(char, infostring)
7619 return C(parsers.nonindentSPACE) / function(s) fenceindent = #s

```

```

7620 * Cg(char^3, "fencelength")
7621 * parsers.optionalspace
7622 * infostring
7623 * (parsers.newline + parsers.eof)
7624 end
7625
7626 local fencetail = function(char)
7627 return parsers.nonindentospace
7628 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
7629 * parsers.optionalspace * (parsers.newline + parsers.eof)
7630 + parsers.eof
7631 end
7632
7633 local fencedline = function(char)
7634 return C(parsers.line - fencetail(char))
7635 / function(s)
7636 local i = 1
7637 local remaining = fenceindent
7638 while true do
7639 local c = s:sub(i, i)
7640 if c == " " and remaining > 0 then
7641 remaining = remaining - 1
7642 i = i + 1
7643 elseif c == "\t" and remaining > 3 then
7644 remaining = remaining - 4
7645 i = i + 1
7646 else
7647 break
7648 end
7649 end
7650 return s:sub(i)
7651 end
7652 end
7653
7654 local TildeFencedCode
7655 = fencehead(parsers.tilde, tilde_infostring)
7656 * Cs(fencedline(parsers.tilde)^0)
7657 * fencetail(parsers.tilde)
7658
7659 local BacktickFencedCode
7660 = fencehead(parsers.backtick, backtick_infostring)
7661 * Cs(fencedline(parsers.backtick)^0)
7662 * fencetail(parsers.backtick)
7663
7664 local infostring_with_attributes
7665 = Ct(C((parsers.linechar
7666 - (parsers.optionalspace

```

```

7667 * parsers.attributes))^0)
7668 * parsers.optionalspace
7669 * Ct(parsers.attributes))
7670
7671 local FencedCode
7672 = (TildeFencedCode + BacktickFencedCode)
7673 / function(infostring, code)
7674 local expanded_code = self.expandtabs(code)
7675
7676 if allow_raw_blocks then
7677 local raw_attr = lpeg.match(parsers.raw_attribute,
7678 infostring)
7679
7680 if raw_attr then
7681 return writer.rawBlock(expanded_code, raw_attr)
7682 end
7683 end
7684
7685 local attr = nil
7686 if allow_attributes then
7687 local match = lpeg.match(infostring_with_attributes,
7688 infostring)
7689
7690 if match then
7691 infostring, attr = table.unpack(match)
7692 end
7693 end
7694 return writer.fencedCode(expanded_code, infostring, attr)
7695 end
7696
7697 self.insert_pattern("Block after Verbatim",
7698 FencedCode, "FencedCode")
7699
7700 local fencestart
7701 if blank_before_code_fence then
7702 fencestart = parsers.fail
7703 else
7704 fencestart = fencehead(parsers.backtick, backtick_infostring)
7705 + fencehead(parsers.tilde, tilde_infostring)
7706 end
7707
7708 self.update_rule("EndlineExceptions", function(previous_pattern)
7709 if previous_pattern == nil then
7710 previous_pattern = parsers.EndlineExceptions
7711 end
7712 return previous_pattern + fencestart
7713 end)
7714
7715 self.add_special_character("`")

```

```

7714 self.add_special_character("~")
7715 end
7716 }
7717 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.

```

7718 M.extensions.fenced_divs = function(blank_before_div_fence)
7719 return {
7720 name = "built-in fenced_divs syntax extension",
7721 extend_writer = function(self)

```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```

7722 function self.div_begin(attributes)
7723 local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",
7724 self.attributes(attributes)}
7725 local end_output = {"\\n\\markdownRendererFencedDivAttributeContextEnd "}
7726 return self.push_attributes("div", attributes, start_output, end_output)
7727 end

```

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

```

7728 function self.div_end()
7729 return self.pop_attributes("div")
7730 end
7731 end, extend_reader = function(self)
7732 local parsers = self.parsers
7733 local writer = self.writer

```

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

```

7734 local fenced_div_infostring
7735 = C((parsers.linechar
7736 - (parsers.spacechar^1
7737 * parsers.colon^1))^1)
7738
7739 local fenced_div_begin = parsers.nonindentospace
7740 * parsers.colon^3
7741 * parsers.optionalspace
7742 * fenced_div_infostring
7743 * (parsers.spacechar^1
7744 * parsers.colon^1)^0
7745 * parsers.optionalspace
7746 * (parsers.newline + parsers.eof)
7747

```

```

7748 local fenced_div_end = parsers.nonindentSPACE
7749 * parsers.colon^3
7750 * parsers.optionalspace
7751 * (parsers.newline + parsers.eof)

```

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

```

7752 self.initialize_named_group("div_level", "0")
7753
7754 local function increment_div_level(increment)
7755 local function update_div_level(s, i, current_level) -- luacheck: ignore s i
7756 current_level = tonumber(current_level)
7757 local next_level = tostring(current_level + increment)
7758 return true, next_level
7759 end
7760
7761 return Cg(Cmt(Cb("div_level"), update_div_level)
7762 , "div_level")
7763 end
7764
7765 local FencedDiv = fenced_div_begin
7766 / function (infostring)
7767 local attr = lpeg.match(Ct(parsers.attributes), infostring)
7768 if attr == nil then
7769 attr = {"." .. infostring}
7770 end
7771 return attr
7772 end
7773 / writer.div_begin
7774 * increment_div_level(1)
7775 * parsers.skipblanklines
7776 * Ct((V("Block") - fenced_div_end)^-1
7777 * (parsers.blanklines
7778 / function()
7779 return writer.interblocksep
7780 end
7781 * (V("Block") - fenced_div_end))^0)
7782 * parsers.skipblanklines
7783 * fenced_div_end * increment_div_level(-1)
7784 * (Cc("") / writer.div_end)
7785
7786 self.insert_pattern("Block after Verbatim",
7787 FencedDiv, "FencedDiv")
7788
7789 self.add_special_character(":")
7790

```



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

```
7791 local function check_div_level(s, i, current_level) -- luacheck: ignore s i
7792 current_level = tonumber(current_level)
7793 return current_level > 0
7794 end
7795
7796 local is_inside_div = Cmt(Cb("div_level"), check_div_level)
7797 local fencestart = is_inside_div * fenced_div_end
7798
7799 self.update_rule("BlockquoteExceptions", function(previous_pattern)
7800 if previous_pattern == nil then
7801 previous_pattern = parsers.BlockquoteExceptions
7802 end
7803 return previous_pattern + fencestart
7804 end)
7805
```

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.

```
7806 if not blank_before_div_fence then
7807 self.update_rule("EndlineExceptions", function(previous_pattern)
7808 if previous_pattern == nil then
7809 previous_pattern = parsers.EndlineExceptions
7810 end
7811 return previous_pattern + fencestart
7812 end)
7813 end
7814 end
7815 }
7816 end
```

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

```
7817 M.extensions.header_attributes = function()
7818 return {
7819 name = "built-in header_attributes syntax extension",
7820 extend_writer = function()
7821 end, extend_reader = function(self)
7822 local parsers = self.parsers
7823 local writer = self.writer
7824
7825 local AtxHeading = Cg(parsers.heading_start, "level")
7826 * parsers.optionalspace
7827 * (C(((parsers.linechar
```

```

7828 - ((parsers.hash^1
7829 * parsers.optionalspace
7830 * parsers.attributes^-1
7831 + parsers.attributes)
7832 * parsers.optionalspace
7833 * parsers.newline))
7834 * (parsers.linechar
7835 - parsers.hash
7836 - parsers.lbrace)^0)^1)
7837 / self.parser_functions.parse_inlines)
7838 * Cg(Ct(parsers.newline
7839 + (parsers.hash^1
7840 * parsers.optionalspace
7841 * parsers.attributes^-1
7842 + parsers.attributes)
7843 * parsers.optionalspace
7844 * parsers.newline), "attributes")
7845 * Cb("level")
7846 * Cb("attributes")
7847 / writer.heading
7848
7849 local SetextHeading = #(parsers.line * S("--"))
7850 * (C(((parsers.linechar
7851 - (parsers.attributes
7852 * parsers.optionalspace
7853 * parsers.newline))
7854 * (parsers.linechar
7855 - parsers.lbrace)^0)^1)
7856 / self.parser_functions.parse_inlines)
7857 * Cg(Ct(parsers.newline
7858 + (parsers.attributes
7859 * parsers.optionalspace
7860 * parsers.newline)), "attributes")
7861 * parsers.heading_level
7862 * Cb("attributes")
7863 * parsers.optionalspace
7864 * parsers.newline
7865 / writer.heading
7866
7867 local Heading = AtxHeading + SetextHeading
7868 self.update_rule("Heading", Heading)
7869 end
7870 }
7871 end

```

**3.1.6.9 Inline Code Attributes** The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```
7872 M.extensions.inline_code_attributes = function()
7873 return {
7874 name = "built-in inline_code_attributes syntax extension",
7875 extend_writer = function()
7876 end, extend_reader = function(self)
7877 local writer = self.writer
7878
7879 local CodeWithAttributes = parsers.inticks
7880 * Ct(parsers.attributes)
7881 / writer.code
7882
7883 self.insert_pattern("Inline before Code",
7884 CodeWithAttributes,
7885 "CodeWithAttributes")
7886 end
7887 }
7888 end
```

**3.1.6.10 Line Blocks** The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
7889 M.extensions.line_blocks = function()
7890 return {
7891 name = "built-in line_blocks syntax extension",
7892 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.

```
7893 function self.lineblock(lines)
7894 if not self.is_writing then return "" end
7895 local buffer = {}
7896 for i = 1, #lines - 1 do
7897 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
7898 end
7899 buffer[#buffer + 1] = lines[#lines]
7900
7901 return {"\\markdownRendererLineBlockBegin\n"
7902 ,buffer,
7903 "\n\\markdownRendererLineBlockEnd "}
7904 end
7905 end, extend_reader = function(self)
7906 local parsers = self.parsers
7907 local writer = self.writer
7908
7909 local LineBlock = Ct(
```

```

7910 (Cs(
7911 ((parsers.pipe * parsers.space)/""
7912 * ((parsers.space)/entities.char_entity("nbsp"))^0
7913 * parsers.linechar^0 * (parsers.newline/"")
7914 * (-parsers.pipe
7915 * (parsers.space^1/" ")
7916 * parsers.linechar^1
7917 * (parsers.newline/"")
7918)^0
7919 * (parsers.blankline/"")^0
7920) / self.parser_functions.parse_inlines)^1) / writer.lineblo
7921
7922 self.insert_pattern("Block after Blockquote",
7923 LineBlock, "LineBlock")
7924 end
7925 }
7926 end

```

**3.1.6.11 Link Attributes** The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

7927 M.extensions.link_attributes = function()
7928 return {
7929 name = "built-in link_attributes syntax extension",
7930 extend_writer = function()
7931 end, extend_reader = function(self)
7932 local parsers = self.parsers
7933 local writer = self.writer
7934 local options = self.options
7935

```

The following patterns define link reference definitions with attributes.

```

7936
7937 local define_reference_parser = parsers.define_reference_parser
7938 * (parsers.spnl
7939 * Ct(parsers.attributes))^~1
7940
7941 local ReferenceWithAttributes = define_reference_parser
7942 * parsers.blankline^1
7943 / self.register_link
7944
7945 self.update_rule("Reference", ReferenceWithAttributes)
7946

```

The following patterns define direct and indirect links with attributes.

```

7947
7948 local function indirect_link(label, sps, tag,
7949 attribute_text,

```

```

7950 attributes)
7951 return writer.defer_call(function()
7952 local r, fallback = self.lookup_reference(label, sps, tag,
7953 attributes)
7954 if r then
7955 return writer.link(
7956 self.parser_functions.parse_inlines_no_link(label),
7957 r.url, r.title, r.attributes)
7958 else
7959 local buf = {fallback}
7960 if attributes then
7961 table.insert(buf, writer.string(attribute_text))
7962 end
7963 return buf
7964 end
7965 end)
7966 end
7967
7968 local DirectLinkWithAttributes = parsers.direct_link
7969 * (Ct(parsers.attributes))^-1
7970 / writer.link
7971
7972 local IndirectLinkWithAttributes = parsers.indirect_link
7973 * (C(Ct(parsers.attributes)))^-1
7974 / indirect_link
7975
7976 local LinkWithAttributes = DirectLinkWithAttributes
7977 + IndirectLinkWithAttributes
7978

```

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.

```

7979 self.update_rule("Link", LinkWithAttributes)
7980

```

The following patterns define direct and indirect images with attributes.

```

7981
7982 local function indirect_image(label, sps, tag,
7983 attribute_text,
7984 attributes)
7985 return writer.defer_call(function()
7986 local r, fallback = self.lookup_reference(label, sps, tag,
7987 attributes)

```

```

7988 if r then
7989 return writer.image(writer.string(label),
7990 r.url, r.title, r.attributes)
7991 else
7992 local buf = {"!", fallback}
7993 if attributes then
7994 table.insert(buf, writer.string(attribute_text))
7995 end
7996 return buf
7997 end
7998 end)
7999 end
8000
8001 local DirectImageWithAttributes = parsers.direct_image
8002 * Ct(parsers.attributes)
8003 / writer.image
8004
8005 local IndirectImageWithAttributes = parsers.indirect_image
8006 * C(Ct(parsers.attributes))
8007 / indirect_image
8008
8009 local ImageWithAttributes = DirectImageWithAttributes
8010 + IndirectImageWithAttributes
8011
8012 self.insert_pattern("Inline before Image",
8013 ImageWithAttributes,
8014 "ImageWithAttributes")
8015

```

The following patterns define autolinks with attributes.

```

8016
8017 local AutoLinkUrlWithAttributes
8018 = parsers.auto_link_url
8019 * Ct(parsers.attributes)
8020 / self.auto_link_url
8021
8022 self.insert_pattern("Inline before AutoLinkUrl",
8023 AutoLinkUrlWithAttributes,
8024 "AutoLinkUrlWithAttributes")
8025
8026 local AutoLinkEmailWithAttributes
8027 = parsers.auto_link_email
8028 * Ct(parsers.attributes)
8029 / self.auto_link_email
8030
8031 self.insert_pattern("Inline before AutoLinkEmail",
8032 AutoLinkEmailWithAttributes,
8033 "AutoLinkEmailWithAttributes")

```

```

8034
8035 if options.relativeReferences then
8036
8037 local AutoLinkRelativeReferenceWithAttributes
8038 = parsers.auto_link_relative_reference
8039 * Ct(parsers.attributes)
8040 / self.auto_link_url
8041
8042 self.insert_pattern(
8043 "Inline before AutoLinkRelativeReference",
8044 AutoLinkRelativeReferenceWithAttributes,
8045 "AutoLinkRelativeReferenceWithAttributes")
8046
8047 end
8048
8049 end
8050 }
8051 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.

```

8052 M.extensions.notes = function(notes, inline_notes)
8053 assert(notes or inline_notes)
8054 return {
8055 name = "built-in notes syntax extension",
8056 extend_writer = function(self)

```

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

```

8057 function self.note(s)
8058 return {"\\markdownRendererNote{" ,s,"}"}
8059 end
8060 end, extend_reader = function(self)
8061 local parsers = self.parsers
8062 local writer = self.writer
8063
8064 if inline_notes then
8065 local InlineNote
8066 = parsers.circumflex
8067 * (parsers.tag / self.parser_functions.parse_inlines_no_inline_no
8068 / writer.note
8069
8070 self.insert_pattern("Inline after Emph",
8071 InlineNote, "InlineNote")

```

```

8072 end
8073 if notes then
8074 local function strip_first_char(s)
8075 return s:sub(2)
8076 end
8077
8078 local RawNoteRef
8079 = #(parsers.lbracket * parsers.circumflex)
8080 * parsers.tag / strip_first_char
8081
8082 local rawnotes = {}
8083
8084 -- like indirect_link
8085 local function lookup_note(ref)
8086 return writer.defer_call(function()
8087 local found = rawnotes[self.normalize_tag(ref)]
8088 if found then
8089 return writer.note(
8090 self.parser_functions.parse_blocks_nested(found))
8091 else
8092 return {"[",
8093 self.parser_functions.parse_inlines("^" .. ref), "]" }
8094 end
8095 end)
8096 end
8097
8098 local function register_note(ref,rawnote)
8099 rawnotes[self.normalize_tag(ref)] = rawnote
8100 return ""
8101 end
8102
8103 local NoteRef = RawNoteRef / lookup_note
8104
8105 local NoteBlock
8106 = parsers.leader * RawNoteRef * parsers.colon
8107 * parsers.spnl * parsers.indented_blocks(parsers.chunk)
8108 / register_note
8109
8110 local Blank = NoteBlock + parsers.Blank
8111 self.update_rule("Blank", Blank)
8112
8113 self.insert_pattern("Inline after Emph",
8114 NoteRef, "NoteRef")
8115 end
8116
8117 self.add_special_character("^")
8118 end

```



```
8119 }
8120 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.

```
8121 M.extensions.pipe_tables = function(table_captions)
8122
8123 local function make_pipe_table_rectangular(rows)
8124 local num_columns = #rows[2]
8125 local rectangular_rows = {}
8126 for i = 1, #rows do
8127 local row = rows[i]
8128 local rectangular_row = {}
8129 for j = 1, num_columns do
8130 rectangular_row[j] = row[j] or ""
8131 end
8132 table.insert(rectangular_rows, rectangular_row)
8133 end
8134 return rectangular_rows
8135 end
8136
8137 local function pipe_table_row(allow_empty_first_column
8138 , nonempty_column
8139 , column_separator
8140 , column)
8141 local row_beginning
8142 if allow_empty_first_column then
8143 row_beginning = -- empty first column
8144 #(parsers.spacechar^4
8145 * column_separator)
8146 * parsers.optionalspace
8147 * column
8148 * parsers.optionalspace
8149 -- non-empty first column
8150 + parsers.nonindentspace
8151 * nonempty_column^-1
8152 * parsers.optionalspace
8153 else
8154 row_beginning = parsers.nonindentspace
8155 * nonempty_column^-1
8156 * parsers.optionalspace
8157 end
8158
8159 return Ct(row_beginning
```

```

8160 * (-- single column with no leading pipes
8161 #(column_separator
8162 * parsers.optionalspace
8163 * parsers.newline)
8164 * column_separator
8165 * parsers.optionalspace
8166 -- single column with leading pipes or
8167 -- more than a single column
8168 + (column_separator
8169 * parsers.optionalspace
8170 * column
8171 * parsers.optionalspace)^1
8172 * (column_separator
8173 * parsers.optionalspace)^-1))
8174 end
8175
8176 return {
8177 name = "built-in pipe_tables syntax extension",
8178 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.

```

8179 function self.table(rows, caption)
8180 if not self.is_writing then return "" end
8181 local buffer = {"\\markdownRendererTable{",
8182 caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}"}
8183 local temp = rows[2] -- put alignments on the first row
8184 rows[2] = rows[1]
8185 rows[1] = temp
8186 for i, row in ipairs(rows) do
8187 table.insert(buffer, "{")
8188 for _, column in ipairs(row) do
8189 if i > 1 then -- do not use braces for alignments
8190 table.insert(buffer, "{")
8191 end
8192 table.insert(buffer, column)
8193 if i > 1 then
8194 table.insert(buffer, "}")
8195 end
8196 end
8197 table.insert(buffer, "}")
8198 end
8199 return buffer
8200 end
8201 end, extend_reader = function(self)
8202 local parsers = self.parsers

```

```

8203 local writer = self.writer
8204
8205 local table_hline_separator = parsers.pipe + parsers.plus
8206
8207 local table_hline_column = (parsers.dash
8208 - #(parsers.dash
8209 * (parsers.spacechar
8210 + table_hline_separator
8211 + parsers.newline)))^1
8212 * (parsers.colon * Cc("r")
8213 + parsers.dash * Cc("d"))
8214 + parsers.colon
8215 * (parsers.dash
8216 - #(parsers.dash
8217 * (parsers.spacechar
8218 + table_hline_separator
8219 + parsers.newline)))^1
8220 * (parsers.colon * Cc("c")
8221 + parsers.dash * Cc("l"))
8222
8223 local table_hline = pipe_table_row(false
8224 , table_hline_column
8225 , table_hline_separator
8226 , table_hline_column)
8227
8228 local table_caption_beginning = parsers.skipblanklines
8229 * parsers.nonindentSPACE
8230 * (P("Table")^-1 * parsers.colon)
8231 * parsers.optionalSPACE
8232
8233 local table_row = pipe_table_row(true
8234 , (C((parsers.linechar - parsers.pipe)^1)
8235 / self.parser_functions.parse_inlines)
8236 , parsers.pipe
8237 , (C((parsers.linechar - parsers.pipe)^0)
8238 / self.parser_functions.parse_inlines))
8239
8240 local table_caption
8241 if table_captions then
8242 table_caption = #table_caption_beginning
8243 * table_caption_beginning
8244 * Ct(parsers.IndentedInline^1)
8245 * parsers.newline
8246 else
8247 table_caption = parsers.fail
8248 end
8249

```

```

8250 local PipeTable = Ct(table_row * parsers.newline
8251 * table_hline
8252 * (parsers.newline * table_row)^0)
8253 / make_pipe_table_rectangular
8254 * table_caption^-1
8255 / writer.table
8256
8257 self.insert_pattern("Block after Blockquote",
8258 PipeTable, "PipeTable")
8259 end
8260 }
8261 end

```

**3.1.6.14 Raw Attributes** The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

8262 M.extensions.raw_inline = function()
8263 return {
8264 name = "built-in raw_inline syntax extension",
8265 extend_writer = function(self)
8266 local options = self.options
8267

```

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

```

8268 function self.rawInline(s, attr)
8269 if not self.is_writing then return "" end
8270 local name = util.cache_verbatim(options.cacheDir, s)
8271 return {"\\markdownRendererInputRawInline{" ,
8272 name,"}{" , self.string(attr),"}"}
8273 end
8274 end, extend_reader = function(self)
8275 local writer = self.writer
8276
8277 local RawInline = parsers.inticks
8278 * parsers.raw_attribute
8279 / writer.rawInline
8280
8281 self.insert_pattern("Inline before Code",
8282 RawInline, "RawInline")
8283 end
8284 }
8285 end

```

**3.1.6.15 Strike-Through** The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

8286 M.extensions.strike_through = function()

```

```

8287 return {
8288 name = "built-in strike_through syntax extension",
8289 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.

```

8290 function self.strike_through(s)
8291 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
8292 end
8293 end, extend_reader = function(self)
8294 local parsers = self.parsers
8295 local writer = self.writer
8296
8297 local StrikeThrough = (
8298 parsers.between(parsers.Inline, parsers.doubletildes,
8299 parsers.doubletildes)
8300) / writer.strike_through
8301
8302 self.insert_pattern("Inline after Emph",
8303 StrikeThrough, "StrikeThrough")
8304
8305 self.add_special_character("~")
8306 end
8307 }
8308 end

```

**3.1.6.16 Subscripts** The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

8309 M.extensions.subscripts = function()
8310 return {
8311 name = "built-in subscripts syntax extension",
8312 extend_writer = function(self)

```

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

```

8313 function self.subscript(s)
8314 return {"\\markdownRendererSubscript{" ,s,"}"}
8315 end
8316 end, extend_reader = function(self)
8317 local parsers = self.parsers
8318 local writer = self.writer
8319
8320 local Subscript = (
8321 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
8322) / writer.subscript
8323
8324 self.insert_pattern("Inline after Emph",

```

```

8325 Subscript, "Subscript")
8326
8327 self.add_special_character("~")
8328 end
8329 }
8330 end

```

**3.1.6.17 Superscripts** The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

8331 M.extensions.superscripts = function()
8332 return {
8333 name = "built-in superscripts syntax extension",
8334 extend_writer = function(self)

```

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

```

8335 function self.superscript(s)
8336 return {"\\markdownRendererSuperscript{" ,s,"}"}
8337 end
8338 end, extend_reader = function(self)
8339 local parsers = self.parsers
8340 local writer = self.writer
8341
8342 local Superscript = (
8343 parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
8344) / writer.superscript
8345
8346 self.insert_pattern("Inline after Emph",
8347 Superscript, "Superscript")
8348
8349 self.add_special_character("^")
8350 end
8351 }
8352 end

```

**3.1.6.18 Tex Math** The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

8353 M.extensions.tex_math = function(tex_math_dollars,
8354 tex_math_single_backslash,
8355 tex_math_double_backslash)
8356 return {
8357 name = "built-in tex_math syntax extension",
8358 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.

```

8359 function self.display_math(s)
8360 if not self.is_writing then return "" end
8361 return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
8362 end

```

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

```

8363 function self.inline_math(s)
8364 if not self.is_writing then return "" end
8365 return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
8366 end
8367 end, extend_reader = function(self)
8368 local parsers = self.parsers
8369 local writer = self.writer
8370
8371 local function between(p, starter, ender)
8372 return (starter * C(p * (p - ender)^0) * ender)
8373 end
8374
8375 local allowed_before_closing = B(parsers.backslash * parsers.any
8376 + parsers.any * (parsers.nonspacechar - parsers

```

The following patterns implement the Pandoc dollar math syntax extension.

```

8377 local dollar_math_content = parsers.backslash^-1
8378 * parsers.any
8379 - parsers.blankline^2
8380 - parsers.dollar
8381
8382 local inline_math_opening_dollars = parsers.dollar
8383 * #(parsers.nonspacechar)
8384
8385 local inline_math_closing_dollars = allowed_before_closing
8386 * parsers.dollar
8387 * -#(parsers.digit)
8388
8389 local inline_math_dollars = between(C(dollar_math_content),
8390 inline_math_opening_dollars,
8391 inline_math_closing_dollars)
8392
8393 local display_math_opening_dollars = parsers.dollar
8394 * parsers.dollar
8395
8396 local display_math_closing_dollars = parsers.dollar
8397 * parsers.dollar
8398
8399 local display_math_dollars = between(C(dollar_math_content),
8400 display_math_opening_dollars,
8401 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```
8402 local backslash_math_content = parsers.any
8403 - parsers.blankline^2
```

The following patterns implement the Pandoc double backslash math syntax extension.

```
8404 local inline_math_opening_double = parsers.backslash
8405 * parsers.backslash
8406 * parsers.lparent
8407 * #(parsers.nonspacechar)
8408
8409 local inline_math_closing_double = allowed_before_closing
8410 * parsers.backslash
8411 * parsers.backslash
8412 * parsers.rparent
8413
8414 local inline_math_double = between(C(backslash_math_content),
8415 inline_math_opening_double,
8416 inline_math_closing_double)
8417
8418 local display_math_opening_double = parsers.backslash
8419 * parsers.backslash
8420 * parsers.lbracket
8421
8422 local display_math_closing_double = allowed_before_closing
8423 * parsers.backslash
8424 * parsers.backslash
8425 * parsers.rbracket
8426
8427 local display_math_double = between(C(backslash_math_content),
8428 display_math_opening_double,
8429 display_math_closing_double)
```

The following patterns implement the Pandoc single backslash math syntax extension.

```
8430 local inline_math_opening_single = parsers.backslash
8431 * parsers.lparent
8432 * #(parsers.nonspacechar)
8433
8434 local inline_math_closing_single = allowed_before_closing
8435 * parsers.backslash
8436 * parsers.rparent
8437
8438 local inline_math_single = between(C(backslash_math_content),
8439 inline_math_opening_single,
8440 inline_math_closing_single)
8441
```



```

8442 local display_math_opening_single = parsers.backslash
8443 * parsers.lbracket
8444
8445 local display_math_closing_single = allowed_before_closing
8446 * parsers.backslash
8447 * parsers.rbracket
8448
8449 local display_math_single = between(C(backslash_math_content),
8450 display_math_opening_single,
8451 display_math_closing_single)
8452
8453 local display_math = parsers.fail
8454
8455 local inline_math = parsers.fail
8456
8457 if tex_math_dollars then
8458 display_math = display_math + display_math_dollars
8459 inline_math = inline_math + inline_math_dollars
8460 end
8461
8462 if tex_math_double_backslash then
8463 display_math = display_math + display_math_double
8464 inline_math = inline_math + inline_math_double
8465 end
8466
8467 if tex_math_single_backslash then
8468 display_math = display_math + display_math_single
8469 inline_math = inline_math + inline_math_single
8470 end
8471
8472 local TexMath = display_math / writer.display_math
8473 + inline_math / writer.inline_math
8474
8475 self.insert_pattern("Inline after Emph",
8476 TexMath, "TexMath")
8477
8478 if tex_math_dollars then
8479 self.add_special_character("$")
8480 end
8481
8482 if tex_math_single_backslash or tex_math_double_backslash then
8483 self.add_special_character("\\")
8484 self.add_special_character("[")
8485 self.add_special_character("]")
8486 self.add_special_character("(")
8487 self.add_special_character("(")
8488 end

```

```

8489 end
8490 }
8491 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.

```

8492 M.extensions.jekyll_data = function(expect_jekyll_data)
8493 return {
8494 name = "built-in jekyll_data syntax extension",
8495 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`.

```

8496 function self.jekyllData(d, t, p)
8497 if not self.is_writing then return "" end
8498
8499 local buf = {}
8500
8501 local keys = {}
8502 for k, _ in pairs(d) do
8503 table.insert(keys, k)
8504 end
8505 table.sort(keys)
8506
8507 if not p then
8508 table.insert(buf, "\\markdownRendererJekyllDataBegin")
8509 end
8510
8511 if #d > 0 then
8512 table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
8513 table.insert(buf, self.identifier(p or "null"))
8514 table.insert(buf, "}{")
8515 table.insert(buf, #keys)
8516 table.insert(buf, "}")
8517 else
8518 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
8519 table.insert(buf, self.identifier(p or "null"))
8520 table.insert(buf, "}{")
8521 table.insert(buf, #keys)
8522 table.insert(buf, "}")
8523 end

```

```

8524
8525 for _, k in ipairs(keys) do
8526 local v = d[k]
8527 local typ = type(v)
8528 k = tostring(k or "null")
8529 if typ == "table" and next(v) ~= nil then
8530 table.insert(
8531 buf,
8532 self.jekyllData(v, t, k)
8533)
8534 else
8535 k = self.identifier(k)
8536 v = tostring(v)
8537 if typ == "boolean" then
8538 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
8539 table.insert(buf, k)
8540 table.insert(buf, "}{"")
8541 table.insert(buf, v)
8542 table.insert(buf, "}")
8543 elseif typ == "number" then
8544 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
8545 table.insert(buf, k)
8546 table.insert(buf, "}{"")
8547 table.insert(buf, v)
8548 table.insert(buf, "}")
8549 elseif typ == "string" then
8550 table.insert(buf, "\\markdownRendererJekyllDataString{")
8551 table.insert(buf, k)
8552 table.insert(buf, "}{"")
8553 table.insert(buf, t(v))
8554 table.insert(buf, "}")
8555 elseif typ == "table" then
8556 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
8557 table.insert(buf, k)
8558 table.insert(buf, "}")
8559 else
8560 error(format("Unexpected type %s for value of " ..
8561 "YAML key %s", typ, k))
8562 end
8563 end
8564 end
8565
8566 if #d > 0 then
8567 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
8568 else
8569 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
8570 end

```

```

8571
8572 if not p then
8573 table.insert(buf, "\\markdownRendererJekyllDataEnd")
8574 end
8575
8576 return buf
8577 end
8578 end, extend_reader = function(self)
8579 local parsers = self.parsers
8580 local writer = self.writer
8581
8582 local JekyllData
8583 = Cmt(C((parsers.line - P("----") - P("..."))^0)
8584 , function(s, i, text) -- luacheck: ignore s i
8585 local data
8586 local ran_ok, _ = pcall(function()
8587 -- TODO: Replace with `require("tinyyaml")` in TeX Live
8588 local tinyyaml = require("markdown-tinyyaml")
8589 data = tinyyaml.parse(text, {timestamps=false})
8590 end)
8591 if ran_ok and data ~= nil then
8592 return true, writer.jekyllData(data, function(s)
8593 return self.parser_functions.parse_blocks_nested(s)
8594 end, nil)
8595 else
8596 return false
8597 end
8598 end
8599)
8600
8601 local UnexpectedJekyllData
8602 = P("----")
8603 * parsers.blankline / 0
8604 * #(-parsers.blankline) -- if followed by blank, it's thematic b
8605 * JekyllData
8606 * (P("----") + P("..."))
8607
8608 local ExpectedJekyllData
8609 = (P("----")
8610 * parsers.blankline / 0
8611 * #(-parsers.blankline) -- if followed by blank, it's thematic
8612)^-1
8613 * JekyllData
8614 * (P("----") + P("..."))^-1
8615
8616 self.insert_pattern("Block before Blockquote",
8617 UnexpectedJekyllData, "UnexpectedJekyllData")

```

```

8618 if expect_jekyll_data then
8619 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
8620 end
8621 end
8622 }
8623 end

```

### 3.1.7 Conversion from Markdown to Plain $\text{\TeX}$

The `new` function returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output. See Section 2.1.1.

```

8624 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

8625 options = options or {}
8626 setmetatable(options, { __index = function (_, key)
8627 return defaultOptions[key] end })

```

Apply built-in syntax extensions based on `options`.

```

8628 local extensions = {}
8629
8630 if options.bracketedSpans then
8631 local bracketed_spans_extension = M.extensions.bracketed_spans()
8632 table.insert(extensions, bracketed_spans_extension)
8633 end
8634
8635 if options.contentBlocks then
8636 local content_blocks_extension = M.extensions.content_blocks(
8637 options.contentBlocksLanguageMap)
8638 table.insert(extensions, content_blocks_extension)
8639 end
8640
8641 if options.definitionLists then
8642 local definition_lists_extension = M.extensions.definition_lists(
8643 options.tightLists)
8644 table.insert(extensions, definition_lists_extension)
8645 end
8646
8647 if options.fencedCode then
8648 local fenced_code_extension = M.extensions.fenced_code(
8649 options.blankBeforeCodeFence,
8650 options.fencedCodeAttributes,
8651 options.rawAttribute)
8652 table.insert(extensions, fenced_code_extension)
8653 end
8654
8655 if options.fencedDivs then

```

```

8656 local fenced_div_extension = M.extensions.fenced_divs(
8657 options.blankBeforeDivFence)
8658 table.insert(extensions, fenced_div_extension)
8659 end
8660
8661 if options.headerAttributes then
8662 local header_attributes_extension = M.extensions.header_attributes()
8663 table.insert(extensions, header_attributes_extension)
8664 end
8665
8666 if options.inlineCodeAttributes then
8667 local inline_code_attributes_extension =
8668 M.extensions.inline_code_attributes()
8669 table.insert(extensions, inline_code_attributes_extension)
8670 end
8671
8672 if options.jekyllData then
8673 local jekyll_data_extension = M.extensions.jekyll_data(
8674 options.expectJekyllData)
8675 table.insert(extensions, jekyll_data_extension)
8676 end
8677
8678 if options.linkAttributes then
8679 local link_attributes_extension =
8680 M.extensions.link_attributes()
8681 table.insert(extensions, link_attributes_extension)
8682 end
8683
8684 if options.lineBlocks then
8685 local line_block_extension = M.extensions.line_blocks()
8686 table.insert(extensions, line_block_extension)
8687 end
8688
8689 if options.pipeTables then
8690 local pipe_tables_extension = M.extensions.pipe_tables(
8691 options.tableCaptions)
8692 table.insert(extensions, pipe_tables_extension)
8693 end
8694
8695 if options.rawAttribute then
8696 local raw_inline_extension = M.extensions.raw_inline()
8697 table.insert(extensions, raw_inline_extension)
8698 end
8699
8700 if options.strikeThrough then
8701 local strike_through_extension = M.extensions.strike_through()
8702 table.insert(extensions, strike_through_extension)

```

```

8703 end
8704
8705 if options.subscripts then
8706 local subscript_extension = M.extensions.subscripts()
8707 table.insert(extensions, subscript_extension)
8708 end
8709
8710 if options.superscripts then
8711 local superscript_extension = M.extensions.superscripts()
8712 table.insert(extensions, superscript_extension)
8713 end
8714
8715 if options.texMathDollars or
8716 options.texMathSingleBackslash or
8717 options.texMathDoubleBackslash then
8718 local tex_math_extension = M.extensions.tex_math(
8719 options.texMathDollars,
8720 options.texMathSingleBackslash,
8721 options.texMathDoubleBackslash)
8722 table.insert(extensions, tex_math_extension)
8723 end
8724

```

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

```

8725 if options.footnotes or options.inlineFootnotes or
8726 options.notes or options.inlineNotes then
8727 local notes_extension = M.extensions.notes(
8728 options.footnotes or options.notes,
8729 options.inlineFootnotes or options.inlineNotes)
8730 table.insert(extensions, notes_extension)
8731 end
8732
8733 if options.citations then
8734 local citations_extension = M.extensions.citations(options.citationNbsps)
8735 table.insert(extensions, citations_extension)
8736 end
8737
8738 if options.fancyLists then
8739 local fancy_lists_extension = M.extensions.fancy_lists()
8740 table.insert(extensions, fancy_lists_extension)
8741 end

```

Apply user-defined syntax extensions based on [options.extensions](#).

```

8742 for _, user_extension_filename in ipairs(options.extensions) do
8743 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

8744 local pathname = util.lookup_files(filename)
8745 local input_file = assert(io.open(pathname, "r"),
8746 [[Could not open user-defined syntax extension "]]
8747 .. pathname .. [[" for reading]])
8748 local input = assert(input_file:read("*a"))
8749 assert(input_file:close())
8750 local user_extension, err = load([[
8751 local sandbox = {}
8752 setmetatable(sandbox, {__index = _G})
8753 _ENV = sandbox
8754]]) .. input)()
8755 assert(user_extension,
8756 [[Failed to compile user-defined syntax extension "]]
8757 .. pathname .. [[":]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

8758 assert(user_extension.api_version ~= nil,
8759 [[User-defined syntax extension "]] .. pathname
8760 .. [[" does not specify mandatory field "api_version"]])
8761 assert(type(user_extension.api_version) == "number",
8762 [[User-defined syntax extension "]] .. pathname
8763 .. [[" specifies field "api_version" of type "]]
8764 .. type(user_extension.api_version)
8765 .. [[" but "number" was expected]])
8766 assert(user_extension.api_version > 0
8767 and user_extension.api_version <= metadata.user_extension_api_version,
8768 [[User-defined syntax extension "]] .. pathname
8769 .. [[" uses syntax extension API version "]]
8770 .. user_extension.api_version .. [[but markdown.lua]]
8771 .. metadata.version .. [[uses API version]]
8772 .. metadata.user_extension_api_version
8773 .. [[, which is incompatible]])
8774
8775 assert(user_extension.grammar_version ~= nil,
8776 [[User-defined syntax extension "]] .. pathname
8777 .. [[" does not specify mandatory field "grammar_version"]])
8778 assert(type(user_extension.grammar_version) == "number",
8779 [[User-defined syntax extension "]] .. pathname
8780 .. [[" specifies field "grammar_version" of type "]]
8781 .. type(user_extension.grammar_version)
8782 .. [[" but "number" was expected]])
8783 assert(user_extension.grammar_version == metadata.grammar_version,
8784 [[User-defined syntax extension "]] .. pathname
8785 .. [[" uses grammar version "]] .. user_extension.grammar_version
8786 .. [[but markdown.lua]] .. metadata.version
8787 .. [[uses grammar version]] .. metadata.grammar_version
8788 .. [[, which is incompatible]])
8789

```



```

8790 assert(user_extension.finalize_grammar ~= nil,
8791 [[User-defined syntax extension "]] .. pathname
8792 .. [[" does not specify mandatory "finalize_grammar" field]])
8793 assert(type(user_extension.finalize_grammar) == "function",
8794 [[User-defined syntax extension "]] .. pathname
8795 .. [[" specifies field "finalize_grammar" of type "]]
8796 .. type(user_extension.finalize_grammar)
8797 .. [[" 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.)

```

8798 local extension = {
8799 name = [[user-defined "]] .. pathname .. [[" syntax extension]],
8800 extend_reader = user_extension.finalize_grammar,
8801 extend_writer = function() end,
8802 }
8803 return extension
8804 end)(user_extension_filename)
8805 table.insert(extensions, user_extension)
8806 end

```

Produce and return a conversion function from markdown to plain TeX.

```

8807 local writer = M.writer.new(options)
8808 local reader = M.reader.new(writer, options)
8809 local convert = reader.finalize_grammar(extensions)
8810
8811 return convert
8812 end
8813
8814 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.

```

8815
8816 local input
8817 if input_filename then
8818 local input_file = assert(io.open(input_filename, "r"),
8819 [[Could not open file "]] .. input_filename .. [[" for reading]])
8820 input = assert(input_file:read("*a"))
8821 assert(input_file:close())
8822 else
8823 input = assert(io.read("*a"))
8824 end
8825

```

First, ensure that the `options.cacheDir` directory exists.

```

8826 local lfs = require("lfs")
8827 if options.cacheDir and not lfs.isdir(options.cacheDir) then
8828 assert(lfs.mkdir(options["cacheDir"]))
8829 end

```

If Kpathsea has not been loaded before or if Lua $\TeX$  has not yet been initialized, configure Kpathsea on top of loading it.

```

8830 local kpse
8831 (function()
8832 local should_initialize = package.loaded.kpse == nil
8833 or tex.initialize ~= nil
8834 local ran_ok
8835 ran_ok, kpse = pcall(require, "kpse")
8836 if ran_ok and should_initialize then
8837 kpse.set_program_name("luatex")
8838 end
8839 end)()
8840 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.

```

8841 if metadata.version ~= md.metadata.version then
8842 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
8843 "markdown.lua " .. md.metadata.version .. ".")
8844 end
8845 local convert = md.new(options)
8846 local output = convert(input)
8847
8848 if output_filename then
8849 local output_file = assert(io.open(output_filename, "w"),
8850 [[Could not open file]] .. output_filename .. [[for writing]])
8851 assert(output_file:write(output))
8852 assert(output_file:close())
8853 else
8854 assert(io.write(output))
8855 end

```

## 3.2 Plain $\TeX$ Implementation

The plain  $\TeX$  implementation provides macros for the interfacing between  $\TeX$  and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain  $\TeX$  exposed by the plain  $\TeX$  interface (see Section 2.2).

### 3.2.1 Logging Facilities

```

8856 \ifx\markdownInfo\undefined

```

```

8857 \def\markdownInfo#1{%
8858 \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
8859 \fi
8860 \ifx\markdownWarning\undefined
8861 \def\markdownWarning#1{%
8862 \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
8863 \fi
8864 \ifx\markdownError\undefined
8865 \def\markdownError#1#2{%
8866 \errhelp{#2.}}%
8867 \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
8868 \fi

```

### 3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

8869 \def\markdownRendererInterblockSeparatorPrototype{\par}%
8870 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
8871 \let\markdownRendererEllipsisPrototype\dots
8872 \def\markdownRendererNbspPrototype{~}%
8873 \def\markdownRendererLeftBracePrototype{\char`}%
8874 \def\markdownRendererRightBracePrototype{\char`}%
8875 \def\markdownRendererDollarSignPrototype{\char`$}%
8876 \def\markdownRendererPercentSignPrototype{\char`}%
8877 \def\markdownRendererAmpersandPrototype{\&}%
8878 \def\markdownRendererUnderscorePrototype{\char`_%}
8879 \def\markdownRendererHashPrototype{\char`#}%
8880 \def\markdownRendererCircumflexPrototype{\char`^}%
8881 \def\markdownRendererBackslashPrototype{\char`\}%
8882 \def\markdownRendererTildePrototype{\char`~}%
8883 \def\markdownRendererPipePrototype{|}%
8884 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
8885 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
8886 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
8887 \markdownInput{#3}}%
8888 \def\markdownRendererContentBlockOnlineImagePrototype{%
8889 \markdownRendererImage}%
8890 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
8891 \markdownRendererInputFencedCode{#3}{#2}}%
8892 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
8893 \def\markdownRendererUlBeginPrototype{}%
8894 \def\markdownRendererUlBeginTightPrototype{}%
8895 \def\markdownRendererUlItemPrototype{}%
8896 \def\markdownRendererUlItemEndPrototype{}%
8897 \def\markdownRendererUlEndPrototype{}%
8898 \def\markdownRendererUlEndTightPrototype{}%
8899 \def\markdownRendererOlBeginPrototype{}%
8900 \def\markdownRendererOlBeginTightPrototype{}%

```

```

8901 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
8902 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
8903 \def\markdownRendererOlItemPrototype{}%
8904 \def\markdownRendererOlItemWithNumberPrototype#1{}%
8905 \def\markdownRendererOlItemEndPrototype{}%
8906 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
8907 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}
8908 \def\markdownRendererFancyOlItemEndPrototype{}%
8909 \def\markdownRendererOlEndPrototype{}%
8910 \def\markdownRendererOlEndTightPrototype{}%
8911 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
8912 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
8913 \def\markdownRendererDlBeginPrototype{}%
8914 \def\markdownRendererDlBeginTightPrototype{}%
8915 \def\markdownRendererDlItemPrototype#1{#1}%
8916 \def\markdownRendererDlItemEndPrototype{}%
8917 \def\markdownRendererDlDefinitionBeginPrototype{}%
8918 \def\markdownRendererDlDefinitionEndPrototype{\par}%
8919 \def\markdownRendererDlEndPrototype{}%
8920 \def\markdownRendererDlEndTightPrototype{}%
8921 \def\markdownRendererEmphasisPrototype#1{\it#1}%
8922 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
8923 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
8924 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
8925 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
8926 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
8927 \def\markdownRendererInputVerbatimPrototype#1{%
8928 \par{\tt\input#1\relax}\par}%
8929 \def\markdownRendererInputFencedCodePrototype#1#2{%
8930 \markdownRendererInputVerbatim{#1}}%
8931 \def\markdownRendererHeadingOnePrototype#1{#1}%
8932 \def\markdownRendererHeadingTwoPrototype#1{#1}%
8933 \def\markdownRendererHeadingThreePrototype#1{#1}%
8934 \def\markdownRendererHeadingFourPrototype#1{#1}%
8935 \def\markdownRendererHeadingFivePrototype#1{#1}%
8936 \def\markdownRendererHeadingSixPrototype#1{#1}%
8937 \def\markdownRendererThematicBreakPrototype{}%
8938 \def\markdownRendererNotePrototype#1{#1}%
8939 \def\markdownRendererCitePrototype#1{}%
8940 \def\markdownRendererTextCitePrototype#1{}%
8941 \def\markdownRendererTickedBoxPrototype{[X]}%
8942 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
8943 \def\markdownRendererUntickedBoxPrototype{[]}%
8944 \def\markdownRendererStrikeThroughPrototype#1{#1}%
8945 \def\markdownRendererSuperscriptPrototype#1{#1}%
8946 \def\markdownRendererSubscriptPrototype#1{#1}%
8947 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%

```

```

8948 \def\markdownRendererInlineMathPrototype#1{${#1$}}%
8949 \ExplSyntaxOn
8950 \cs_gset:Npn
8951 \markdownRendererHeaderAttributeContextBeginPrototype
8952 {
8953 \group_begin:
8954 \color_group_begin:
8955 }
8956 \cs_gset:Npn
8957 \markdownRendererHeaderAttributeContextEndPrototype
8958 {
8959 \color_group_end:
8960 \group_end:
8961 }
8962 \cs_gset_eq:NN
8963 \markdownRendererBracketedSpanAttributeContextBeginPrototype
8964 \markdownRendererHeaderAttributeContextBeginPrototype
8965 \cs_gset_eq:NN
8966 \markdownRendererBracketedSpanAttributeContextEndPrototype
8967 \markdownRendererHeaderAttributeContextEndPrototype
8968 \cs_gset_eq:NN
8969 \markdownRendererFencedDivAttributeContextBeginPrototype
8970 \markdownRendererHeaderAttributeContextBeginPrototype
8971 \cs_gset_eq:NN
8972 \markdownRendererFencedDivAttributeContextEndPrototype
8973 \markdownRendererHeaderAttributeContextEndPrototype
8974 \cs_gset_eq:NN
8975 \markdownRendererFencedCodeAttributeContextBeginPrototype
8976 \markdownRendererHeaderAttributeContextBeginPrototype
8977 \cs_gset_eq:NN
8978 \markdownRendererFencedCodeAttributeContextEndPrototype
8979 \markdownRendererHeaderAttributeContextEndPrototype
8980 \cs_gset:Npn
8981 \markdownRendererReplacementCharacterPrototype
8982 {
8983 % TODO: Replace with \codepoint_generate:nn in TeX Live 2023
8984 \sys_if_engine_pdftex:TF
8985 { ^^ef^^bf^^bd }
8986 { ^^^^fffd }
8987 }
8988 \ExplSyntaxOff
8989 \def\markdownRendererSectionBeginPrototype{}%
8990 \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.

```

8991 \ExplSyntaxOn
8992 \cs_new:Nn
8993 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
8994 {
8995 \str_case:nn
8996 { #2 }
8997 {
8998 { md } { \markdownInput{#1} }
8999 { tex } { \markdownEscape{#1} \unskip }
9000 }
9001 }
9002 \cs_new:Nn
9003 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
9004 {
9005 \str_case:nn
9006 { #2 }
9007 {
9008 { md } { \markdownInput{#1} }
9009 { tex } { \markdownEscape{#1} }
9010 }
9011 }
9012 \cs_gset:Npn
9013 \markdownRendererInputRawInlinePrototype#1#2
9014 {
9015 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
9016 { #1 }
9017 { #2 }
9018 }
9019 \cs_gset:Npn
9020 \markdownRendererInputRawBlockPrototype#1#2
9021 {
9022 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
9023 { #1 }
9024 { #2 }
9025 }
9026 \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_t1` 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  $p$ .

```
9027 \ExplSyntaxOn
9028 \seq_new:N \g_@@_jekyll_data_datatypes_seq
9029 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
9030 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
9031 \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.

```
9032 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
9033 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
9034 {
9035 \seq_if_empty:NF
9036 \g_@@_jekyll_data_datatypes_seq
9037 {
9038 \seq_get_right:NN
9039 \g_@@_jekyll_data_datatypes_seq
9040 \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.

```
9041 \str_if_eq:NNTF
9042 \l_tmpa_tl
9043 \c_@@_jekyll_data_sequence_tl
9044 {
9045 \seq_put_right:Nn
9046 \g_@@_jekyll_data_wildcard_absolute_address_seq
9047 { * }
9048 }
9049 {
9050 \seq_put_right:Nn
9051 \g_@@_jekyll_data_wildcard_absolute_address_seq
9052 { #1 }
9053 }
9054 }
9055 }
```

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.

```
9056 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
9057 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
9058 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
9059 {
9060 \seq_pop_left:NN #1 \l_tmpa_tl
9061 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
9062 \seq_put_left:NV #1 \l_tmpa_tl
9063 }
9064 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
9065 {
9066 \markdown_jekyll_data_concatenate_address:NN
9067 \g_@@_jekyll_data_wildcard_absolute_address_seq
9068 \g_@@_jekyll_data_wildcard_absolute_address_tl
9069 \seq_get_right:NN
9070 \g_@@_jekyll_data_wildcard_absolute_address_seq
9071 \g_@@_jekyll_data_wildcard_relative_address_tl
9072 }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:n` and `\markdown_jekyll_data_pop:` macros.

```
9073 \cs_new:Nn \markdown_jekyll_data_push:n
9074 {
9075 \markdown_jekyll_data_push_address_segment:n
```



```

9076 { #1 }
9077 \seq_put_right:NV
9078 \g_@@_jekyll_data_datatypes_seq
9079 #2
9080 \markdown_jekyll_data_update_address_tls:
9081 }
9082 \cs_new:Nn \markdown_jekyll_data_pop:
9083 {
9084 \seq_pop_right:NN
9085 \g_@@_jekyll_data_wildcard_absolute_address_seq
9086 \l_tmpa_tl
9087 \seq_pop_right:NN
9088 \g_@@_jekyll_data_datatypes_seq
9089 \l_tmpa_tl
9090 \markdown_jekyll_data_update_address_tls:
9091 }

```

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.

```

9092 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
9093 {
9094 \keys_set_known:nn
9095 { markdown/jekyllData }
9096 { { #1 } = { #2 } }
9097 }
9098 \cs_generate_variant:Nn
9099 \markdown_jekyll_data_set_keyval:nn
9100 { Vn }
9101 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
9102 {
9103 \markdown_jekyll_data_push:nN
9104 { #1 }
9105 \c_@@_jekyll_data_scalar_tl
9106 \markdown_jekyll_data_set_keyval:Vn
9107 \g_@@_jekyll_data_wildcard_absolute_address_tl
9108 { #2 }
9109 \markdown_jekyll_data_set_keyval:Vn
9110 \g_@@_jekyll_data_wildcard_relative_address_tl
9111 { #2 }
9112 \markdown_jekyll_data_pop:
9113 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

9114 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
9115 \markdown_jekyll_data_push:nN
9116 { #1 }

```

```

9117 \c_@@_jekyll_data_sequence_tl
9118 }
9119 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
9120 \markdown_jekyll_data_push:nN
9121 { #1 }
9122 \c_@@_jekyll_data_mapping_tl
9123 }
9124 \def\markdownRendererJekyllDataSequenceEndPrototype{
9125 \markdown_jekyll_data_pop:
9126 }
9127 \def\markdownRendererJekyllDataMappingEndPrototype{
9128 \markdown_jekyll_data_pop:
9129 }
9130 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
9131 \markdown_jekyll_data_set_keyvals:nN
9132 { #1 }
9133 { #2 }
9134 }
9135 \def\markdownRendererJekyllDataEmptyPrototype#1{}
9136 \def\markdownRendererJekyllDataNumberPrototype#1#2{
9137 \markdown_jekyll_data_set_keyvals:nN
9138 { #1 }
9139 { #2 }
9140 }
9141 \def\markdownRendererJekyllDataStringPrototype#1#2{
9142 \markdown_jekyll_data_set_keyvals:nN
9143 { #1 }
9144 { #2 }
9145 }
9146 \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).

```

9147 \ExplSyntaxOn
9148 \tl_new:N \g_@@_formatted_lua_options_tl
9149 \cs_new:Nn \@@_format_lua_options:
9150 {
9151 \tl_gclear:N
9152 \g_@@_formatted_lua_options_tl
9153 \seq_map_function:NN
9154 \g_@@_lua_options_seq
9155 \@@_format_lua_option:n
9156 }

```

```

9157 \cs_new:Nn \@@_format_lua_option:n
9158 {
9159 \@@_typecheck_option:n
9160 { #1 }
9161 \@@_get_option_type:nN
9162 { #1 }
9163 \l_tmpa_tl
9164 \bool_case_true:nF
9165 {
9166 {
9167 \str_if_eq_p:VV
9168 \l_tmpa_tl
9169 \c_@@_option_type_boolean_tl ||
9170 \str_if_eq_p:VV
9171 \l_tmpa_tl
9172 \c_@@_option_type_number_tl ||
9173 \str_if_eq_p:VV
9174 \l_tmpa_tl
9175 \c_@@_option_type_counter_tl
9176 }
9177 {
9178 \@@_get_option_value:nN
9179 { #1 }
9180 \l_tmpa_tl
9181 \tl_gput_right:Nx
9182 \g_@@_formatted_lua_options_tl
9183 { #1~::~ \l_tmpa_tl ,~ }
9184 }
9185 }
9186 \str_if_eq_p:VV
9187 \l_tmpa_tl
9188 \c_@@_option_type_clist_tl
9189 }
9190 {
9191 \@@_get_option_value:nN
9192 { #1 }
9193 \l_tmpa_tl
9194 \tl_gput_right:Nx
9195 \g_@@_formatted_lua_options_tl
9196 { #1~::~\c_left_brace_str }
9197 \clist_map_inline:Vn
9198 \l_tmpa_tl
9199 {
9200 \tl_gput_right:Nx
9201 \g_@@_formatted_lua_options_tl
9202 { "##1" ,~ }
9203 }

```

```

9204 \tl_gput_right:Nx
9205 \g_@@_formatted_lua_options_tl
9206 { \c_right_brace_str ,~ }
9207 }
9208 }
9209 {
9210 \@@_get_option_value:nN
9211 { #1 }
9212 \l_tmpa_tl
9213 \tl_gput_right:Nx
9214 \g_@@_formatted_lua_options_tl
9215 { #1~~ " \l_tmpa_tl " ,~ }
9216 }
9217 }
9218 \cs_generate_variant:Nn
9219 \clist_map_inline:nn
9220 { Vn }
9221 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
9222 \def\markdownLuaOptions{ \g_@@_formatted_lua_options_tl }
9223 \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.

```
9224 \def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```

9225 local lfs = require("lfs")
9226 local cacheDir = "\markdownOptionCacheDir"
9227 if not lfs.isdir(cacheDir) then
9228 assert(lfs.mkdir(cacheDir))
9229 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.

```

9230 local md = require("markdown")
9231 local convert = md.new(\markdownLuaOptions)
9232 }%

```

### 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.

```

9233 \ExplSyntaxOn
9234 \prg_new_conditional:Nnn
9235 \@@_if_option:n

```

```

9236 { TF, T, F }
9237 {
9238 \@@_get_option_type:nN
9239 { #1 }
9240 \l_tmpa_tl
9241 \str_if_eq:NNF
9242 \l_tmpa_tl
9243 \c_@@_option_type_boolean_tl
9244 {
9245 \msg_error:nxxx
9246 { markdown }
9247 { expected-boolean-option }
9248 { #1 }
9249 { \l_tmpa_tl }
9250 }
9251 \@@_get_option_value:nN
9252 { #1 }
9253 \l_tmpa_tl
9254 \str_if_eq:NNTF
9255 \l_tmpa_tl
9256 \c_@@_option_value_true_tl
9257 { \prg_return_true: }
9258 { \prg_return_false: }
9259 }
9260 \msg_new:nnn
9261 { markdown }
9262 { expected-boolean-option }
9263 {
9264 Option~#1~has~type~#2,~
9265 but~a~boolean~was~expected.
9266 }
9267 \let\markdownIfOption=\@@_if_option:nTF
9268 \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.

```

9269 \csname newread\endcsname\markdownInputFileStream
9270 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

9271 \begingroup
9272 \catcode\^^I=12%
9273 \gdef\markdownReadAndConvertTab{^^I}%
9274 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX} 2_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

9275 `\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.

```
9276 \catcode\^^M=13%
9277 \catcode\^^I=13%
9278 \catcode|=0%
9279 \catcode\\=12%
9280 |catcode@=14%
9281 |catcode|=12@
9282 |gdef|markdownReadAndConvert#1#2{@
9283 |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
9284 |markdownIfOption{frozenCache}{@
9285 |immediate|openout|markdownOutputFileStream@
9286 |markdownOptionInputTempFileName|relax@
9287 |markdownInfo{Buffering markdown input into the temporary @
9288 |input file "|markdownOptionInputTempFileName" and scanning @
9289 |for the closing token sequence "#1"}@
9290 }@
```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```
9291 |def|do##1{|catcode`##1=12}|dospecials@
9292 |catcode`|=12@
9293 |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.

```
9294 |def|markdownReadAndConvertStripPercentSign##1{@
9295 |markdownIfOption{stripPercentSigns}{@
9296 |if##1%@
9297 |expandafter|expandafter|expandafter@
9298 |markdownReadAndConvertProcessLine@
9299 |else@
9300 |expandafter|expandafter|expandafter@
9301 |markdownReadAndConvertProcessLine@
9302 |expandafter|expandafter|expandafter##1@
9303 |fi@
```

```

9304 }{@
9305 |expandafter@
9306 |markdownReadAndConvertProcessLine@
9307 |expandafter##1@
9308 }@
9309 }@

```

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.

```

9310 |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.

```

9311 |ifx|relax##3|relax@
9312 |markdownIfOption{frozenCache}{}{@
9313 |immediate|write|markdownOutputFileStream{##1}@
9314 }@
9315 |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.

```

9316 |def^^M{@
9317 |markdownInfo{The ending token sequence was found}@
9318 |markdownIfOption{frozenCache}{}{@
9319 |immediate|closeout|markdownOutputFileStream@
9320 }@
9321 |endgroup@
9322 |markdownInput{@
9323 |markdownOptionOutputDir@
9324 /|markdownOptionInputTempFileName@
9325 }@
9326 #2}@
9327 |fi@

```

Repeat with the next line.

```

9328 ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

9329 |catcode`|^I=13@
9330 |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.

```

9331 |catcode`|^M=13@
9332 |def^M##1^M{@
9333 |def^M###1^M{@
9334 |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
9335 ^M}@
9336 ^M}@

```

Reset the character categories back to the former state.

```
9337 |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.

```

9338 \ExplSyntaxOn
9339 \int_compare:nT
9340 { \markdownMode = 3 }
9341 {
9342 \markdownInfo{Using mode~3:~The~lt3luabridge~package}
9343 \file_input:n { lt3luabridge.tex }
9344 \cs_new:Npn
9345 \markdownLuaExecute
9346 { \luabridgeExecute }
9347 }
9348 \ExplSyntaxOff

```

### 3.2.5 Lua Shell Escape Bridge

The following  $\TeX$  code is intended for  $\TeX$  engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the Lua $\TeX$  engine, their  $\TeX$  distribution contains it, and uses shell access to produce and execute Lua scripts using the  $\TeX$  Lua interpreter [1, Section 4.1.1].

```

9349 \ifnum\markdownMode<2\relax
9350 \ifnum\markdownMode=0\relax
9351 \markdownWarning{Using mode 0: Shell escape via write18
9352 (deprecated, to be removed in Markdown 3.0.0)}%
9353 \else
9354 \markdownWarning{Using mode 1: Shell escape via os.execute
9355 (deprecated, to be removed in Markdown 3.0.0)}%
9356 \fi

```



The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (1), disabled (0), or restricted (2).

Inherit the value of the `\pdfshellescape` (LuaTeX, PdfTeX) or the `\shellescape` (XeTeX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```

9357 \ifx\pdfshellescape\undefined
9358 \ifx\shellescape\undefined
9359 \ifnum\markdownMode=0\relax
9360 \def\markdownExecuteShellEscape{1}%
9361 \else
9362 \def\markdownExecuteShellEscape{%
9363 \directlua{tex.sprint(status.shell_escape or "1")}}%
9364 \fi
9365 \else
9366 \let\markdownExecuteShellEscape\shellescape
9367 \fi
9368 \else
9369 \let\markdownExecuteShellEscape\pdfshellescape
9370 \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.

```

9371 \ifnum\markdownMode=0\relax
9372 \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
9373 \else
9374 \def\markdownExecuteDirect#1{%
9375 \directlua{os.execute("\luaescapestring{#1}")}}%
9376 \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.

```

9377 \def\markdownExecute#1{%
9378 \ifnum\markdownExecuteShellEscape=1\relax
9379 \markdownExecuteDirect{#1}%
9380 \else
9381 \markdownError{I can not access the shell}{Either run the TeX
9382 compiler with the --shell-escape or the --enable-write18 flag,
9383 or set shell_escape=t in the texmf.cnf file}%
9384 \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.

```

9385 \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.

```
9386 \catcode`|=0%
9387 \catcode`\|=12%
9388 |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.

```
9389 |immediate|openout|markdownOutputFileStream=%
9390 |markdownOptionHelperScriptFileName
9391 |markdownInfo{Writing a helper Lua script to the file
9392 "|markdownOptionHelperScriptFileName"}%
9393 |immediate|write|markdownOutputFileStream{%
9394 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.

```
9395 local kpse
9396 (function()
9397 local should_initialize = package.loaded.kpse == nil
9398 or tex.initialize
9399 local ran_ok
9400 ran_ok, kpse = pcall(require, "kpse")
9401 if ran_ok and should_initialize then
9402 kpse.set_program_name("luatex")
9403 end
9404 end)()
9405 #1
9406 end)
```

If there was an error, use the file `errorTempFileName` to store the error message.

```
9407 if not ran_ok then
9408 local file = io.open("%
9409 |markdownOptionOutputDir
9410 /|markdownOptionErrorTempFileName", "w")
9411 if file then
9412 file:write(error .. "\n")
9413 file:close()
9414 end
9415 print('\|markdownError{An error was encountered while executing
9416 Lua code}{For further clues, examine the file
9417 "|markdownOptionOutputDir
9418 /|markdownOptionErrorTempFileName}')
9419 end}%
9420 |immediate|closeout|markdownOutputFileStream
```

Execute the generated `helperScriptFileName` Lua script using the `TeXLua` binary and store the output in the `outputTempFileName` file.

```

9421 |markdownInfo{Executing a helper Lua script from the file
9422 "|markdownOptionHelperScriptFileName" and storing the result in the
9423 file "|markdownOptionOutputTempFileName"}%
9424 |markdownExecute{texlua "|markdownOptionOutputDir
9425 /|markdownOptionHelperScriptFileName" > %
9426 "|markdownOptionOutputDir
9427 /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `outputTempFileName` file.

```

9428 |input|markdownOptionOutputTempFileName|relax}%
9429 |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.

```

9430 \fi
9431 \ifnum\markdownMode=2\relax
9432 \markdownWarning{Using mode 2: Direct Lua access
9433 (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,

```

9434 \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.

```

9435 \catcode`\|=0%
9436 \catcode`\|=12%
9437 |gdef|markdownLuaExecute#1{%
9438 |directlua{%
9439 local function print(input)
9440 local output = {}
9441 for line in input:gmatch("[^\r\n]+") do
9442 table.insert(output, line)
9443 end
9444 tex.print(output)
9445 end
9446 #1
9447 }%
9448 }%

```

```
9449 |endgroup
9450 \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.

```
9451 \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.

```
9452 \catcode`|=0%
9453 \catcode`\|=12%
9454 \catcode`|&=6%
9455 |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.

```
9456 |begingroup
9457 |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.

```
9458 |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
9459 |markdownIfOption{frozenCache}{%
9460 |ifnum|markdownOptionFrozenCacheCounter=0|relax
9461 |markdownInfo{Reading frozen cache from
9462 |"|markdownOptionFrozenCacheFileName"|}%
9463 |input|markdownOptionFrozenCacheFileName|relax
9464 |fi
9465 |markdownInfo{Including markdown document number
9466 |"|the|markdownOptionFrozenCacheCounter" from frozen cache}%
9467 |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
9468 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
9469 }{%
9470 |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

9471 |openin|markdownInputFileStream&1
9472 |closein|markdownInputFileStream
9473 |markdownPrepareLuaOptions
9474 |markdownLuaExecute{%
9475 |markdownPrepare
9476 local file = assert(io.open("&1", "r"),
9477 [[Could not open file "&1" for reading]])
9478 local input = assert(file:read("*a"))
9479 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.

```

9480 print(convert(input))}%

```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```

9481 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
9482 }%
9483 |endgroup
9484 }%
9485 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of  $\TeX$  to execute a  $\TeX$  document in the middle of a markdown document fragment.

```

9486 \gdef\markdownEscape#1{%
9487 \catcode`\%=14\relax
9488 \catcode`\#=6\relax
9489 \input #1\relax
9490 \catcode`\%=12\relax
9491 \catcode`\#=12\relax
9492 }%

```

### 3.3 $\LaTeX$ Implementation

The  $\LaTeX$  implementation makes use of the fact that, apart from some subtle differences,  $\LaTeX$  implements the majority of the plain  $\TeX$  format [12, Section 9]. As a consequence, we can directly reuse the existing plain  $\TeX$  implementation.

```

9493 \def\markdownVersionSpace{ }%
9494 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
9495 \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.

```

9496 \ExplSyntaxOn
9497 \@@_latex_define_renderers:
9498 \@@_latex_define_renderer_prototypes:
9499 \ExplSyntaxOff

```

### 3.3.1 Logging Facilities

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

### 3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.2).

```
9500 \let\markdownInputPlainTeX\markdownInput
9501 \renewcommand\markdownInput[2] [] {%
9502 \begingroup
9503 \markdownSetup{#1}%
9504 \markdownInputPlainTeX{#2}%
9505 \endgroup}%
```

The `markdown`, and `markdown*`  $\LaTeX$  environments are implemented using the `\markdownReadAndConvert` macro.

```
9506 \renewenvironment{markdown}{%
9507 \markdownReadAndConvert@markdown{}}{%
9508 \markdownEnd}%
9509 \renewenvironment{markdown*}[1]{%
9510 \markdownSetup{#1}%
9511 \markdownReadAndConvert@markdown*}{%
9512 \markdownEnd}%
9513 \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*.

```
9514 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
9515 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
9516 |gdef|markdownReadAndConvert@markdown#1<%
9517 |markdownReadAndConvert<\end{markdown#1}>%
9518 <|end<markdown#1>>>%
9519 |endgroup
```

**3.3.2.1  $\LaTeX$  Themes** This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
9520 \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.

```

9521 \newcommand\markdownLaTeXThemeName{}
9522 \seq_new:N \g_@@_latex_themes_seq
9523 \seq_gput_right:NV
9524 \g_@@_latex_themes_seq
9525 \markdownLaTeXThemeName
9526 \newcommand\markdownLaTeXThemeLoad[2]{
9527 \def\@tempa{%
9528 \def\markdownLaTeXThemeName{#2}
9529 \seq_gput_right:NV
9530 \g_@@_latex_themes_seq
9531 \markdownLaTeXThemeName
9532 \RequirePackage{#1}
9533 \seq_pop_right:NN
9534 \g_@@_latex_themes_seq
9535 \l_tmpa_tl
9536 \seq_get_right:NN
9537 \g_@@_latex_themes_seq
9538 \l_tmpa_tl
9539 \exp_args:NNV
9540 \def
9541 \markdownLaTeXThemeName
9542 \l_tmpa_tl}
9543 \ifmarkdownLaTeXLoaded
9544 \@tempa
9545 \else
9546 \exp_args:No
9547 \AtEndOfPackage
9548 { \@tempa }
9549 \fi}
9550 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
9551 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
9552 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
9553 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
9554 \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:

```
9555 \renewcommand\markdownRendererInputFencedCodePrototype[2]{%
9556 \def\next##1 ##2\relax{%
9557 \ifthenelse{equal{##1}{dot}}{%

```

```

9558 \markdownIfOption{frozenCache}{-}{%
9559 \immediate\write18{%
9560 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
9561 then
9562 dot -Tpdf -o #1.pdf #1;
9563 cp #1 #1.pdf.source;
9564 fi}}%

```

We include the typeset image using the image token renderer:

```

9565 \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:

```

9566 }{%
9567 \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
9568 }%
9569 }%
9570 \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

9571 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
9572 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```

9573 \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.

```

9574 \newcount\markdown@witiko@graphicx@http@counter
9575 \markdown@witiko@graphicx@http@counter=0
9576 \newcommand\markdown@witiko@graphicx@http@filename{%
9577 \markdownOptionCacheDir/witiko_graphicx_http%
9578 .\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.

```

9579 \newcommand\markdown@witiko@graphicx@http@download[2]{%
9580 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:

```

9581 \begingroup
9582 \catcode`\%=12
9583 \catcode`\^^A=14

```



We redefine the image token renderer prototype, so that it tries to download an online image.

```
9584 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
9585 \begingroup
9586 \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:

```
9587 \markdownIfOption{frozenCache}{}{^^A
9588 \immediate\write18{^^A
9589 mkdir -p "\markdownOptionCacheDir";
9590 if printf '%s' "#3" | grep -q -E '^https?:';
9591 then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
9592 OUTPUT_PREFIX="\markdownOptionCacheDir";
9593 OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
9594 OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/)";
9595 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
9596 if ! [-e "$OUTPUT"];
9597 then
9598 \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
9599 printf '%s' "$OUTPUT" > "\filename";
9600 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:

```
9601 else
9602 printf '%s' '#3' > "\filename";
9603 fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
9604 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
9605 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
9606 {#1}{#2}{\filename}{#4}^^A
9607 \endgroup
9608 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
9609 \endgroup
```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
9610 \renewcommand\markdownRendererTildePrototype{~}%
```

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```
9611 \DeclareOption*{%
9612 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
9613 \ProcessOptions\relax
```

After processing the options, activate the `jekyllDataRenderes`, `renderers`, `rendererPrototypes`, and `code` keys.

```
9614 \ExplSyntaxOn
9615 \keys_define:nn
9616 { markdown/latex-options }
9617 {
9618 renderers .code:n = {
9619 \keys_set:nn
9620 { markdown/latex-options/renderers }
9621 { #1 }
9622 },
9623 }
9624 \@@_with_various_cases:nn
9625 { rendererPrototypes }
9626 {
9627 \keys_define:nn
9628 { markdown/latex-options }
9629 {
9630 #1 .code:n = {
9631 \keys_set:nn
9632 { markdown/latex-options/renderer-prototypes }
9633 { ##1 }
9634 },
9635 }
9636 }
```

The `code` key is used to immediately expand and execute code, which can be especially useful in  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  snippets.

```
9637 \keys_define:nn
9638 { markdown/latex-options }
9639 {
9640 code .code:n = { #1 },
9641 }
```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```
9642 \@@_with_various_cases:nn
9643 { jekyllDataRenderers }
9644 {
9645 \keys_define:nn
```

```

9646 { markdown/latex-options }
9647 {
9648 #1 .code:n = {
9649 \tl_set:Nn
9650 \l_tmpa_tl
9651 { ##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.

```

9652 \tl_replace_all:NnV
9653 \l_tmpa_tl
9654 { / }
9655 \c_backslash_str
9656 \keys_set:nV
9657 { markdown/latex-options/jekyll-data-renderers }
9658 \l_tmpa_tl
9659 },
9660 }
9661 }
9662 \keys_define:nn
9663 { markdown/latex-options/jekyll-data-renderers }
9664 {
9665 unknown .code:n = {
9666 \tl_set_eq:NN
9667 \l_tmpa_tl
9668 \l_keys_key_str
9669 \tl_replace_all:NVn
9670 \l_tmpa_tl
9671 \c_backslash_str
9672 { / }
9673 \tl_put_right:Nn
9674 \l_tmpa_tl
9675 {
9676 .code:n = { #1 }
9677 }
9678 \keys_define:nV
9679 { markdown/jekyllData }
9680 \l_tmpa_tl
9681 }
9682 }
9683 \cs_generate_variant:Nn
9684 \keys_define:nn
9685 { nV }
9686 \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.

```
9687 \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.

```
9688 \@ifclassloaded{beamer}{}{%
9689 \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
9690 \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
9691 }
```

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.

```
9692 \ExplSyntaxOn
9693 \@ifpackageloaded{paralist}{
9694 \tl_new:N
9695 \l_@@_latex_fancy_list_item_label_number_style_tl
9696 \tl_new:N
9697 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9698 \cs_new:Nn
9699 \@@_latex_fancy_list_item_label_number:nn
9700 {
9701 \str_case:nn
9702 { #1 }
9703 {
9704 { Decimal } { #2 }
9705 { LowerRoman } { \int_to_roman:n { #2 } }
9706 { UpperRoman } { \int_to_Roman:n { #2 } }
9707 { LowerAlpha } { \int_to_alph:n { #2 } }
9708 { UpperAlpha } { \int_to_alph:n { #2 } }
9709 }
9710 }
9711 \cs_new:Nn
9712 \@@_latex_fancy_list_item_label_delimiter:n
9713 {
9714 \str_case:nn
9715 { #1 }
9716 {
9717 { Default } { . }
9718 { OneParen } {) }
9719 { Period } { . }
9720 }
9721 }
9722 \cs_new:Nn
```

```

9723 \@@_latex_fancy_list_item_label:nnn
9724 {
9725 \@@_latex_fancy_list_item_label_number:nn
9726 { #1 }
9727 { #3 }
9728 \@@_latex_fancy_list_item_label_delimiter:n
9729 { #2 }
9730 }
9731 \cs_new:Nn
9732 \@@_latex_paralist_style:nn
9733 {
9734 \str_case:nn
9735 { #1 }
9736 {
9737 { Decimal } { 1 }
9738 { LowerRoman } { i }
9739 { UpperRoman } { I }
9740 { LowerAlpha } { a }
9741 { UpperAlpha } { A }
9742 }
9743 \@@_latex_fancy_list_item_label_delimiter:n
9744 { #2 }
9745 }
9746 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

9747 ulBeginTight = {%
9748 \group_begin:
9749 \pltopsep=\topsep
9750 \plpartopsep=\partopsep
9751 \begin{compactitem}
9752 },
9753 ulEndTight = {
9754 \end{compactitem}
9755 \group_end:
9756 },
9757 fancyOlBegin = {
9758 \group_begin:
9759 \tl_set:Nn
9760 \l_@@_latex_fancy_list_item_label_number_style_tl
9761 { #1 }
9762 \tl_set:Nn
9763 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9764 { #2 }
9765 \tl_set:Nn
9766 \l_tmpa_tl

```

```

9767 { \begin{enumerate}[]
9768 \tl_put_right:Nx
9769 \l_tmpa_tl
9770 { \@@_latex_paralist_style:nn { #1 } { #2 } }
9771 \tl_put_right:Nn
9772 \l_tmpa_tl
9773 {] }
9774 \l_tmpa_tl
9775 },
9776 fancyOlEnd = {
9777 \end{enumerate}
9778 \group_end:
9779 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

9780 olBeginTight = {%
9781 \group_begin:
9782 \plpartopsep=\partopsep
9783 \pltopsep=\topsep
9784 \begin{compactenum}
9785 },
9786 olEndTight = {
9787 \end{compactenum}
9788 \group_end:
9789 },
9790 fancyOlBeginTight = {
9791 \group_begin:
9792 \tl_set:Nn
9793 \l_@@_latex_fancy_list_item_label_number_style_tl
9794 { #1 }
9795 \tl_set:Nn
9796 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9797 { #2 }
9798 \tl_set:Nn
9799 \l_tmpa_tl
9800 {
9801 \plpartopsep=\partopsep
9802 \pltopsep=\topsep
9803 \begin{compactenum}[
9804]
9805 \tl_put_right:Nx
9806 \l_tmpa_tl
9807 { \@@_latex_paralist_style:nn { #1 } { #2 } }
9808 \tl_put_right:Nn
9809 \l_tmpa_tl
9810 {] }

```

```

9811 \l_tmpa_tl
9812 },
9813 fancyO1EndTight = {
9814 \end{compactenum}
9815 \group_end:
9816 },
9817 fancyO1ItemWithNumber = {
9818 \item
9819 [
9820 \@@_latex_fancy_list_item_label:VVn
9821 \l_@@_latex_fancy_list_item_label_number_style_tl
9822 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
9823 { #1 }
9824]
9825 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

9826 dlBeginTight = {
9827 \group_begin:
9828 \plpartopsep=\partopsep
9829 \pltopsep=\topsep
9830 \begin{compactdesc}
9831 },
9832 dlEndTight = {
9833 \end{compactdesc}
9834 \group_end:
9835 }}}
9836 \cs_generate_variant:Nn
9837 \@@_latex_fancy_list_item_label:nnn
9838 { VVn }
9839 }{
9840 \markdownSetup{rendererPrototypes={
9841 ulBeginTight = {\markdownRendererUlBegin},
9842 ulEndTight = {\markdownRendererUlEnd},
9843 fancyO1Begin = {\markdownRendererO1Begin},
9844 fancyO1End = {\markdownRendererO1End},
9845 olBeginTight = {\markdownRendererO1Begin},
9846 olEndTight = {\markdownRendererO1End},
9847 fancyO1BeginTight = {\markdownRendererO1Begin},
9848 fancyO1EndTight = {\markdownRendererO1End},
9849 dlBeginTight = {\markdownRendererDlBegin},
9850 dlEndTight = {\markdownRendererDlEnd}}}
9851 }
9852 \ExplSyntaxOff
9853 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

9854 \@ifpackageloaded{unicode-math}{
9855 \markdownSetup{rendererPrototypes={
9856 untickedBox = {\mdlgwhtsquare},
9857 }}
9858 }{
9859 \RequirePackage{amssymb}
9860 \markdownSetup{rendererPrototypes={
9861 untickedBox = {\square},
9862 }}
9863 }
9864 \RequirePackage{csvsimple}
9865 \RequirePackage{fancyvrb}
9866 \RequirePackage{graphicx}
9867 \markdownSetup{rendererPrototypes={
9868 hardLineBreak = {\},
9869 leftBrace = {\textbraceleft},
9870 rightBrace = {\textbraceright},
9871 dollarSign = {\textdollar},
9872 underscore = {\textunderscore},
9873 circumflex = {\textasciicircum},
9874 backslash = {\textbackslash},
9875 tilde = {\textasciitilde},
9876 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>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.

```

9877 codeSpan = {%
9878 \ifmmode
9879 \text{#1}%
9880 \else
9881 \texttt{#1}%
9882 \fi
9883 }}
9884 \ExplSyntaxOn
9885 \markdownSetup{
9886 rendererPrototypes = {
9887 contentBlock = {

```

---

<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.



```

9888 \str_case:nnF
9889 { #1 }
9890 {
9891 { csv }
9892 {
9893 \begin{table}
9894 \begin{center}
9895 \csvautotabular{#3}
9896 \end{center}
9897 \tl_if_empty:nF
9898 { #4 }
9899 { \caption{#4} }
9900 \end{table}
9901 }
9902 { tex } { \markdownEscape{#3} }
9903 }
9904 { \markdownInput{#3} }
9905 },
9906 },
9907 }
9908 \ExplSyntaxOff
9909 \markdownSetup{rendererPrototypes={
9910 image = {%
9911 \begin{figure}%
9912 \begin{center}%
9913 \includegraphics{#3}%
9914 \end{center}%
9915 \ifx\empty#4\empty\else
9916 \caption{#4}%
9917 \fi
9918 \end{figure}},
9919 ulBegin = {\begin{itemize}},
9920 ulEnd = {\end{itemize}},
9921 olBegin = {\begin{enumerate}},
9922 olItem = {\item{}},
9923 olItemWithNumber = {\item[#1.]},
9924 olEnd = {\end{enumerate}},
9925 dlBegin = {\begin{description}},
9926 dlItem = {\item[#1]},
9927 dlEnd = {\end{description}},
9928 emphasis = {\emph{#1}},
9929 tickedBox = {\\boxtimes},
9930 halfTickedBox = {\\boxdot},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

9931 headerAttributeContextBegin = {%

```

```

9932 \markdownSetup{
9933 rendererPrototypes = {
9934 attributeIdentifier = {%
9935 \begingroup
9936 \def\next####1{%
9937 \def#####1#####1{%
9938 \endgroup
9939 #####1{#####1}%
9940 \label{##1}%
9941 }%
9942 }%
9943 \next\markdownRendererHeadingOne
9944 \next\markdownRendererHeadingTwo
9945 \next\markdownRendererHeadingThree
9946 \next\markdownRendererHeadingFour
9947 \next\markdownRendererHeadingFive
9948 \next\markdownRendererHeadingSix
9949 },
9950 },
9951 }%
9952 },
9953 headerAttributeContextEnd = {},
9954 superscript = {#1},
9955 subscript = {\textsubscript{#1}},
9956 displayMath = {\begin{displaymath}#1\end{displaymath}},
9957 inlineMath = {\begin{math}#1\end{math}},
9958 blockQuoteBegin = {\begin{quotation}},
9959 blockQuoteEnd = {\end{quotation}},
9960 inputVerbatim = {\VerbatimInput{#1}},
9961 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
9962 note = {\footnote{#1}}}

```

**3.3.4.1 Fenced Code** When no infostring has been specified, default to the indented code block renderer.

```

9963 \RequirePackage{ltxcmds}
9964 \ExplSyntaxOn
9965 \cs_gset:Npn
9966 \markdownRendererInputFencedCodePrototype#1#2
9967 {
9968 \tl_if_empty:nTF
9969 { #2 }
9970 { \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.

```

9971 {
9972 \regex_extract_once:nnN

```

```

9973 { \w* }
9974 { #2 }
9975 \l_tmpa_seq
9976 \seq_pop_left:NN
9977 \l_tmpa_seq
9978 \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

9979 \ltx@ifpackageloaded
9980 { minted }
9981 {
9982 \catcode`\#=6\relax
9983 \exp_args:NV
9984 \inputminted
9985 \l_tmpa_tl
9986 { #1 }
9987 \catcode`\#=12\relax
9988 }
9989 {

```

When the listings package is loaded, use it for syntax highlighting.

```

9990 \ltx@ifpackageloaded
9991 { listings }
9992 { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostrng were given.

```

9993 { \markdownRendererInputFencedCode{#1}{ } }
9994 }
9995 }
9996 }
9997 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

9998 \ExplSyntaxOn
9999 \def\markdownLATEXStrongEmphasis#1{%
10000 \str_if_in:NnTF
10001 \f@series
10002 { b }
10003 { \textnormal{#1} }
10004 { \textbf{#1} }
10005 }
10006 \ExplSyntaxOff
10007 \markdownSetup{rendererPrototypes={strongEmphasis={%
10008 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

10009 \@ifundefined{chapter}{%
10010 \markdownSetup{rendererPrototypes = {

```

```

10011 headingOne = {\section{#1}},
10012 headingTwo = {\subsection{#1}},
10013 headingThree = {\subsubsection{#1}},
10014 headingFour = {\paragraph{#1}\leavevmode},
10015 headingFive = {\subparagraph{#1}\leavevmode}}}
10016 }{%
10017 \markdownSetup{rendererPrototypes = {
10018 headingOne = {\chapter{#1}},
10019 headingTwo = {\section{#1}},
10020 headingThree = {\subsection{#1}},
10021 headingFour = {\subsubsection{#1}},
10022 headingFive = {\paragraph{#1}\leavevmode},
10023 headingSix = {\subparagraph{#1}\leavevmode}}}
10024 }%

```

**3.3.4.2 Tickboxes** If the `taskLists` option is enabled, we will hide bullets in un-ordered list items with tickboxes.

```

10025 \markdownSetup{
10026 rendererPrototypes = {
10027 ulItem = {%
10028 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
10029 },
10030 },
10031 }
10032 \def\markdownLaTeXUItem{%
10033 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
10034 \item[\markdownLaTeXCheckbox]%
10035 \expandafter\@gobble
10036 \else
10037 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
10038 \item[\markdownLaTeXCheckbox]%
10039 \expandafter\expandafter\expandafter\@gobble
10040 \else
10041 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
10042 \item[\markdownLaTeXCheckbox]%
10043 \expandafter\expandafter\expandafter\expandafter
10044 \expandafter\expandafter\expandafter\@gobble
10045 \else
10046 \item{}%
10047 \fi
10048 \fi
10049 \fi
10050 }

```

**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.

```

10051 \@ifundefined{HCode}{}{
10052 \markdownSetup{
10053 rendererPrototypes = {
10054 inlineHtmlTag = {%
10055 \ifvmode
10056 \IgnorePar
10057 \EndP
10058 \fi
10059 \HCode{#1}%
10060 },
10061 inputBlockHtmlElement = {%
10062 \ifvmode
10063 \IgnorePar
10064 \fi
10065 \EndP
10066 \special{t4ht* <#1}%
10067 \par
10068 \ShowPar
10069 },
10070 },
10071 }
10072 }

```

**3.3.4.4 Citations** Here is a basic implementation for citations that uses the `LATEX` `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `BibLATEX` `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

10073 \newcount\markdownLaTeXCitationsCounter
10074
10075 % Basic implementation
10076 \RequirePackage{gobble}
10077 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
10078 \advance\markdownLaTeXCitationsCounter by 1\relax
10079 \ifx\relax#4\relax
10080 \ifx\relax#5\relax
10081 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10082 \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
10083 \expandafter\expandafter\expandafter
10084 \expandafter\expandafter\expandafter\expandafter
10085 \@gobblethree
10086 \fi
10087 \else% Before a postnote (#5), dump the accumulator

```

---

<sup>29</sup>See <https://tug.org/tex4ht/>.

```

10088 \ifx\relax#1\relax\else
10089 \cite{#1}%
10090 \fi
10091 \cite[#5]{#6}%
10092 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10093 \else
10094 \expandafter\expandafter\expandafter
10095 \expandafter\expandafter\expandafter\expandafter
10096 \expandafter\expandafter\expandafter
10097 \expandafter\expandafter\expandafter\expandafter
10098 \markdownLaTeXBasicCitations
10099 \fi
10100 \expandafter\expandafter\expandafter
10101 \expandafter\expandafter\expandafter\expandafter{%
10102 \expandafter\expandafter\expandafter
10103 \expandafter\expandafter\expandafter\expandafter}%
10104 \expandafter\expandafter\expandafter
10105 \expandafter\expandafter\expandafter\expandafter{%
10106 \expandafter\expandafter\expandafter
10107 \expandafter\expandafter\expandafter\expandafter}%
10108 \expandafter\expandafter\expandafter
10109 \@gobblethree
10110 \fi
10111 \else% Before a prenote (#4), dump the accumulator
10112 \ifx\relax#1\relax\else
10113 \cite{#1}%
10114 \fi
10115 \ifnum\markdownLaTeXCitationsCounter>1\relax
10116 \space % Insert a space before the prenote in later citations
10117 \fi
10118 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
10119 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10120 \else
10121 \expandafter\expandafter\expandafter
10122 \expandafter\expandafter\expandafter\expandafter
10123 \markdownLaTeXBasicCitations
10124 \fi
10125 \expandafter\expandafter\expandafter{%
10126 \expandafter\expandafter\expandafter}%
10127 \expandafter\expandafter\expandafter{%
10128 \expandafter\expandafter\expandafter}%
10129 \expandafter
10130 \@gobblethree
10131 \fi\markdownLaTeXBasicCitations{#1#2#6},}
10132 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
10133
10134 % Natbib implementation

```

```

10135 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
10136 \advance\markdownLaTeXCitationsCounter by 1\relax
10137 \ifx\relax#3\relax
10138 \ifx\relax#4\relax
10139 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10140 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
10141 \expandafter\expandafter\expandafter
10142 \expandafter\expandafter\expandafter\expandafter
10143 \@gobbletwo
10144 \fi
10145 \else% Before a postnote (#4), dump the accumulator
10146 \ifx\relax#1\relax\else
10147 \citep{#1}%
10148 \fi
10149 \citep[] [#4]{#5}%
10150 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10151 \else
10152 \expandafter\expandafter\expandafter
10153 \expandafter\expandafter\expandafter\expandafter
10154 \expandafter\expandafter\expandafter
10155 \expandafter\expandafter\expandafter\expandafter
10156 \markdownLaTeXNatbibCitations
10157 \fi
10158 \expandafter\expandafter\expandafter
10159 \expandafter\expandafter\expandafter\expandafter{%
10160 \expandafter\expandafter\expandafter
10161 \expandafter\expandafter\expandafter\expandafter}%
10162 \expandafter\expandafter\expandafter
10163 \@gobbletwo
10164 \fi
10165 \else% Before a prenote (#3), dump the accumulator
10166 \ifx\relax#1\relax\relax\else
10167 \citep{#1}%
10168 \fi
10169 \citep[#3] [#4]{#5}%
10170 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10171 \else
10172 \expandafter\expandafter\expandafter
10173 \expandafter\expandafter\expandafter\expandafter
10174 \markdownLaTeXNatbibCitations
10175 \fi
10176 \expandafter\expandafter\expandafter{%
10177 \expandafter\expandafter\expandafter}%
10178 \expandafter
10179 \@gobbletwo
10180 \fi\markdownLaTeXNatbibCitations{#1,#5}}
10181 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%

```

```

10182 \advance\markdownLaTeXCitationsCounter by 1\relax
10183 \ifx\relax#3\relax
10184 \ifx\relax#4\relax
10185 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10186 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
10187 \expandafter\expandafter\expandafter
10188 \expandafter\expandafter\expandafter\expandafter
10189 \@gobbletwo
10190 \fi
10191 \else% After a prenote or a postnote, dump the accumulator
10192 \ifx\relax#1\relax\else
10193 \citet{#1}%
10194 \fi
10195 , \citet[#3][#4]{#5}%
10196 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
10197 ,
10198 \else
10199 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
10200 ,
10201 \fi
10202 \fi
10203 \expandafter\expandafter\expandafter
10204 \expandafter\expandafter\expandafter\expandafter
10205 \markdownLaTeXNatbibTextCitations
10206 \expandafter\expandafter\expandafter
10207 \expandafter\expandafter\expandafter\expandafter{%
10208 \expandafter\expandafter\expandafter
10209 \expandafter\expandafter\expandafter\expandafter}%
10210 \expandafter\expandafter\expandafter
10211 \@gobbletwo
10212 \fi
10213 \else% After a prenote or a postnote, dump the accumulator
10214 \ifx\relax#1\relax\relax\else
10215 \citet{#1}%
10216 \fi
10217 , \citet[#3][#4]{#5}%
10218 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
10219 ,
10220 \else
10221 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
10222 ,
10223 \fi
10224 \fi
10225 \expandafter\expandafter\expandafter
10226 \markdownLaTeXNatbibTextCitations
10227 \expandafter\expandafter\expandafter{%
10228 \expandafter\expandafter\expandafter}%

```



```

10229 \expandafter
10230 \@gobbletwo
10231 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
10232
10233 % BibLaTeX implementation
10234 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
10235 \advance\markdownLaTeXCitationsCounter by 1\relax
10236 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10237 \autocites#1[#3][#4]{#5}%
10238 \expandafter\@gobbletwo
10239 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
10240 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
10241 \advance\markdownLaTeXCitationsCounter by 1\relax
10242 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
10243 \textcites#1[#3][#4]{#5}%
10244 \expandafter\@gobbletwo
10245 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
10246
10247 \markdownSetup{rendererPrototypes = {
10248 cite = {%
10249 \markdownLaTeXCitationsCounter=1%
10250 \def\markdownLaTeXCitationsTotal{#1}%
10251 \@ifundefined{autocites}{%
10252 \@ifundefined{citep}{%
10253 \expandafter\expandafter\expandafter
10254 \markdownLaTeXBasicCitations
10255 \expandafter\expandafter\expandafter{%
10256 \expandafter\expandafter\expandafter}%
10257 \expandafter\expandafter\expandafter{%
10258 \expandafter\expandafter\expandafter}%
10259 }{%
10260 \expandafter\expandafter\expandafter
10261 \markdownLaTeXNatbibCitations
10262 \expandafter\expandafter\expandafter{%
10263 \expandafter\expandafter\expandafter}%
10264 }%
10265 }{%
10266 \expandafter\expandafter\expandafter
10267 \markdownLaTeXBibLaTeXCitations
10268 \expandafter{\expandafter}%
10269 }},
10270 textCite = {%
10271 \markdownLaTeXCitationsCounter=1%
10272 \def\markdownLaTeXCitationsTotal{#1}%
10273 \@ifundefined{autocites}{%
10274 \@ifundefined{citep}{%
10275 \expandafter\expandafter\expandafter

```

```

10276 \markdownLaTeXBasicTextCitations
10277 \expandafter\expandafter\expandafter{%
10278 \expandafter\expandafter\expandafter}%
10279 \expandafter\expandafter\expandafter{%
10280 \expandafter\expandafter\expandafter}%
10281 }{%
10282 \expandafter\expandafter\expandafter
10283 \markdownLaTeXNatbibTextCitations
10284 \expandafter\expandafter\expandafter{%
10285 \expandafter\expandafter\expandafter}%
10286 }%
10287 }{%
10288 \expandafter\expandafter\expandafter
10289 \markdownLaTeXBibLaTeXTextCitations
10290 \expandafter{\expandafter}%
10291 }}}}
```

### 3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```

10292 \RequirePackage{url}
10293 \RequirePackage{expl3}
10294 \ExplSyntaxOn
10295 \def\markdownRendererLinkPrototype#1#2#3#4{
10296 \tl_set:Nn \l_tmpa_tl { #1 }
10297 \tl_set:Nn \l_tmpb_tl { #2 }
10298 \bool_set:Nn
10299 \l_tmpa_bool
10300 {
10301 \tl_if_eq_p:NN
10302 \l_tmpa_tl
10303 \l_tmpb_tl
10304 }
10305 \tl_set:Nn \l_tmpa_tl { #4 }
10306 \bool_set:Nn
10307 \l_tmpb_bool
10308 {
10309 \tl_if_empty_p:N
10310 \l_tmpa_tl
10311 }

```

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.

```

10312 \bool_if:nTF
10313 {
10314 \l_tmpa_bool && \l_tmpb_bool
10315 }
10316 {
```

```

10317 \markdownLaTeXRendererAutolink { #2 } { #3 }
10318 }{
10319 \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
10320 }
10321 }
10322 \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.
10323 \tl_set:Nn
10324 \l_tmpa_tl
10325 { #2 }
10326 \tl_trim_spaces:N
10327 \l_tmpa_tl
10328 \tl_set:Nx
10329 \l_tmpb_tl
10330 {
10331 \tl_range:Nnn
10332 \l_tmpa_tl
10333 { 1 }
10334 { 1 }
10335 }
10336 \str_if_eq:NNTF
10337 \l_tmpb_tl
10338 \c_hash_str
10339 {
10340 \tl_set:Nx
10341 \l_tmpb_tl
10342 {
10343 \tl_range:Nnn
10344 \l_tmpa_tl
10345 { 2 }
10346 { -1 }
10347 }
10348 \exp_args:NV
10349 \ref
10350 \l_tmpb_tl
10351 }{
10352 \url { #2 }
10353 }
10354 }
10355 \ExplSyntaxOff
10356 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
10357 #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.

```

10358 \newcount\markdownLaTeXRowCount
10359 \newcount\markdownLaTeXRowTotal
10360 \newcount\markdownLaTeXColumnCounter
10361 \newcount\markdownLaTeXColumnTotal
10362 \newtoks\markdownLaTeXTable
10363 \newtoks\markdownLaTeXTableAlignment
10364 \newtoks\markdownLaTeXTableEnd
10365 \AtBeginDocument{%
10366 \@ifpackageloaded{booktabs}{%
10367 \def\markdownLaTeXTopRule{\toprule}%
10368 \def\markdownLaTeXMidRule{\midrule}%
10369 \def\markdownLaTeXBottomRule{\bottomrule}%
10370 }{%
10371 \def\markdownLaTeXTopRule{\hline}%
10372 \def\markdownLaTeXMidRule{\hline}%
10373 \def\markdownLaTeXBottomRule{\hline}%
10374 }%
10375 }
10376 \markdownSetup{rendererPrototypes={
10377 table = {%
10378 \markdownLaTeXTable={}%
10379 \markdownLaTeXTableAlignment={}%
10380 \markdownLaTeXTableEnd={%
10381 \markdownLaTeXBottomRule
10382 \end{tabular}}%
10383 \ifx\empty#1\empty\else
10384 \addto@hook\markdownLaTeXTable{%
10385 \begin{table}
10386 \centering%
10387 \addto@hook\markdownLaTeXTableEnd{%
10388 \caption{#1}
10389 \end{table}}%
10390 \fi
10391 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
10392 \markdownLaTeXRowCount=0%
10393 \markdownLaTeXRowTotal=#2%
10394 \markdownLaTeXColumnTotal=#3%
10395 \markdownLaTeXRenderTableRow
10396 }
10397 }}
10398 \def\markdownLaTeXRenderTableRow#1{%
10399 \markdownLaTeXColumnCounter=0%
10400 \ifnum\markdownLaTeXRowCount=0\relax
10401 \markdownLaTeXReadAlignments#1%
10402 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
10403 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
10404 \the\markdownLaTeXTableAlignment}}%

```

```

10405 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
10406 \else
10407 \markdownLaTeXRenderTableCell#1%
10408 \fi
10409 \ifnum\markdownLaTeXRowCounter=1\relax
10410 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
10411 \fi
10412 \advance\markdownLaTeXRowCounter by 1\relax
10413 \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
10414 \the\markdownLaTeXTable
10415 \the\markdownLaTeXTableEnd
10416 \expandafter\@gobble
10417 \fi\markdownLaTeXRenderTableRow}
10418 \def\markdownLaTeXReadAlignments#1{%
10419 \advance\markdownLaTeXColumnCounter by 1\relax
10420 \if#1d%
10421 \addto@hook\markdownLaTeXTableAlignment{1}%
10422 \else
10423 \addto@hook\markdownLaTeXTableAlignment{#1}%
10424 \fi
10425 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
10426 \expandafter\@gobble
10427 \fi\markdownLaTeXReadAlignments}
10428 \def\markdownLaTeXRenderTableCell#1{%
10429 \advance\markdownLaTeXColumnCounter by 1\relax
10430 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
10431 \addto@hook\markdownLaTeXTable{#1&}%
10432 \else
10433 \addto@hook\markdownLaTeXTable{#1\\}%
10434 \expandafter\@gobble
10435 \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.

```

10436
10437 \markdownIfOption{lineBlocks}{%
10438 \RequirePackage{verse}
10439 \markdownSetup{rendererPrototypes={
10440 lineBlockBegin = {%
10441 \begingroup
10442 \def\markdownRendererHardLineBreak{\}%
10443 \begin{verse}%
10444 },
10445 lineBlockEnd = {%
10446 \end{verse}%
10447 \endgroup

```

```

10448 },
10449 }}
10450 }{}
10451

```

**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.

```

10452 \ExplSyntaxOn
10453 \keys_define:nn
10454 { markdown/jekyllData }
10455 {
10456 author .code:n = { \author{#1} },
10457 date .code:n = { \date{#1} },
10458 title .code:n = { \title{#1} },
10459 }

```

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.

```

10460 % TODO: Remove the command definition in TeX Live 2021.
10461 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
10462 \markdownSetup{
10463 rendererPrototypes = {
10464 jekyllDataEnd = {
10465 % TODO: Remove the else branch in TeX Live 2021.
10466 \IfFormatAtLeastTF
10467 { 2020-10-01 }
10468 { \AddToHook{begindocument/end}{\maketitle} }
10469 {
10470 \ifx\@onlypreamble\@notprerr
10471 % We are in the document
10472 \maketitle
10473 \else
10474 % We are in the preamble
10475 \RequirePackage{etoolbox}
10476 \AfterEndPreamble{\maketitle}
10477 \fi
10478 }
10479 },
10480 },
10481 }
10482 \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.

```

10483 \markdownIfOption{strikeThrough}{%
10484 \RequirePackage{soulutf8}%
10485 \markdownSetup{
10486 rendererPrototypes = {
10487 strikeThrough = {%
10488 \st{#1}%
10489 },
10490 }
10491 }
10492 }{}

```

**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.

```

10493 \markdownIfOption{strikeThrough}{%
10494 \RequirePackage{soulutf8}%
10495 \markdownSetup{
10496 rendererPrototypes = {
10497 strikeThrough = {%
10498 \st{#1}%
10499 },
10500 }
10501 }
10502 }{}

```

**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.

```

10503 \ExplSyntaxOn
10504 \@@_if_option:nT
10505 { linkAttributes }
10506 {
10507 \RequirePackage{graphicx}
10508 \markdownSetup{
10509 rendererPrototypes = {
10510 imageAttributeContextBegin = {
10511 \group_begin:
10512 \markdownSetup{
10513 rendererPrototypes = {
10514 attributeKeyValue = {
10515 \setkeys
10516 { Gin }
10517 { { ##1 } = { ##2 } }

```

```

10518 },
10519 },
10520 }
10521 },
10522 imageAttributeContextEnd = {
10523 \group_end:
10524 },
10525 },
10526 }
10527 }
10528 \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`.

```

10529 \ExplSyntaxOn
10530 \cs_gset:Npn
10531 \markdownRendererInputRawInlinePrototype#1#2
10532 {
10533 \str_case:nnF
10534 { #2 }
10535 {
10536 { latex }
10537 {
10538 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10539 { #1 }
10540 { tex }
10541 }
10542 }
10543 {
10544 \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10545 { #1 }
10546 { #2 }
10547 }
10548 }
10549 \cs_gset:Npn
10550 \markdownRendererInputRawBlockPrototype#1#2
10551 {
10552 \str_case:nnF
10553 { #2 }
10554 {
10555 { latex }
10556 {
10557 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
10558 { #1 }
10559 { tex }
10560 }

```



```

10561 }
10562 {
10563 \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
10564 { #1 }
10565 { #2 }
10566 }
10567 }
10568 \ExplSyntaxOff
10569 \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.

```

10570 \newcommand\markdownMakeOther{%
10571 \count0=128\relax
10572 \loop
10573 \catcode\count0=11\relax
10574 \advance\count0 by 1\relax
10575 \ifnum\count0<256\repeat}%

```

## 3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the documentation is scarce) the majority of the plain TeX format required by the plain TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` L<sup>A</sup>T<sub>E</sub>X package.

```

10576 \def\markdownMakeOther{%
10577 \count0=128\relax
10578 \loop
10579 \catcode\count0=11\relax
10580 \advance\count0 by 1\relax
10581 \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTeXt.

```

10582 \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```
10583 \long\def\inputmarkdown{%
10584 \dosingleempty
10585 \doinputmarkdown}%
10586 \long\def\doinputmarkdown[#1]#2{%
10587 \begingroup
10588 \iffirstargument
10589 \setupmarkdown{#1}%
10590 \fi
10591 \markdownInput{#2}%
10592 \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 ConTeXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTeXt MkIV and therefore to insert hard line breaks into markdown text.

```
10593 \ifx\startluacode\undefined % MkII
10594 \begingroup
10595 \catcode`\|=0%
10596 \catcode`\|=12%
10597 |gdef|startmarkdown{%
10598 |markdownReadAndConvert{\stopmarkdown}%
10599 {|stopmarkdown}}%
10600 |gdef|stopmarkdown{%
10601 |markdownEnd}%
10602 |endgroup
10603 \else % MkIV
10604 \startluacode
10605 document.markdown_buffering = false
10606 local function preserve_trailing_spaces(line)
10607 if document.markdown_buffering then
10608 line = line:gsub("[\t][\t]$", "\t\t")
10609 end
10610 return line
10611 end
10612 resolvers.installinputlinehandler(preserve_trailing_spaces)
10613 \stopluacode
10614 \begingroup
10615 \catcode`\|=0%
10616 \catcode`\|=12%
```

```

10617 |gdef|startmarkdown{%
10618 |ctxlua{document.markdown_buffering = true}%
10619 |markdownReadAndConvert{\stopmarkdown}%
10620 |stopmarkdown}}%
10621 |gdef|stopmarkdown{%
10622 |ctxlua{document.markdown_buffering = false}%
10623 |markdownEnd}%
10624 |endgroup
10625 \fi

```

### 3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

10626 \def\markdownRendererHardLineBreakPrototype{\blank}%
10627 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
10628 \def\markdownRendererRightBracePrototype{\textbraceright}%
10629 \def\markdownRendererDollarSignPrototype{\textdollar}%
10630 \def\markdownRendererPercentSignPrototype{\percent}%
10631 \def\markdownRendererUnderscorePrototype{\textunderscore}%
10632 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
10633 \def\markdownRendererBackslashPrototype{\textbackslash}%
10634 \def\markdownRendererTildePrototype{\textasciitilde}%
10635 \def\markdownRendererPipePrototype{\char`|}%
10636 \def\markdownRendererLinkPrototype#1#2#3#4{%
10637 \useURL[#1][#3][][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
10638 \fi\texttt<\hyphenatedurl{#3}>}}%
10639 \usemodule[database]
10640 \defineseparatedlist
10641 [MarkdownConTeXtCSV]
10642 [separator={,},
10643 before=\bTABLE,after=\eTABLE,
10644 first=\bTR,last=\eTR,
10645 left=\bTD,right=\eTD]
10646 \def\markdownConTeXtCSV{csv}
10647 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
10648 \def\markdownConTeXtCSV@arg{#1}%
10649 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
10650 \placetable[][tab:#1]{#4}{%
10651 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
10652 \else
10653 \markdownInput{#3}%
10654 \fi}%
10655 \def\markdownRendererImagePrototype#1#2#3#4{%
10656 \placefigure[][#4]{\externalfigure[#3]}%
10657 \def\markdownRendererUlBeginPrototype{\startitemize}%
10658 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
10659 \def\markdownRendererUlItemPrototype{\item}%

```

```

10660 \def\markdownRendererUListEndPrototype{\stopitemize}%
10661 \def\markdownRendererUListEndTightPrototype{\stopitemize}%
10662 \def\markdownRendererOListBeginPrototype{\startitemize[n]}%
10663 \def\markdownRendererOListBeginTightPrototype{\startitemize[packed,n]}%
10664 \def\markdownRendererOListItemPrototype{\item}%
10665 \def\markdownRendererOListItemWithNumberPrototype#1{\sym{#1.}}%
10666 \def\markdownRendererOListEndPrototype{\stopitemize}%
10667 \def\markdownRendererOListEndTightPrototype{\stopitemize}%
10668 \definedescription
10669 [MarkdownConTeXtDListItemPrototype]
10670 [location=hanging,
10671 margin=standard,
10672 headstyle=bold]%
10673 \definestartstop
10674 [MarkdownConTeXtDListPrototype]
10675 [before=\blank,
10676 after=\blank]%
10677 \definestartstop
10678 [MarkdownConTeXtDListTightPrototype]
10679 [before=\blank\startpacked,
10680 after=\stoppacked\blank]%
10681 \def\markdownRendererDListBeginPrototype{%
10682 \startMarkdownConTeXtDListPrototype}%
10683 \def\markdownRendererDListBeginTightPrototype{%
10684 \startMarkdownConTeXtDListTightPrototype}%
10685 \def\markdownRendererDListItemPrototype#1{%
10686 \startMarkdownConTeXtDListItemPrototype{#1}}%
10687 \def\markdownRendererDListItemEndPrototype{%
10688 \stopMarkdownConTeXtDListItemPrototype}%
10689 \def\markdownRendererDListEndPrototype{%
10690 \stopMarkdownConTeXtDListPrototype}%
10691 \def\markdownRendererDListEndTightPrototype{%
10692 \stopMarkdownConTeXtDListTightPrototype}%
10693 \def\markdownRendererEmphasisPrototype#1{\em#1}%
10694 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
10695 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
10696 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
10697 \def\markdownRendererLineBlockBeginPrototype{%
10698 \begingroup
10699 \def\markdownRendererHardLineBreak{
10700 }%
10701 \startlines
10702 }%
10703 \def\markdownRendererLineBlockEndPrototype{%
10704 \stoptlines
10705 \endgroup
10706 }%

```

```
10707 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
```

**3.4.2.1 Fenced Code** When no infostring has been specified, default to the indented code block renderer.

```
10708 \ExplSyntaxOn
10709 \cs_gset:Npn
10710 \markdownRendererInputFencedCodePrototype#1#2
10711 {
10712 \tl_if_empty:nTF
10713 { #2 }
10714 { \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 ConT<sub>E</sub>Xt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```
\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
 \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
 \stopmarkdown
\stoptext
```

```
10715 {
10716 \regex_extract_once:nnN
10717 { \w* }
10718 { #2 }
10719 \l_tmpa_seq
10720 \seq_pop_left:NN
10721 \l_tmpa_seq
10722 \l_tmpa_tl
10723 \typefile[\l_tmpa_tl] []{#1}
10724 }
10725 }
10726 \ExplSyntaxOff
10727 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
```

```

10728 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
10729 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
10730 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
10731 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
10732 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
10733 \def\markdownRendererThematicBreakPrototype{%
10734 \blackrule[height=1pt, width=\hsize]}%
10735 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
10736 \def\markdownRendererTickedBoxPrototype{\boxtimes$}
10737 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
10738 \def\markdownRendererUntickedBoxPrototype{\square$}
10739 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
10740 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
10741 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
10742 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%
10743 \def\markdownRendererInlineMathPrototype#1{\$#1$}%

```

### 3.4.2.2 Tables

There is a basic implementation of tables.

```

10744 \newcount\markdownConTeXtRowCounter
10745 \newcount\markdownConTeXtRowTotal
10746 \newcount\markdownConTeXtColumnCounter
10747 \newcount\markdownConTeXtColumnTotal
10748 \newtoks\markdownConTeXtTable
10749 \newtoks\markdownConTeXtTableFloat
10750 \def\markdownRendererTablePrototype#1#2#3{%
10751 \markdownConTeXtTable={}%
10752 \ifx\empty#1\empty
10753 \markdownConTeXtTableFloat={%
10754 \the\markdownConTeXtTable}%
10755 \else
10756 \markdownConTeXtTableFloat={%
10757 \placetable{#1}{\the\markdownConTeXtTable}}%
10758 \fi
10759 \beginngroup
10760 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
10761 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
10762 \setupTABLE[r][1][topframe=on, bottomframe=on]
10763 \setupTABLE[r][#1][bottomframe=on]
10764 \markdownConTeXtRowCounter=0%
10765 \markdownConTeXtRowTotal=#2%
10766 \markdownConTeXtColumnTotal=#3%
10767 \markdownConTeXtRenderTableRow}
10768 \def\markdownConTeXtRenderTableRow#1{%
10769 \markdownConTeXtColumnCounter=0%
10770 \ifnum\markdownConTeXtRowCounter=0\relax
10771 \markdownConTeXtReadAlignments#1%

```

```

10772 \markdownConTeXtTable={\bTABLE}%
10773 \else
10774 \markdownConTeXtTable=\expandafter{%
10775 \the\markdownConTeXtTable\bTR}%
10776 \markdownConTeXtRenderTableCell#1%
10777 \markdownConTeXtTable=\expandafter{%
10778 \the\markdownConTeXtTable\eTR}%
10779 \fi
10780 \advance\markdownConTeXtRowCounter by 1\relax
10781 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
10782 \markdownConTeXtTable=\expandafter{%
10783 \the\markdownConTeXtTable\eTABLE}%
10784 \the\markdownConTeXtTableFloat
10785 \endgroup
10786 \expandafter\gobbleoneargument
10787 \fi\markdownConTeXtRenderTableRow}
10788 \def\markdownConTeXtReadAlignments#1{%
10789 \advance\markdownConTeXtColumnCounter by 1\relax
10790 \if#1d%
10791 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
10792 \fi\if#1l%
10793 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
10794 \fi\if#1c%
10795 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
10796 \fi\if#1r%
10797 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
10798 \fi
10799 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
10800 \expandafter\gobbleoneargument
10801 \fi\markdownConTeXtReadAlignments}
10802 \def\markdownConTeXtRenderTableCell#1{%
10803 \advance\markdownConTeXtColumnCounter by 1\relax
10804 \markdownConTeXtTable=\expandafter{%
10805 \the\markdownConTeXtTable\bTD#1\eTD}%
10806 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
10807 \expandafter\gobbleoneargument
10808 \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`.

```

10809 \ExplSyntaxOn
10810 \cs_gset:Npn
10811 \markdownRendererInputRawInlinePrototype#1#2
10812 {
10813 \str_case:nnF

```

```

10814 { #2 }
10815 {
10816 { latex }
10817 {
10818 \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10819 { #1 }
10820 { context }
10821 }
10822 }
10823 {
10824 \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
10825 { #1 }
10826 { #2 }
10827 }
10828 }
10829 \cs_gset:Npn
10830 \markdownRendererInputRawBlockPrototype#1#2
10831 {
10832 \str_case:nnF
10833 { #2 }
10834 {
10835 { context }
10836 {
10837 \@_plain_tex_default_input_raw_block_renderer_prototype:nn
10838 { #1 }
10839 { tex }
10840 }
10841 }
10842 {
10843 \@_plain_tex_default_input_raw_block_renderer_prototype:nn
10844 { #1 }
10845 { #2 }
10846 }
10847 }
10848 \cs_gset_eq:NN
10849 \markdownRendererInputRawBlockPrototype
10850 \markdownRendererInputRawInlinePrototype
10851 \ExplSyntaxOff
10852 \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 T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [8] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [9] Vít Novotný. *L<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

|                                    |    |
|------------------------------------|----|
| <code>blankBeforeBlockquote</code> | 18 |
| <code>blankBeforeCodeFence</code>  | 18 |
| <code>blankBeforeDivFence</code>   | 18 |

|                                   |                                   |
|-----------------------------------|-----------------------------------|
| blankBeforeHeading                | 19                                |
| bracketedSpans                    | 19, 56                            |
| breakableBlockquotes              | 19                                |
| cacheDir                          | 4, 16, 23, 50, 115, 221, 276, 289 |
| citationNbsps                     | 20                                |
| citations                         | 20, 82, 89                        |
| codeSpans                         | 21                                |
| contentBlocks                     | 17, 21                            |
| contentBlocksLanguageMap          | 17                                |
| debugExtensions                   | 9, 17, 22, 217                    |
| debugExtensionsFileName           | 17, 22                            |
| defaultOptions                    | 10, 46, 261                       |
| definitionLists                   | 22, 62                            |
| eagerCache                        | 23, 23                            |
| \endmarkdown                      | 99                                |
| entities.char_entity              | 175                               |
| entities.dec_entity               | 175                               |
| entities.hex_entity               | 175                               |
| errorTempFileName                 | 51, 282                           |
| expandtabs                        | 202                               |
| expectJekyllData                  | 23                                |
| extensions                        | 24, 127, 222                      |
| extensions.bracketed_spans        | 222                               |
| extensions.citations              | 223                               |
| extensions.content_blocks         | 226                               |
| extensions.definition_lists       | 229                               |
| extensions.fancy_lists            | 231                               |
| extensions.fenced_code            | 235                               |
| extensions.fenced_divs            | 239                               |
| extensions.header_attributes      | 241                               |
| extensions.inline_code_attributes | 243                               |
| extensions.jekyll_data            | 258                               |
| extensions.line_blocks            | 243                               |
| extensions.link_attributes        | 244                               |
| extensions.notes                  | 247                               |
| extensions.pipe_table             | 249                               |
| extensions.raw_inline             | 252                               |
| extensions.strike_through         | 252                               |
| extensions.subscripts             | 253                               |
| extensions.superscripts           | 254                               |

|                                                           |                                                 |
|-----------------------------------------------------------|-------------------------------------------------|
| <code>extensions.tex_math</code>                          | 254                                             |
| <code>fancyLists</code>                                   | 26, 77–81, 292                                  |
| <code>fencedCode</code>                                   | 26, 35, 59, 65, 82, 287                         |
| <code>fencedCodeAttributes</code>                         | 27                                              |
| <code>fencedDiv</code>                                    | 66                                              |
| <code>fencedDivs</code>                                   | 27, 36                                          |
| <code>finalizeCache</code>                                | 17, 23, 28, 28, 50, 115, 221                    |
| <code>frozenCache</code>                                  | 17, 28, 50, 106, 115, 287, 289                  |
| <code>frozenCacheCounter</code>                           | 28, 221, 284, 285                               |
| <code>frozenCacheFileName</code>                          | 17, 28, 50, 221                                 |
| <code>hardLineBreaks</code>                               | 29                                              |
| <code>hashEnumerators</code>                              | 29                                              |
| <code>headerAttributes</code>                             | 29, 36, 54, 55, 67                              |
| <code>helperScriptFileName</code>                         | 50–52, 282, 283                                 |
| <code>html</code>                                         | 30, 69, 70, 301                                 |
| <code>hybrid</code>                                       | 30, 35, 40, 43, 52, 84, 106, 115, 179, 203, 284 |
| <code>inlineCodeAttributes</code>                         | 31, 60                                          |
| <code>inlineNotes</code>                                  | 31                                              |
| <code>\inputmarkdown</code>                               | 123, 124, 314                                   |
| <code>inputTempFileName</code>                            | 278, 279                                        |
| <code>iterlines</code>                                    | 201                                             |
| <code>jekyllData</code>                                   | 4, 23, 24, 32, 91–94                            |
| <code>languages_json</code>                               | 226, 226                                        |
| <code>lineBlocks</code>                                   | 33, 72                                          |
| <code>linkAttributes</code>                               | 32, 71, 74, 311                                 |
| <code>\markdown</code>                                    | 99                                              |
| <code>markdown</code>                                     | 99, 99, 286                                     |
| <code>markdown*</code>                                    | 99, 99, 100, 286                                |
| <code>\markdown_jekyll_data_concatenate_address:NN</code> | 272                                             |
| <code>\markdown_jekyll_data_pop:</code>                   | 272                                             |
| <code>\markdown_jekyll_data_push:nN</code>                | 272                                             |
| <code>\markdown_jekyll_data_push_address_segment:n</code> | 271                                             |
| <code>\markdown_jekyll_data_set_keyval:Nn</code>          | 273                                             |
| <code>\markdown_jekyll_data_set_keyvals:nn</code>         | 273                                             |
| <code>\markdown_jekyll_data_update_address_tls:</code>    | 272                                             |
| <code>\markdownBegin</code>                               | 48, 48, 49, 97, 99, 123                         |
| <code>\markdownEnd</code>                                 | 48, 48, 49, 97, 99, 123                         |

|                                                                  |                                |
|------------------------------------------------------------------|--------------------------------|
| <code>\markdownError</code>                                      | 96, 96                         |
| <code>\markdownEscape</code>                                     | 48, 49, 285                    |
| <code>\markdownExecute</code>                                    | 281                            |
| <code>\markdownExecuteDirect</code>                              | 281, 281                       |
| <code>\markdownExecuteShellEscape</code>                         | 281, 281                       |
| <code>\markdownIfOption</code>                                   | 276                            |
| <code>\markdownIfSnippetExists</code>                            | 101                            |
| <code>\markdownInfo</code>                                       | 96                             |
| <code>\markdownInput</code>                                      | 48, 49, 99, 100, 124, 284, 286 |
| <code>\markdownInputFileStream</code>                            | 277                            |
| <code>\markdownInputPlainTeX</code>                              | 286                            |
| <code>\markdownLuaExecute</code>                                 | 280, 281, 283, 284             |
| <code>\markdownLuaOptions</code>                                 | 274, 276                       |
| <code>\markdownLuaRegisterIBCallback</code>                      | 98                             |
| <code>\markdownLuaUnregisterIBCallback</code>                    | 98                             |
| <code>\markdownMakeOther</code>                                  | 96, 313                        |
| <code>\markdownMode</code>                                       | 5, 51, 97, 97, 280, 283        |
| <code>\markdownOptionErrorTempFileName</code>                    | 51                             |
| <code>\markdownOptionFinalizeCache</code>                        | 50                             |
| <code>\markdownOptionFrozenCache</code>                          | 50                             |
| <code>\markdownOptionHelperScriptFileName</code>                 | 50                             |
| <code>\markdownOptionHybrid</code>                               | 52                             |
| <code>\markdownOptionInputTempFileName</code>                    | 51                             |
| <code>\markdownOptionOutputDir</code>                            | 51                             |
| <code>\markdownOptionOutputTempFileName</code>                   | 51                             |
| <code>\markdownOptionStripPercentSigns</code>                    | 54                             |
| <code>\markdownOutputFileStream</code>                           | 277                            |
| <code>\markdownPrepare</code>                                    | 276                            |
| <code>\markdownPrepareLuaOptions</code>                          | 274                            |
| <code>\markdownReadAndConvert</code>                             | 97, 277, 286, 314              |
| <code>\markdownReadAndConvertProcessLine</code>                  | 279, 280                       |
| <code>\markdownReadAndConvertStripPercentSigns</code>            | 278                            |
| <code>\markdownReadAndConvertTab</code>                          | 277                            |
| <code>\markdownRendererAttributeClassName</code>                 | 54                             |
| <code>\markdownRendererAttributeIdentifier</code>                | 54                             |
| <code>\markdownRendererAttributeKeyValue</code>                  | 55                             |
| <code>\markdownRendererBlockHtmlCommentBegin</code>              | 69                             |
| <code>\markdownRendererBlockHtmlCommentEnd</code>                | 69                             |
| <code>\markdownRendererBlockQuoteBegin</code>                    | 55                             |
| <code>\markdownRendererBlockQuoteEnd</code>                      | 56                             |
| <code>\markdownRendererBracketedSpanAttributeContextBegin</code> | 56                             |
| <code>\markdownRendererBracketedSpanAttributeContextEnd</code>   | 56                             |

|                                                               |         |
|---------------------------------------------------------------|---------|
| <code>\markdownRendererCite</code>                            | 82, 89  |
| <code>\markdownRendererCodeSpan</code>                        | 59      |
| <code>\markdownRendererCodeSpanAttributeContextBegin</code>   | 60      |
| <code>\markdownRendererCodeSpanAttributeContextEnd</code>     | 60      |
| <code>\markdownRendererContentBlock</code>                    | 60, 61  |
| <code>\markdownRendererContentBlockCode</code>                | 61      |
| <code>\markdownRendererContentBlockOnlineImage</code>         | 61      |
| <code>\markdownRendererDisplayMath</code>                     | 88      |
| <code>\markdownRendererDlBegin</code>                         | 62      |
| <code>\markdownRendererDlBeginTight</code>                    | 62      |
| <code>\markdownRendererDlDefinitionBegin</code>               | 63      |
| <code>\markdownRendererDlDefinitionEnd</code>                 | 63      |
| <code>\markdownRendererDlEnd</code>                           | 64      |
| <code>\markdownRendererDlEndTight</code>                      | 64      |
| <code>\markdownRendererDlItem</code>                          | 62      |
| <code>\markdownRendererDlItemEnd</code>                       | 63      |
| <code>\markdownRendererDocumentBegin</code>                   | 75      |
| <code>\markdownRendererDocumentEnd</code>                     | 75      |
| <code>\markdownRendererEllipsis</code>                        | 37, 64  |
| <code>\markdownRendererEmphasis</code>                        | 65, 117 |
| <code>\markdownRendererFancyOlBegin</code>                    | 77, 78  |
| <code>\markdownRendererFancyOlBeginTight</code>               | 78      |
| <code>\markdownRendererFancyOlEnd</code>                      | 81      |
| <code>\markdownRendererFancyOlEndTight</code>                 | 81      |
| <code>\markdownRendererFancyOlItem</code>                     | 79      |
| <code>\markdownRendererFancyOlItemEnd</code>                  | 80      |
| <code>\markdownRendererFancyOlItemWithNumber</code>           | 80      |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code> | 65      |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code>   | 65      |
| <code>\markdownRendererFencedDivAttributeContextBegin</code>  | 66      |
| <code>\markdownRendererFencedDivAttributeContextEnd</code>    | 66      |
| <code>\markdownRendererFootnote</code>                        | 76, 95  |
| <code>\markdownRendererFootnotePrototype</code>               | 76, 95  |
| <code>\markdownRendererHalfTickedBox</code>                   | 90      |
| <code>\markdownRendererHardLineBreak</code>                   | 73      |
| <code>\markdownRendererHeaderAttributeContextBegin</code>     | 67      |
| <code>\markdownRendererHeaderAttributeContextEnd</code>       | 67      |
| <code>\markdownRendererHeadingFive</code>                     | 68      |
| <code>\markdownRendererHeadingFour</code>                     | 68      |
| <code>\markdownRendererHeadingOne</code>                      | 67      |
| <code>\markdownRendererHeadingSix</code>                      | 69      |
| <code>\markdownRendererHeadingThree</code>                    | 68      |

|                                                          |         |
|----------------------------------------------------------|---------|
| <code>\markdownRendererHeadingTwo</code>                 | 68      |
| <code>\markdownRendererHorizontalRule</code>             | 89, 95  |
| <code>\markdownRendererHorizontalRulePrototype</code>    | 89, 95  |
| <code>\markdownRendererImage</code>                      | 71      |
| <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>            | 59      |
| <code>\markdownRendererInputRawBlock</code>              | 82      |
| <code>\markdownRendererInputRawInline</code>             | 82      |
| <code>\markdownRendererInputVerbatim</code>              | 59      |
| <code>\markdownRendererInterblockSeparator</code>        | 72      |
| <code>\markdownRendererJekyllDataBegin</code>            | 91      |
| <code>\markdownRendererJekyllDataBoolean</code>          | 93      |
| <code>\markdownRendererJekyllDataEmpty</code>            | 94      |
| <code>\markdownRendererJekyllDataEnd</code>              | 91      |
| <code>\markdownRendererJekyllDataMappingBegin</code>     | 92      |
| <code>\markdownRendererJekyllDataMappingEnd</code>       | 92      |
| <code>\markdownRendererJekyllDataNumber</code>           | 93      |
| <code>\markdownRendererJekyllDataSequenceBegin</code>    | 92      |
| <code>\markdownRendererJekyllDataSequenceEnd</code>      | 93      |
| <code>\markdownRendererJekyllDataString</code>           | 94      |
| <code>\markdownRendererLineBlockBegin</code>             | 72      |
| <code>\markdownRendererLineBlockEnd</code>               | 72      |
| <code>\markdownRendererLineBreak</code>                  | 73      |
| <code>\markdownRendererLineBreakPrototype</code>         | 73      |
| <code>\markdownRendererLink</code>                       | 74, 117 |
| <code>\markdownRendererLinkAttributeContextBegin</code>  | 74      |
| <code>\markdownRendererLinkAttributeContextEnd</code>    | 74      |
| <code>\markdownRendererNbsp</code>                       | 75      |
| <code>\markdownRendererNote</code>                       | 76      |
| <code>\markdownRendererOlBegin</code>                    | 77      |
| <code>\markdownRendererOlBeginTight</code>               | 77      |
| <code>\markdownRendererOlEnd</code>                      | 80      |
| <code>\markdownRendererOlEndTight</code>                 | 81      |
| <code>\markdownRendererOlItem</code>                     | 37, 78  |
| <code>\markdownRendererOlItemEnd</code>                  | 79      |
| <code>\markdownRendererOlItemWithNumber</code>           | 37, 79  |
| <code>\markdownRendererReplacementCharacter</code>       | 83      |

|                                                |                          |
|------------------------------------------------|--------------------------|
| <code>\markdownRendererSectionBegin</code>     | 83                       |
| <code>\markdownRendererSectionEnd</code>       | 83                       |
| <code>\markdownRendererStrikeThrough</code>    | 86                       |
| <code>\markdownRendererStrongEmphasis</code>   | 65                       |
| <code>\markdownRendererSubscript</code>        | 87                       |
| <code>\markdownRendererSuperscript</code>      | 87                       |
| <code>\markdownRendererTable</code>            | 87                       |
| <code>\markdownRendererTextCite</code>         | 89                       |
| <code>\markdownRendererThematicBreak</code>    | 89                       |
| <code>\markdownRendererTickedBox</code>        | 90                       |
| <code>\markdownRendererUlBegin</code>          | 57                       |
| <code>\markdownRendererUlBeginTight</code>     | 57                       |
| <code>\markdownRendererUlEnd</code>            | 58                       |
| <code>\markdownRendererUlEndTight</code>       | 58                       |
| <code>\markdownRendererUlItem</code>           | 57                       |
| <code>\markdownRendererUlItemEnd</code>        | 58                       |
| <code>\markdownRendererUntickedBox</code>      | 90                       |
| <code>\markdownSetup</code>                    | 100, 100, 285, 290       |
| <code>\markdownSetupSnippet</code>             | 100, 100                 |
| <code>\markdownWarning</code>                  | 96                       |
| <br>                                           |                          |
| <code>new</code>                               | 8, 261                   |
| <code>notes</code>                             | 33, 76                   |
| <br>                                           |                          |
| <code>outputTempFileName</code>                | 51, 283                  |
| <br>                                           |                          |
| <code>parsers</code>                           | 189, 201                 |
| <code>parsers.commented_line</code>            | 191                      |
| <code>pipeTables</code>                        | 7, 34, 39, 87            |
| <code>preserveTabs</code>                      | 34, 38, 202              |
| <br>                                           |                          |
| <code>rawAttribute</code>                      | 35, 35, 82               |
| <code>reader</code>                            | 8, 127, 189, 200, 222    |
| <code>reader-&gt;add_special_character</code>  | 8, 10, 217               |
| <code>reader-&gt;auto_link_email</code>        | 209                      |
| <code>reader-&gt;auto_link_url</code>          | 209                      |
| <code>reader-&gt;create_parser</code>          | 202                      |
| <code>reader-&gt;finalize_grammar</code>       | 213                      |
| <code>reader-&gt;initialize_named_group</code> | 217                      |
| <code>reader-&gt;insert_pattern</code>         | 8, 9, 213, 214, 219, 245 |
| <code>reader-&gt;lookup_reference</code>       | 205                      |
| <code>reader-&gt;normalize_tag</code>          | 201                      |
| <code>reader-&gt;options</code>                | 201                      |

|                                                |                                     |
|------------------------------------------------|-------------------------------------|
| reader->parser_functions                       | 202                                 |
| reader->parser_functions.name                  | 202                                 |
| reader->parser_functions.parse_inlines_no_link | 245                                 |
| reader->parsers                                | 201, 201                            |
| reader->register_link                          | 204                                 |
| reader->update_rule                            | 213, 216, 219, 245                  |
| reader->writer                                 | 201                                 |
| reader.new                                     | 200, 201                            |
| relativeReferences                             | 35                                  |
|                                                |                                     |
| \setupmarkdown                                 | 124, 124                            |
| shiftHeadings                                  | 7, 36                               |
| slice                                          | 7, 36, 176, 185, 186                |
| smartEllipses                                  | 37, 64, 115                         |
| \startmarkdown                                 | 123, 123, 314                       |
| startNumber                                    | 37, 78–80                           |
| \stopmarkdown                                  | 123, 123, 314                       |
| strikeThrough                                  | 37, 86, 311                         |
| stripIndent                                    | 38, 202                             |
| stripPercentSigns                              | 278                                 |
| subscripts                                     | 38, 87                              |
| superscripts                                   | 39, 87                              |
| syntax                                         | 214, 218, 219                       |
|                                                |                                     |
| tableCaptions                                  | 7, 39                               |
| taskLists                                      | 40, 90, 300                         |
| texComments                                    | 40, 203                             |
| texMathDollars                                 | 41, 88                              |
| texMathDoubleBackslash                         | 41, 88                              |
| texMathSingleBackslash                         | 41, 88                              |
| tightLists                                     | 42, 57, 58, 62, 64, 77, 78, 81, 292 |
|                                                |                                     |
| underscores                                    | 42                                  |
| util.cache                                     | 127, 128                            |
| util.cache_verbatim                            | 128                                 |
| util.encode_json_string                        | 128                                 |
| util.err                                       | 127                                 |
| util.escaper                                   | 131                                 |
| util.expand_tabs_in_line                       | 128                                 |
| util.flatten                                   | 129                                 |
| util.intersperse                               | 130                                 |
| util.lookup_files                              | 128                                 |
| util.map                                       | 130                                 |



|                                  |                              |
|----------------------------------|------------------------------|
| util.pathname                    | 131                          |
| util.rope_last                   | 130                          |
| util.rope_to_string              | 130                          |
| util.table_copy                  | 128                          |
| util.walk                        | 129, 130                     |
| walkable_syntax                  | 9, 17, 22, 213, 214, 216–219 |
| writer                           | 127, 127, 176, 222           |
| writer->active_attributes        | 185, 185, 186                |
| writer->attributes               | 184                          |
| writer->block_html_comment       | 182                          |
| writer->block_html_element       | 182                          |
| writer->blockquote               | 183                          |
| writer->bulletitem               | 181                          |
| writer->bulletlist               | 180                          |
| writer->citations                | 223                          |
| writer->code                     | 179                          |
| writer->contentblock             | 227                          |
| writer->defer_call               | 189, 189                     |
| writer->definitionlist           | 229                          |
| writer->display_math             | 254                          |
| writer->div_begin                | 239                          |
| writer->div_end                  | 239                          |
| writer->document                 | 183                          |
| writer->ellipsis                 | 178                          |
| writer->emphasis                 | 182                          |
| writer->escape                   | 179                          |
| writer->escape_minimal           | 179                          |
| writer->escape_programmatic_text | 179                          |
| writer->escape_typographic_text  | 179                          |
| writer->escaped_chars            | 178, 179                     |
| writer->escaped_minimal_strings  | 178, 179                     |
| writer->escaped_strings          | 178                          |
| writer->escaped_uri_chars        | 178, 179                     |
| writer->fancyitem                | 232                          |
| writer->fancylist                | 231                          |
| writer->fencedCode               | 235                          |
| writer->get_state                | 188                          |
| writer->hard_line_break          | 177                          |
| writer->heading                  | 187                          |
| writer->identifier               | 179                          |
| writer->image                    | 180                          |

|                             |               |
|-----------------------------|---------------|
| writer->inline_html_comment | 182           |
| writer->inline_html_tag     | 182           |
| writer->inline_math         | 255           |
| writer->interblocksep       | 177           |
| writer->is_writing          | 176, 176      |
| writer->jekyllData          | 258           |
| writer->lineblock           | 243           |
| writer->link                | 180           |
| writer->math                | 179           |
| writer->nbsp                | 177           |
| writer->note                | 247           |
| writer->options             | 176           |
| writer->ordereditem         | 182           |
| writer->orderedlist         | 181           |
| writer->pack                | 177, 221      |
| writer->paragraph           | 177           |
| writer->plain               | 177           |
| writer->pop_attributes      | 185, 185, 186 |
| writer->push_attributes     | 185, 185, 186 |
| writer->rawBlock            | 236           |
| writer->rawInline           | 252           |
| writer->set_state           | 189           |
| writer->slice_begin         | 176           |
| writer->slice_end           | 176           |
| writer->space               | 177           |
| writer->span                | 222           |
| writer->strike_through      | 253           |
| writer->string              | 179           |
| writer->strong              | 183           |
| writer->subscript           | 253           |
| writer->suffix              | 177           |
| writer->superscript         | 254           |
| writer->table               | 250           |
| writer->thematic_break      | 178           |
| writer->checkbox            | 183           |
| writer->uri                 | 179           |
| writer->verbatim            | 183           |
| writer.new                  | 176, 176      |