

A Markdown Interpreter for T_EX

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

Version 3.13.0-0-gdd212d58
2026-01-02

Contents

1	Introduction	1	3	Implementation	170
1.1	Requirements	2	3.1	Lua Implementation	170
1.2	Feedback	6	3.2	Plain T _E X Implementation	410
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation	454
2	Interfaces	7	3.4	ConT _E Xt Implementation	494
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	55	References		506
2.3	L ^A T _E X Interface	156	Index		507
2.4	ConT _E Xt Interface	165			

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	51
3	A sequence diagram of typesetting a document using the Lua CLI	52
4	An example directed graph	79
5	An example mindmap	80
6	An example UML sequence diagram	81
7	The banner of the Markdown package	82
8	A pushdown automaton that recognizes T _E X comments	288

1 Introduction

The Markdown package¹ converts CommonMark² markup to T_EX commands. The functionality is provided both as a Lua module and as plain T_EX, L^AT_EX, and ConT_EXt macro packages that can be used to directly typeset T_EX documents containing markdown markup. Unlike other converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

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

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

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

```

1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown "
4             .. "to plain TeX",
5   author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný, "
6             .. "Andrej Genčur",
7   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
8             "2016-2026 Vít Starý Novotný, Andrej Genčur"},
9   license    = "LPPL 1.3c"
10 }
11
12 if not modules then modules = { } end
13 modules['markdown'] = metadata

```

1.1 Requirements

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

1.1.1 Lua Requirements

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

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

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

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

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

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

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

Only load the package outside the ConT_EXt format, where we can use the resolvers API [1, Section 11.5].

```
16 if resolvers == nil then
17   (function()
```

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

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

All the abovelisted modules are statically linked into the current version of the LuaT_EX engine [2, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

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

```
26 hard lua-tinyyaml
```

1.1.2 Plain T_EX Requirements

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

expl3 A package that enables the expl3 language [3] from the L^AT_EX3 kernel in T_EX Live ≤ 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

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

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

Note that this support for TeX engines other than LuaTeX comes with some limitations with respect to file and directory names. Specifically, the filenames of your .tex files may not contain spaces⁴. If `-output-directory` is provided, it may not contain spaces either.

32 `hard lt3luabridge`

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

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

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [2, Section 4.2.4].

The Lua File System module is statically linked into the LuaTeX engine [2, Section 4.3].

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

1.1.3 L^ATeX Requirements

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

33 `\NeedsTeXFormat{LaTeX2e}`

34 `\RequirePackage{expl3}`

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

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

⁴See <https://github.com/Witiko/markdown/issues/573>.

35 `soft url`

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

36 `soft graphics`

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

The package `paralist` will be used unless the option `experimental` has been enabled, in which case, the package `enumitem` will be used. Furthermore, enabling any test phase [4] will also cause `enumitem` to be used. In a future major version, `enumitem` will replace `paralist` altogether.

37 `soft enumitem`

38 `soft paralist`

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

39 `soft fancyvrb`

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

40 `soft csvsimple`

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

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

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

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

44 `soft amsmath`

45 `soft amssymb`

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

46 `soft graphics`

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

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

soul and xcolor Packages that are used in the default renderer prototypes for strike-throughs and marked text in pdfTeX.

```
49 soft soul
50 soft xcolor
```

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

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

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

```
53 soft ltxcmds
```

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

```
54 soft luaxml
```

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

```
55 soft verse
```

1.1.4 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 2.2.6).

1.2 Feedback

Please use the Markdown project page on GitHub⁵ 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.⁶ community question answering web site under the `markdown` tag.

⁵See <https://github.com/witiko/markdown/issues>.

⁶See <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 [5] is gratefully acknowledged.

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

The T_EX implementation of the package draws inspiration from several sources including the source code of L^AT_EX 2_ε, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from T_EX, the filecontents package by Scott Pakin and others.

2 Interfaces

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

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

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

2.1 Lua Interface

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

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

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

2.1.1 Conversion from Markdown to Plain T_EX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain T_EX according to the table `options`

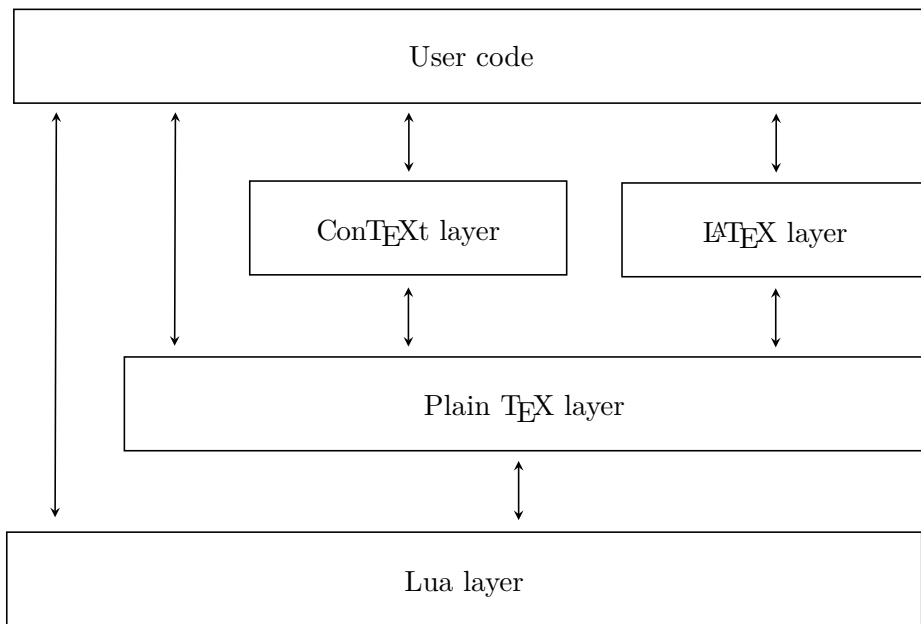


Figure 1: A block diagram of the Markdown package

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

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

```

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

```

2.1.2 User-Defined Syntax Extensions

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

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

```

57 local walkable_syntax = {

```



```

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

```

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

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

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

2.1.3 Options

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

```
89 local defaultOptions = {}
90 local experimentalOptions = {}
91 setmetatable(experimentalOptions, { __index = function (_, key)
92   return defaultOptions[key] end })
```

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

```
93 \ExplSyntaxOn
94 \seq_new:N \g_@@_lua_options_seq
```

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

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

```

125     { #4 }
126     \@@_typecheck_option:n
127     { #2 }
128 }
129 \cs_generate_variant:Nn
130 \@@_add_option:nnnn
131 { Vnnn }
132 \cs_new:Nn
133 \@@_add_experimental_lua_option:n
134 {
135     \@@_add_experimental_option:Vn
136     \c_@@_option_layer_lua_tl
137     { #1 }
138 }
139 \cs_new:Nn
140 \@@_add_experimental_option:nn
141 {
142     \seq_gput_right:cn
143     { g_@@_ #1 _options_seq }
144     { #2 }
145     \seq_gput_right:cn
146     { g_@@_experimental_ #1 _options_seq }
147     { #2 }
148     \prop_gput:cnn
149     { g_@@_ #1 _option_types_prop }
150     { #2 }
151     { boolean }
152 }
153 \cs_generate_variant:Nn
154 \@@_add_experimental_option:nn
155 { Vn }
156 \tl_const:Nn \c_@@_option_value_true_tl { true }
157 \tl_const:Nn \c_@@_option_value_false_tl { false }
158 \cs_new:Nn \@@_typecheck_option:n
159 {
160     \@@_get_option_type:nN
161     { #1 }
162     \l_tmpa_tl
163     \str_case_e:Vn
164     \l_tmpa_tl
165     {
166         { \c_@@_option_type_boolean_tl }
167         {
168             \@@_get_option_value:nN
169             { #1 }
170             \l_tmpa_tl
171             \bool_if:nF

```

```

172         {
173             \str_if_eq_p:eV
174             { \l_tmpa_tl }
175             \c_@@_option_value_true_tl ||
176             \str_if_eq_p:eV
177             { \l_tmpa_tl }
178             \c_@@_option_value_false_tl
179         }
180     {
181         \msg_error:nnne
182         { markdown }
183         { failed-typecheck-for-boolean-option }
184         { #1 }
185         { \l_tmpa_tl }
186     }
187 }
188 }
189 }
190 \msg_new:nnn
191 { markdown }
192 { failed-typecheck-for-boolean-option }
193 {
194     Option~#1~has~value~#2,~
195     but~a~boolean~(true~or~false)~was~expected.
196 }
197 \cs_generate_variant:Nn
198 \str_case_e:nn
199 { Vn }
200 \cs_generate_variant:Nn
201 \msg_error:nnnn
202 { nnne }
203 \prg_generate_conditional_variant:Nnn
204 \str_if_eq:nn
205 { eV }
206 { p }
207 \seq_new:N
208 \g_@@_option_types_seq
209 \tl_const:Nn
210 \c_@@_option_type_clist_tl
211 { clist }
212 \seq_gput_right:NV
213 \g_@@_option_types_seq
214 \c_@@_option_type_clist_tl
215 \tl_const:Nn
216 \c_@@_option_type_counter_tl
217 { counter }
218 \seq_gput_right:NV

```

```

219 \g_@@_option_types_seq
220 \c_@@_option_type_counter_tl
221 \tl_const:Nn
222 \c_@@_option_type_boolean_tl
223 { boolean }
224 \seq_gput_right:NV
225 \g_@@_option_types_seq
226 \c_@@_option_type_boolean_tl
227 \tl_const:Nn
228 \c_@@_option_type_number_tl
229 { number }
230 \seq_gput_right:NV
231 \g_@@_option_types_seq
232 \c_@@_option_type_number_tl
233 \tl_const:Nn
234 \c_@@_option_type_path_tl
235 { path }
236 \seq_gput_right:NV
237 \g_@@_option_types_seq
238 \c_@@_option_type_path_tl
239 \tl_const:Nn
240 \c_@@_option_type_slice_tl
241 { slice }
242 \seq_gput_right:NV
243 \g_@@_option_types_seq
244 \c_@@_option_type_slice_tl
245 \tl_const:Nn
246 \c_@@_option_type_string_tl
247 { string }
248 \seq_gput_right:NV
249 \g_@@_option_types_seq
250 \c_@@_option_type_string_tl
251 \cs_new:Nn
252 \@@_get_option_type:nN
253 {
254   \bool_set_false:N
255   \l_tmpa_bool
256   \seq_map_inline:Nn
257   \g_@@_option_layers_seq
258   {
259     \prop_get:cnNT
260     { g_@@_ ##1 _option_types_prop }
261     { #1 }
262     \l_tmpa_tl
263     {
264       \bool_set_true:N
265       \l_tmpa_bool

```

```

266         \seq_map_break:
267     }
268 }
269 \bool_if:NF
270   \l_tmpa_bool
271   {
272     \msg_error:nnn
273       { markdown }
274       { undefined-option }
275       { #1 }
276   }
277 \seq_if_in:NVF
278   \g_@@_option_types_seq
279   \l_tmpa_tl
280   {
281     \msg_error:nnnV
282       { markdown }
283       { unknown-option-type }
284       { #1 }
285     \l_tmpa_tl
286   }
287 \tl_set_eq:NN
288   #2
289   \l_tmpa_tl
290 }
291 \msg_new:nnn
292   { markdown }
293   { unknown-option-type }
294   {
295     Option~#1~has~unknown~type~#2.
296   }
297 \msg_new:nnn
298   { markdown }
299   { undefined-option }
300   {
301     Option~#1~is~undefined.
302   }
303 \cs_generate_variant:Nn
304   \msg_error:nnnn
305   { nnnV }
306 \cs_new:Nn
307   \@@_get_default_option_value:nN
308   {
309     \bool_set_false:N
310       \l_tmpa_bool
311     \seq_map_inline:Nn
312       \g_@@_option_layers_seq

```

```

313     {
314         \seq_if_in:cnT
315         { g__markdown_experimental_ ##1 _options_seq }
316         { #1 }
317         {
318             \@@_get_option_value:nN
319             { experimental }
320             #2
321             \bool_set_true:N
322             \l_tmpa_bool
323             \seq_map_break:
324         }
325         \prop_get:cnNT
326         { g_@@_default_ ##1 _options_prop }
327         { #1 }
328         #2
329         {
330             \bool_set_true:N
331             \l_tmpa_bool
332             \seq_map_break:
333         }
334     }
335     \bool_if:NF
336     \l_tmpa_bool
337     {
338         \msg_error:nnn
339         { markdown }
340         { undefined-option }
341         { #1 }
342     }
343 }
344 \cs_new:Nn
345 \@@_get_option_value:nN
346 {
347     \@@_option_tl_to_csname:nN
348     { #1 }
349     \l_tmpa_tl
350     \cs_if_free:cTF
351     { \l_tmpa_tl }
352     {
353         \@@_get_default_option_value:nN
354         { #1 }
355         #2
356     }
357     {
358         \@@_get_option_type:nN
359         { #1 }

```

```

360         \l_tmpa_tl
361     \str_if_eq:NNTF
362         \c_@@_option_type_counter_tl
363         \l_tmpa_tl
364     {
365         \@@_option_tl_to_csname:nN
366         { #1 }
367         \l_tmpa_tl
368         \tl_set:Nx
369             #2
370         { \the \cs:w \l_tmpa_tl \cs_end: }
371     }
372     {
373         \@@_option_tl_to_csname:nN
374         { #1 }
375         \l_tmpa_tl
376         \tl_set:Nv
377             #2
378         { \l_tmpa_tl }
379     }
380 }
381 }
382 \cs_new:Nn \@@_option_tl_to_csname:nN
383 {
384     \tl_set:Nn
385         \l_tmpa_tl
386         { \str_uppercase:n { #1 } }
387     \tl_set:Nx
388         #2
389         {
390             markdownOption
391             \tl_head:f { \l_tmpa_tl }
392             \tl_tail:n { #1 }
393         }
394 }

```

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

```

395 \cs_new:Nn \@@_with_various_cases:nn
396 {
397     \seq_clear:N
398     \l_tmpa_seq
399     \seq_map_inline:Nn
400         \g_@@_cases_seq
401         {
402             \tl_set:Nn

```



```

403         \l_tmpa_tl
404         { #1 }
405         \use:c { ##1 }
406         \l_tmpa_tl
407         \seq_put_right:NV
408         \l_tmpa_seq
409         \l_tmpa_tl
410     }
411     \seq_map_inline:Nn
412     \l_tmpa_seq
413     { #2 }
414 }

```

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

```

415 \seq_new:N \g_@@_cases_seq
416 \cs_new:Nn \@@_camel_case:N
417 {
418     \regex_replace_all:nnN
419     { _ ([a-z]) }
420     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
421     #1
422     \tl_set:Nx
423     #1
424     { #1 }
425 }
426 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
427 \cs_new:Nn \@@_snake_case:N
428 {
429     \regex_replace_all:nnN
430     { ([a-z])([A-Z]) }
431     { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
432     #1
433     \tl_set:Nx
434     #1
435     { #1 }
436 }
437 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 General Behavior

`eagerCache=true, false`

default: `true`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. Furthermore, it can also significantly improve the processing speed for documents that

require multiple compilation runs, since each markdown document is only converted once. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

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

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

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

```
438 \@@_add_lua_option:nnn
439   { eagerCache }
440   { boolean }
441   { true }
442 defaultOptions.eagerCache = true
```

`experimental=true, false`

default: `false`

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

At the moment, this has the following effects:

1. The version `experimental` of the theme `witiko/markdown/defaults` will be loaded.
2. Warnings for hard-deprecated features will become errors.
3. The experimental option `htmlOverLinks` will be enabled.

In the future, the effects may extend to other areas as well.

false Experimental features will be disabled.

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

`singletonCache=true, false`

default: `true`

true Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time- and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions. This has been the default behavior since version 3.0.0 of the Markdown package.

false Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also #226 (comment)⁷. This was the default behavior until version 3.0.0 of the Markdown package.

```
448 \@@_add_lua_option:nnn
449   { singletonCache }
450   { boolean }
451   { true }

452 defaultOptions.singletonCache = true

453 local singletonCache = {
454   convert = nil,
455   options = nil,
456 }
```

`unicodeNormalization=true, false`

default: `true`

true Markdown documents will be normalized using one of the four Unicode normalization forms⁸ before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

false Markdown documents will not be Unicode-normalized before conversion.

```
457 \@@_add_lua_option:nnn
458   { unicodeNormalization }
459   { boolean }
460   { true }

461 defaultOptions.unicodeNormalization = true
```

⁷See <https://github.com/witiko/markdown/pull/226#issuecomment-1599641634>.

⁸See <https://unicode.org/faq/normalization.html>.

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

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

```
462 \@@_add_lua_option:nnn
463   { unicodeNormalizationForm }
464   { string }
465   { nfc }
466 defaultOptions.unicodeNormalizationForm = "nfc"
```

2.1.5 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.

```
467 \str_new:N
468   \g_@@_unquoted_jobname_str
469 \str_gset:NV
470   \g_@@_unquoted_jobname_str
471   \c_sys_jobname_str
472 \bool_new:N
473   \g_@@_jobname_quoted_bool
```

```

474 \regex_replace_all:nnNtF
475 { \A ("|') ( .* ) ("|') \Z }
476 { \2 }
477 \g_@@_unquoted_jobname_str
478 {
479     \bool_gset_true:N
480     \g_@@_jobname_quoted_bool
481 }
482 {
483     \bool_gset_false:N
484     \g_@@_jobname_quoted_bool
485 }
486 \@@_add_lua_option:nnn
487 { cacheDir }
488 { path }
489 {
490     \markdownOptionOutputDir
491     / _markdown_
492     \str_use:N
493     \g_@@_unquoted_jobname_str
494 }
495 defaultOptions.cacheDir = "."

```

`contentBlocksLanguageMap`=*<filename>*

default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.5.9 for more information.

```

496 \@@_add_lua_option:nnn
497 { contentBlocksLanguageMap }
498 { path }
499 { markdown-languages.json }
500 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"

```

`debugExtensionsFileName`=*<filename>*

default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```

501 \@@_add_lua_option:nnn
502   { debugExtensionsFileName }
503   { path }
504   {
505     \markdownOptionOutputDir
506     /
507     \str_use:N
508     \g_@@_unquoted_jobname_str
509     .debug-extensions.json
510   }
511 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

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

```

512 \@@_add_lua_option:nnn
513   { frozenCacheFileName }
514   { path }
515   { \markdownOptionCacheDir / frozenCache.tex }
516 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

2.1.6 Parser Options

autoIdentifiers=true, false default: false

true Enable the Pandoc auto identifiers syntax extension⁹:

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

123 Sesame Street

false Disable the Pandoc auto identifiers syntax extension.

See also the option **gfmAutoIdentifiers**.

```

517 \@@_add_lua_option:nnn

```

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

```

518 { autoIdentifiers }
519 { boolean }
520 { false }

521 defaultOptions.autoIdentifiers = false

```

`blankBeforeBlockquote=true, false` default: false

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

```

522 \@@_add_lua_option:nnn
523 { blankBeforeBlockquote }
524 { boolean }
525 { false }

526 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.

```

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

531 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.

```

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

536 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.

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

`blankBeforeHtmlBlock=true, false` default: false

- `true` Require a blank line between a paragraph and the following Common-Mark HTML block¹⁰.
- `false` Do not require a blank line between a paragraph and the following CommonMark HTML block.

```
542 \@@_add_lua_option:nnn
543 { blankBeforeHtmlBlock }
544 { boolean }
545 { false }
546 defaultOptions.blankBeforeHtmlBlock = false
```

`blankBeforeList=true, false` default: false

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

```
547 \@@_add_lua_option:nnn
548 { blankBeforeList }
549 { boolean }
550 { false }
551 defaultOptions.blankBeforeList = false
```

¹⁰See <https://spec.commonmark.org/0.31.2/#html-blocks>.

`bracketedSpans=true, false` default: false

true Enable the Pandoc bracketed span syntax extension¹¹:

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

false Disable the Pandoc bracketed span syntax extension.

```
552 \@@_add_lua_option:nnn
553   { bracketedSpans }
554   { boolean }
555   { false }
556 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: true

true A blank line separates block quotes.

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

```
557 \@@_add_lua_option:nnn
558   { breakableBlockquotes }
559   { boolean }
560   { true }
561 defaultOptions.breakableBlockquotes = true
```

`citationNbsps=true, false` default: false

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

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

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

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

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension¹²:

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.

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

`codeSpans=true, false`

default: true

`true` Enable the code span syntax:

Use the `printf()` function.
``There is a literal backtick (``) here.``

`false` Disable the code span syntax. This allows you to easily use the quotation mark ligatures in texts that do not contain code spans:

``This is a quote.``

```
572 \@@_add_lua_option:nnn
573   { codeSpans }
574   { boolean }
575   { true }
576 defaultOptions.codeSpans = true
```

¹²See <https://pandoc.org/MANUAL.html#extension-citations>.

`contentBlocks=true, false`

default: false

true

: Enable the iA Writer content blocks syntax extension [6]:

```
``` 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.

```
577 \@@_add_lua_option:nnn
578   { contentBlocks }
579   { boolean }
580   { false }

581 defaultOptions.contentBlocks = false
```

`contentLevel=block, inline`

default: block

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

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

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

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

```
582 \@@_add_lua_option:nnn
583   { contentLevel }
584   { string }
585   { block }

586 defaultOptions.contentLevel = "block"
```

`debugExtensions=true, false`

default: false

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

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

```
587 \@@_add_lua_option:nnn
588   { debugExtensions }
589   { boolean }
590   { false }

591 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.

```
592 \@@_add_lua_option:nnn
593   { definitionLists }
594   { boolean }
595   { false }

596 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: false

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

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

`expectJekyllData=true, false`

default: false

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

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

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

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

```
602 \@@_add_lua_option:nnn
603   { expectJekyllData }
604   { boolean }
605   { false }

606 defaultOptions.expectJekyllData = false
```

`extensions=<filenames>`

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

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

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

```

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

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

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

return strike_through

```

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

```

607 metadata.user_extension_api_version = 2
608 metadata.grammar_version = 4

```

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

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

```

609 \cs_generate_variant:Nn
610 \@@_add_lua_option:nnn
611 { nnV }
612 \@@_add_lua_option:nnV
613 { extensions }
614 { clist }
615 \c_empty_clist
616 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: `false`

`true`      Enable the Pandoc fancy list syntax extension<sup>13</sup>:

<sup>13</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

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

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

```
617 \@@_add_lua_option:nnn
618 { fancyLists }
619 { boolean }
620 { false }

621 defaultOptions.fancyLists = false
```

**fencedCode=true, false**

default: true

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

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

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

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

```
622 \@@_add_lua_option:nnn
623 { fencedCode }
624 { boolean }
625 { true }

626 defaultOptions.fencedCode = true
```



`fencedCodeAttributes=true, false`

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>14</sup>:

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

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

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

631 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

**true** Enable the Pandoc fenced div syntax extension<sup>15</sup>:

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

And another.
:::::
```

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

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

636 defaultOptions.fencedDivs = false
```

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

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

`finalizeCache=true, false`

default: false

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

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

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

641 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>`

default: 0

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

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

```
642 \@@_add_lua_option:nnn
643   { frozenCacheCounter }
644   { counter }
645   { 0 }

646 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: false

`true`      Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>16</sup>:

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

`# 123 Sesame Street`

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

---

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

See also the option `autoIdentifiers`.

```
647 \@@_add_lua_option:nnn
648   { gfmAutoIdentifiers }
649   { boolean }
650   { false }

651 defaultOptions.gfmAutoIdentifiers = false
```

`hashEnumerators=true, false`

default: `false`

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

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

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

```
652 \@@_add_lua_option:nnn
653   { hashEnumerators }
654   { boolean }
655   { false }

656 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.

```
657 \@@_add_lua_option:nnn
658   { headerAttributes }
659   { boolean }
660   { false }

661 defaultOptions.headerAttributes = false
```

`html=true, false`

default: `true`

- true** Enable the recognition of inline HTML tags, block HTML elements, HTML comments, HTML instructions, and entities in the input. Inline HTML tags, block HTML elements and HTML comments will be rendered, HTML instructions will be ignored, and HTML entities will be replaced with the corresponding Unicode codepoints.
- false** Disable the recognition of HTML markup. Any HTML markup in the input will be rendered as plain text.

```
662 \@@_add_lua_option:nnn
663   { html }
664   { boolean }
665   { true }

666 defaultOptions.html = true
```

`htmlOverLinks=true, false`

default: `false`

- true** When the option `HTML` is enabled and a text can be understood either as a hyperlink or HTML, interpret it as HTML.
- This is especially relevant when the option `relativeReferences` is enabled, since it makes e.g. `</foo>` a valid hyperlink.
- false** When the option `HTML` is enabled and a text can be understood either as a hyperlink or HTML, interpret it as a hyperlink.

This is an experimental option. Whenever the option `experimental` is enabled and this option is unspecified, it will be enabled. Like other experimental options, this option will be enabled by default and soft-deprecated in the next major release of the Markdown package.

```
667 \@@_add_experimental_lua_option:n
668   { htmlOverLinks }

669 defaultOptions.htmlOverLinks = false
670 experimentalOptions.htmlOverLinks = true
```

`hybrid=true, false`

default: `false`

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

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

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

Consider one of the following better alternatives for mixing T<sub>E</sub>X and markdown:

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

```
Here is a mathematical formula:

/math-formula.tex
```

- With the `rawAttribute` option, authors can denote raw text spans and code blocks that will be interpreted as T<sub>E</sub>X code:

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

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

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

```
$H_2 O$ is a liquid.

Here is a mathematical formula:
\[distance[i] =
```

```

\begin{dcases}
  a & b \\
  c & d
\end{dcases}
\]

```

```

671 \@@_add_lua_option:nnn
672   { hybrid }
673   { boolean }
674   { false }
675 defaultOptions.hybrid = false

```

`inlineCodeAttributes=true, false` default: false

**true** Enable the Pandoc inline code span attribute extension<sup>17</sup>:

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

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

```

676 \@@_add_lua_option:nnn
677   { inlineCodeAttributes }
678   { boolean }
679   { false }
680 defaultOptions.inlineCodeAttributes = false

```

`inlineNotes=true, false` default: false

**true** Enable the Pandoc inline note syntax extension<sup>18</sup>:

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

**false** Disable the Pandoc inline note syntax extension.

```

681 \@@_add_lua_option:nnn
682   { inlineNotes }
683   { boolean }
684   { false }
685 defaultOptions.inlineNotes = false

```

<sup>17</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-inline_code_attributes).

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

`jeekyllData=true, false`

default: false

**true** Enable the Pandoc YAML metadata block syntax extension<sup>19</sup> for entering metadata in YAML:

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

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

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

```
686 \@@_add_lua_option:nnn
687 { jeekyllData }
688 { boolean }
689 { false }
690 defaultOptions.jekyllData = false
```

`linkAttributes=true, false`

default: false

**true** Enable the Pandoc link and image attribute syntax extension<sup>20</sup>:

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

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

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

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

<sup>19</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).

<sup>20</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

`lineBlocks=true, false`

default: false

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

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

`false` Disable the Pandoc line block syntax extension.

```
696 \@@_add_lua_option:nnn
697   { lineBlocks }
698   { boolean }
699   { false }

700 defaultOptions.lineBlocks = false
```

`mark=true, false`

default: false

`true` Enable the Pandoc mark syntax extension<sup>22</sup>:

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

`false` Disable the Pandoc mark syntax extension.

```
701 \@@_add_lua_option:nnn
702   { mark }
703   { boolean }
704   { false }

705 defaultOptions.mark = false
```

`notes=true, false`

default: false

`true` Enable the Pandoc note syntax extension<sup>23</sup>:

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

[^1]: Here is the note.

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

<sup>21</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

<sup>22</sup>See <https://pandoc.org/MANUAL.html#extension-mark>.

<sup>23</sup>See <https://pandoc.org/MANUAL.html#extension-footnotes>.



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.

```
706 \@@_add_lua_option:nnn
707   { notes }
708   { boolean }
709   { false }
710 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.

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

`preserveTabs=true, false`

default: `true`

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

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

```
716 \@@_add_lua_option:nnn
717   { preserveTabs }
718   { boolean }
719   { true }

720 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: `false`

`true` Enable the Pandoc raw attribute syntax extension<sup>24</sup>.

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

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

Here is a mathematical formula:

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

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

`false` Disable the Pandoc raw attribute syntax extension.

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

725 defaultOptions.rawAttribute = false
```

---

<sup>24</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).

`relativeReferences=true, false`

default: false

`true` Enable relative references<sup>25</sup> in autolinks:

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

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

`false` Disable relative references in autolinks.

```
726 \@@_add_lua_option:nnn
727   { relativeReferences }
728   { boolean }
729   { false }
```

```
730 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.

```
731 \@@_add_lua_option:nnn
732   { shiftHeadings }
733   { number }
734   { 0 }
```

```
735 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.

---

<sup>25</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

- $\wedge\langle identifier \rangle$  selects the beginning of a section (see the `headerAttributes` option) or a fenced div (see the `fencedDivs` option) with the HTML attribute `#\langle identifier \rangle`.
- $\$ \langle identifier \rangle$  selects the end of a section with the HTML attribute `#\langle identifier \rangle`.
- $\langle identifier \rangle$  corresponds to  $\wedge\langle identifier \rangle$  for the first selector and to  $\$ \langle identifier \rangle$  for the second selector.

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

```

736 \@@_add_lua_option:nnn
737   { slice }
738   { slice }
739   { ^~$ }
740 defaultOptions.slice = "^ $"

```

`smartEllipses=true, false` default: false

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

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

```

741 \@@_add_lua_option:nnn
742   { smartEllipses }
743   { boolean }
744   { false }
745 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 `\markdownRenderOlItemWithNumber`  $\TeX$  macro.

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

```

746 \@@_add_lua_option:nnn
747   { startNumber }
748   { boolean }
749   { true }
750 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

**true** Enable the Pandoc strike-through syntax extension<sup>26</sup>:

```
This ~~is deleted text.~~
```

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

```
751 \@@_add_lua_option:nnn
752 { strikeThrough }
753 { boolean }
754 { false }
755 defaultOptions.strikeThrough = false
```

`stripIndent=true, false`

default: false

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

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

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

```
756 \@@_add_lua_option:nnn
757 { stripIndent }
758 { boolean }
759 { false }
760 defaultOptions.stripIndent = false
```

---

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

`subscripts=true, false`

default: false

`true` Enable the Pandoc subscript syntax extension<sup>27</sup>:

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

`false` Disable the Pandoc subscript syntax extension.

```
761 \@@_add_lua_option:nnn
762 { subscripts }
763 { boolean }
764 { false }

765 defaultOptions.subscripts = false
```

`superscripts=true, false`

default: false

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

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

`false` Disable the Pandoc superscript syntax extension.

```
766 \@@_add_lua_option:nnn
767 { superscripts }
768 { boolean }
769 { false }

770 defaultOptions.superscripts = false
```

`tableAttributes=true, false`

default: false

`true`

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

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

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

<sup>27</sup>See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

<sup>28</sup>See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

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

```
771 \@@_add_lua_option:nnn
772   { tableAttributes }
773   { boolean }
774   { false }

775 defaultOptions.tableAttributes = false
```

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

**true**

: Enable the Pandoc table caption syntax extension<sup>29</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.

```
776 \@@_add_lua_option:nnn
777 { tableCaptions }
778 { boolean }
779 { false }

780 defaultOptions.tableCaptions = false
```

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

**true**      Enable the Pandoc task list syntax extension<sup>30</sup>:

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

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

<sup>29</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>30</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

```

781 \@@_add_lua_option:nnn
782 { taskLists }
783 { boolean }
784 { false }
785 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.

```

786 \@@_add_lua_option:nnn
787 { texComments }
788 { boolean }
789 { false }
790 defaultOptions.texComments = false

```

`texMathDollars=true, false`

default: false

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

```

inline math: $E=mc^2$

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

```

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

```

791 \@@_add_lua_option:nnn
792 { texMathDollars }
793 { boolean }
794 { false }
795 defaultOptions.texMathDollars = false

```

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



`texMathDoubleBackslash=true, false` default: false

**true** Enable the Pandoc double backslash math syntax extension<sup>32</sup>:

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

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

```
796 \@@_add_lua_option:nnn
797 { texMathDoubleBackslash }
798 { boolean }
799 { false }
800 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

**true** Enable the Pandoc single backslash math syntax extension<sup>33</sup>:

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

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

```
801 \@@_add_lua_option:nnn
802 { texMathSingleBackslash }
803 { boolean }
804 { false }
805 defaultOptions.texMathSingleBackslash = false
```

`tightLists=true, false` default: true

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

---

<sup>32</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>33</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_single\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash).

```

- This is
- a tight
- unordered list.

- This is

 not a tight

- unordered list.

```

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

```

806 \@@_add_lua_option:nnn
807 { tightLists }
808 { boolean }
809 { true }

810 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.

```

811 \@@_add_lua_option:nnn
812 { underscores }
813 { boolean }
814 { true }
815 \ExplSyntaxOff

816 defaultOptions.underscores = true

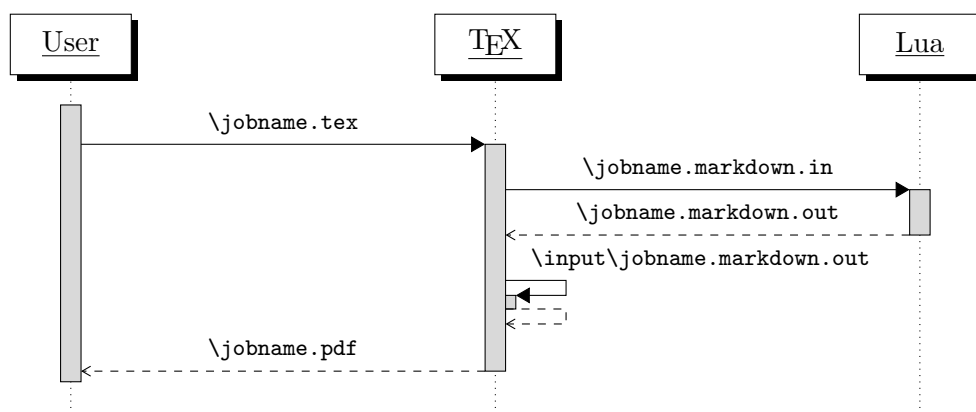
```

### 2.1.7 Command-Line Interface

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

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

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

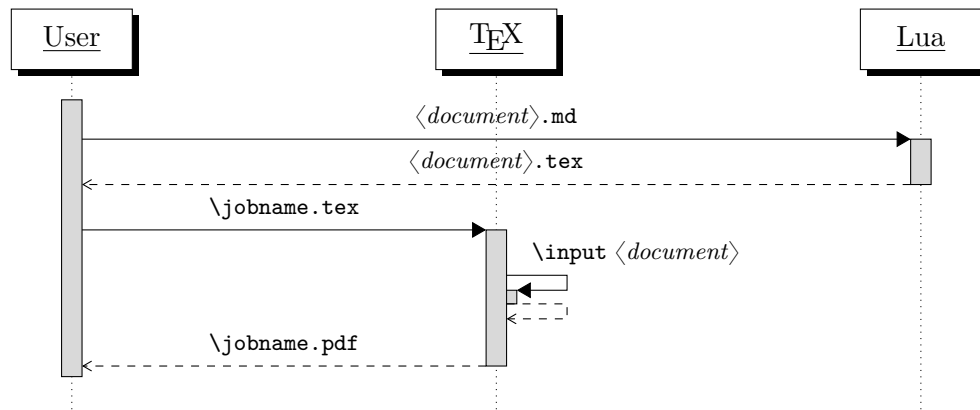


**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the  $\text{\TeX}$  interface**

```

817 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
818 .SH NAME
819 markdown2tex \- convert .md files to .tex
820 .SH SYNOPSIS
</lua-cli-manpage> <*lua-cli>
821 local HELP_STRING = "Usage: " .. [[
</lua-cli> <*lua-cli,lua-cli-manpage>
822 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
823
</lua-cli,lua-cli-manpage> <*lua-cli-manpage>
824 .SH DESCRIPTION

```



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

```

825 % \end{macrocode}
826 </lua-cli-manpage>
827 <*lua-cli, lua-cli-manpage>
828 % \begin{macrocode}
829 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
830 Manual (https://ctan.org/pkg/markdown).
831
832 When OUTPUT_FILE is unspecified, the result of the conversion will be
833 written to the standard output. When INPUT_FILE is also unspecified, the
834 result of the conversion will be read from the standard input.
835 % \end{macrocode}
836 </lua-cli, lua-cli-manpage>
837 <*lua-cli>
838 % \begin{macrocode}
839
840 Report bugs to: witiko@mail.muni.cz
841 Markdown package home page: <https://github.com/witiko/markdown>]]
842
843 local VERSION_STRING = [[
844 markdown2tex (Markdown)]] .. metadata.version .. [[
845
846 Copyright (C)]] .. table.concat(metadata.copyright,
847 "\nCopyright (C) ") .. [[
848
849 License:]] .. metadata.license
850
851 local function warn(s)
852 io.stderr:write("Warning: " .. s .. "\n")
853 end

```

```

854
855 local function error(s)
856 io.stderr:write("Error: " .. s .. "\n")
857 os.exit(1)
858 end

```

To make it easier to copy-and-paste options from Pandoc [7] 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 [8] also show that snake\_case is faster to read than camelCase.

```

859 local function camel_case(option_name)
860 local cased_option_name = option_name:gsub("_(%l)", function(match)
861 return match:sub(2, 2):upper()
862 end)
863 return cased_option_name
864 end
865
866 local function snake_case(option_name)
867 local cased_option_name = option_name:gsub("%l%u", function(match)
868 return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
869 end)
870 return cased_option_name
871 end
872
873 local cases = {camel_case, snake_case}
874 local various_case_options = {}
875 for option_name, _ in pairs(defaultOptions) do
876 for _, case in ipairs(cases) do
877 various_case_options[case(option_name)] = option_name
878 end
879 end
880
881 local process_options = true
882 local options = {}
883 local input_filename
884 local output_filename
885 for i = 1, #arg do
886 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.

```

887 if arg[i] == "--" then
888 process_options = false
889 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.

```

890 elseif arg[i]:match("=") then
891 local key, value = arg[i]:match("(.-)=(.*)")
892 if defaultOptions[key] == nil and
893 various_case_options[key] ~= nil then
894 key = various_case_options[key]
895 end

```

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

```

896 local default_type = type(defaultOptions[key])
897 if default_type == "boolean" then
898 options[key] = (value == "true")
899 elseif default_type == "number" then
900 options[key] = tonumber(value)
901 elseif default_type == "table" then
902 options[key] = {}
903 for item in value:gmatch("[^,]+") do
904 table.insert(options[key], item)
905 end
906 else
907 if default_type ~= "string" then
908 if default_type == "nil" then
909 warn('Option "' .. key .. '" not recognized.')
910 else
911 warn('Option "' .. key .. '" type not recognized, ' ..
912 'please file a report to the package maintainer.')
913 end
914 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
915 key .. '" as a string.')
916 end
917 options[key] = value
918 end
919 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.

```

920 elseif arg[i] == "--help" or arg[i] == "-h" then
921 print(HELP_STRING)
922 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.

```

923 elseif arg[i] == "--version" or arg[i] == "-v" then
924 print(VERSION_STRING)
925 os.exit()
926 end
927 end

```

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

```

928 if input_filename == nil then
929 input_filename = arg[i]

```

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

```

930 elseif output_filename == nil then
931 output_filename = arg[i]
932 else
933 error('Unexpected argument: "' .. arg[i] .. "'')
934 end
935 ::continue::
936 end

```

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

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

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

```
\input hello
```

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

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

```

937 \def\markdownLastModified{((LASTMODIFIED))}%
938 \def\markdownVersion{((VERSION))}%

```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown and YAML

The interface exposes the  $\text{\markdownBegin}$ ,  $\text{\markdownEnd}$ ,  $\text{\yamlBegin}$ ,  $\text{\yamlEnd}$ ,  $\text{\markinline}$ ,  $\text{\markdownInput}$ ,  $\text{\yamlInput}$ , and  $\text{\markdownEscape}$  macros.

#### 2.2.1.1 Typesetting Markdown and YAML directly

The  $\text{\markdownBegin}$  macro marks the beginning of a markdown document fragment and the  $\text{\markdownEnd}$  macro marks its end.

```
939 \let\markdownBegin\relax
940 \let\markdownEnd\relax
```

You may prepend your own code to the  $\text{\markdownBegin}$  macro and redefine the  $\text{\markdownEnd}$  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  $\text{\markdownEnd}$  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  $\text{\markdownEnd}$  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  $\text{\TeX}$  [9, p. 46]. As a corollary, the  $\text{\markdownBegin}$  macro also ignores them.

The  $\text{\markdownBegin}$  and  $\text{\markdownEnd}$  macros will also consume the rest of the lines at which they appear. In the following example plain  $\text{\TeX}$  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  $\text{\markdownBegin}$  and  $\text{\markdownEnd}$  macros.

The following example plain  $\text{\TeX}$  code showcases the usage of the  $\text{\markdownBegin}$  and  $\text{\markdownEnd}$  macros:



```

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

```

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

```

941 \let\uyamlBegin\uyamlBegin
942 \def\uyamlEnd{\uyamlEnd\endgroup}

```

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

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

```

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

```

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

```

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

```

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

```

943 \let\markinline\markinline

```

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

```

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

```

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

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

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

### 2.2.1.2 Typesetting Markdown and YAML from external documents

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

```
944 \let\markdownInput\relax
```

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

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

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

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

```
945 \def\yamlInput#1{%
946 \begingroup
947 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
948 \markdownInput{#1}%
949 \endgroup
950 }%
```

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

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

```

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

```

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

```

\input markdown
\yamlSetup{jeekyllData, expectJeekyllData, ensureJeekyllData}
\markdownInput{hello.yml}
\bye

```

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

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

```

951 \let\markdownEscape\relax

```

### 2.2.2 Options

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

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

```

952 \ExplSyntaxOn
953 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
954 \cs_generate_variant:Nn
955 \tl_const:Nn
956 { NV }
957 \tl_if_exist:NF
958 \c_@@_top_layer_tl
959 {
960 \tl_const:NV
961 \c_@@_top_layer_tl
962 \c_@@_option_layer_plain_tex_tl
963 }

```

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

```

964 \seq_new:N \g_@@_plain_tex_options_seq

```

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

```

965 \prop_new:N \g_@@_plain_tex_option_types_prop
966 \prop_new:N \g_@@_default_plain_tex_options_prop
967 \seq_new:N \g_@@_experimental_plain_tex_options_seq
968 \seq_gput_right:NV
969 \g_@@_option_layers_seq
970 \c_@@_option_layer_plain_tex_tl
971 \cs_new:Nn
972 \@@_add_plain_tex_option:nnn
973 {
974 \@@_add_option:Vnnn
975 \c_@@_option_layer_plain_tex_tl
976 { #1 }
977 { #2 }
978 { #3 }
979 }

```

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

```

980 \cs_new:Nn
981 \@@_setup:n
982 {
983 \keys_set:nn
984 { markdown/options }
985 { #1 }
986 }
987 \cs_gset_eq:NN
988 \markdownSetup
989 \@@_setup:n

```

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

```

990 \cs_gset_eq:NN
991 \yamlSetup
992 \markdownSetup

```

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

```

993 \prg_new_conditional:Nnn
994 \@@_if_option:n

```

```

995 { TF, T, F }
996 {
997 \@@_get_option_type:nN
998 { #1 }
999 \l_tmpa_tl
1000 \str_if_eq:NNF
1001 \l_tmpa_tl
1002 \c_@@_option_type_boolean_tl
1003 {
1004 \msg_error:nxxx
1005 { markdown }
1006 { expected-boolean-option }
1007 { #1 }
1008 { \l_tmpa_tl }
1009 }
1010 \@@_get_option_value:nN
1011 { #1 }
1012 \l_tmpa_tl
1013 \str_if_eq:NNTF
1014 \l_tmpa_tl
1015 \c_@@_option_value_true_tl
1016 { \prg_return_true: }
1017 { \prg_return_false: }
1018 }
1019 \msg_new:nnn
1020 { markdown }
1021 { expected-boolean-option }
1022 {
1023 Option~#1~has~type~#2,~
1024 but~a~boolean~was~expected.
1025 }
1026 \let
1027 \markdownIfOption
1028 \@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

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

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

```

1029 \@@_add_plain_tex_option:nnn
1030 { frozenCache }
1031 { boolean }
1032 { false }

```

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

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

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

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

```

1033 \tl_set:Nn
1034 \l_tmpa_tl
1035 {
1036 \str_use:N
1037 \g_@@_unquoted_jobname_str
1038 .markdown.in
1039 }
1040 \bool_if:NT
1041 \g_@@_jobname_quoted_bool
1042 {
1043 \tl_put_left:Nn
1044 \l_tmpa_tl
1045 { " }
1046 \tl_put_right:Nn
1047 \l_tmpa_tl
1048 { " }
1049 }
1050 \cs_generate_variant:Nn
1051 \@@_add_plain_tex_option:nnn
1052 { nnV }
1053 \@@_add_plain_tex_option:nnV
1054 { inputTempFileName }
1055 { path }
1056 \l_tmpa_tl

```

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

or, since T<sub>E</sub>X Live 2024, to the value of the `-output-directory` option of your T<sub>E</sub>X engine.

In MikT<sub>E</sub>X, this automatic detection is currently only supported with LuaT<sub>E</sub>X<sup>34</sup>. If you need to use MikT<sub>E</sub>X and cannot use LuaT<sub>E</sub>X, you can either a) fix the automatic detection by setting the environmental variable `TEXMF_OUTPUT_DIRECTORY` manually or by setting the `\markdownOptionOutputDir` option manually.

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

```
1057 \@@_add_plain_tex_option:nnn
1058 { outputDir }
1059 { path }
1060 { . }
```

### 2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section 2.2.3). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

---

<sup>34</sup>See <https://github.com/MiKTeX/miktex/issues/1630>.

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

```
1061 \@@_add_plain_tex_option:nnn
1062 { plain }
1063 { boolean }
1064 { false }
```

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

Here is how you would enable the macro in a  $\text{\LaTeX}$  document:

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

Here is how you would enable the macro in a  $\text{ConTeXt}$  document:

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

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

```
1065 \@@_add_plain_tex_option:nnn
1066 { noDefaults }
1067 { boolean }
1068 { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing  $\text{\TeX}$  package documentation using the Doc  $\text{\LaTeX}$  package [10] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
1069 \seq_gput_right:Nn
1070 \g_@@_plain_tex_options_seq
1071 { stripPercentSigns }
1072 \prop_gput:Nnn
1073 \g_@@_plain_tex_option_types_prop
1074 { stripPercentSigns }
1075 { boolean }
1076 \prop_gput:Nnx
1077 \g_@@_default_plain_tex_options_prop
1078 { stripPercentSigns }
1079 { false }
```



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

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

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

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

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

```
1080 \cs_new:Nn
1081 \@@_define_option_commands_and_keyvals:
1082 {
1083 \seq_map_inline:Nn
1084 \g_@@_option_layers_seq
1085 {
1086 \seq_map_inline:cn
1087 { g_@@_ ##1 _options_seq }
1088 {
1089 \@@_define_option_command:n
1090 { #####1 }
```

To make it easier to copy-and-paste options from Pandoc [7] 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 [8] also show that `snake_case` is faster to read than `camelCase`.

```
1091 \@@_with_various_cases:nn
1092 { #####1 }
1093 {
1094 \@@_define_option_keyval:nnn
1095 { ##1 }
1096 { #####1 }
1097 { #####1 }
1098 }
1099 }
1100 }
1101 }
1102 \cs_new:Nn
1103 \@@_define_option_command:n
1104 {
```

For experimental options, redirect the option command to the option command `\markdownOptionExperimental`.

```

1105 \bool_set_false:N
1106 \l_tmpa_bool
1107 \seq_map_inline:Nn
1108 \g_@@_option_layers_seq
1109 {
1110 \seq_if_in:cnT
1111 { g_@@_experimental_ ##1 _options_seq }
1112 { #1 }
1113 {
1114 \bool_set_true:N
1115 \l_tmpa_bool
1116 \seq_map_break:
1117 }
1118 }
1119 \bool_if:NTF
1120 \l_tmpa_bool
1121 {
1122 _@@_option_tl_to_csname:nN
1123 { #1 }
1124 \l_tmpa_tl
1125 \cs_if_exist:cF
1126 { \l_tmpa_tl }
1127 {
1128 \cs_gset:cpn
1129 { \l_tmpa_tl }
1130 { \markdownOptionExperimental }
1131 }
1132 }
1133 {

```

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

```

1134 \str_if_eq:nnTF
1135 { #1 }
1136 { outputDir }
1137 { \@@_define_option_command_output_dir: }
1138 {

```

Do not override options defined before loading the package.

```

1139 \@@_option_tl_to_csname:nN
1140 { #1 }
1141 \l_tmpa_tl
1142 \cs_if_exist:cF
1143 { \l_tmpa_tl }
1144 {
1145 \@@_get_default_option_value:nN
1146 { #1 }
1147 \l_tmpa_tl

```

```

1148 \@@_set_option_value:nV
1149 { #1 }
1150 \l_tmpa_tl
1151 }
1152 }
1153 }
1154 }
1155 \ExplSyntaxOff
1156 \input lt3luabridge.tex

```

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

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

```

1157 \ExplSyntaxOn
1158 \cs_new:Nn
1159 \@@_define_option_command_output_dir:
1160 {
1161 \cs_if_free:NT
1162 \markdownOptionOutputDir
1163 {
1164 \bool_if:nTF
1165 {
1166 \cs_if_exist_p:N
1167 \luabridge_tl_set:Nn &&
1168 (
1169 \int_compare_p:nNn
1170 { \g_luabridge_method_int }
1171 =
1172 { \c_luabridge_method_directlua_int } ||
1173 \sys_if_shell_unrestricted_p:
1174)
1175 }
1176 {

```

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

```

1177 \group_begin:
1178 \cctab_select:N

```

```

1179 \c_str_cctab
1180 \luabridge_tl_set:Nn
1181 \l_tmpa_tl
1182 {
1183 print(
1184 (status.output_directory)
1185 or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1186 or~"."
1187)
1188 }
1189 \tl_gset:NV
1190 \markdownOptionOutputDir
1191 \l_tmpa_tl
1192 \group_end:
1193 }
1194 {
1195 \tl_gset:Nn
1196 \markdownOptionOutputDir
1197 { . }
1198 }
1199 }
1200 }
1201 \cs_new:Nn
1202 \@@_set_option_value:nn
1203 {
1204 \@@_define_option:n
1205 { #1 }
1206 \@@_get_option_type:nN
1207 { #1 }
1208 \l_tmpa_tl
1209 \str_if_eq:NNTF
1210 \c_@@_option_type_counter_tl
1211 \l_tmpa_tl
1212 {
1213 \@@_option_tl_to_csname:nN
1214 { #1 }
1215 \l_tmpa_tl
1216 \int_gset:cn
1217 { \l_tmpa_tl }
1218 { #2 }
1219 }
1220 {
1221 \@@_option_tl_to_csname:nN
1222 { #1 }
1223 \l_tmpa_tl
1224 \cs_set:cpn
1225 { \l_tmpa_tl }

```

```

1226 { #2 }
1227 }
1228 }
1229 \cs_generate_variant:Nn
1230 \@@_set_option_value:nn
1231 { nV }
1232 \cs_new:Nn
1233 \@@_define_option:n
1234 {
1235 \@@_option_tl_to_csname:nN
1236 { #1 }
1237 \l_tmpa_tl
1238 \cs_if_free:cT
1239 { \l_tmpa_tl }
1240 {
1241 \@@_get_option_type:nN
1242 { #1 }
1243 \l_tmpb_tl
1244 \str_if_eq:NNT
1245 \c_@@_option_type_counter_tl
1246 \l_tmpb_tl
1247 {
1248 \@@_option_tl_to_csname:nN
1249 { #1 }
1250 \l_tmpa_tl
1251 \int_new:c
1252 { \l_tmpa_tl }
1253 }
1254 }
1255 }
1256 \cs_new:Nn
1257 \@@_define_option_keyval:nnn
1258 {
1259 \prop_get:cnN
1260 { g_@@_ #1 _option_types_prop }
1261 { #2 }
1262 \l_tmpa_tl
1263 \str_if_eq:VVTF
1264 \l_tmpa_tl
1265 \c_@@_option_type_boolean_tl
1266 {
1267 \keys_define:nn
1268 { markdown/options }
1269 {

```

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

```

1270 #3 .code:n = {
1271 \tl_set:Nx
1272 \l_tmpa_tl
1273 {
1274 \str_case:nnF
1275 { ##1 }
1276 {
1277 { yes } { true }
1278 { no } { false }
1279 }
1280 { ##1 }
1281 }
1282 \@@_set_option_value:nV
1283 { #2 }
1284 \l_tmpa_tl
1285 },
1286 #3 .default:n = { true },
1287 }
1288 }
1289 {
1290 \keys_define:nn
1291 { markdown/options }
1292 {
1293 #3 .code:n = {
1294 \@@_set_option_value:nn
1295 { #2 }
1296 { ##1 }
1297 },
1298 }
1299 }

```

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.

```

1300 \str_if_eq:VVT
1301 \l_tmpa_tl
1302 \c_@@_option_type_clist_tl
1303 {
1304 \tl_set:Nn
1305 \l_tmpa_tl
1306 { #3 }
1307 \tl_reverse:N
1308 \l_tmpa_tl
1309 \str_if_eq:enF
1310 {

```

```

1311 \tl_head:V
1312 \l_tmpa_tl
1313 }
1314 { s }
1315 {
1316 \msg_error:nnn
1317 { markdown }
1318 { malformed-name-for-clist-option }
1319 { #3 }
1320 }
1321 \tl_set:Nx
1322 \l_tmpa_tl
1323 {
1324 \tl_tail:V
1325 \l_tmpa_tl
1326 }
1327 \tl_reverse:N
1328 \l_tmpa_tl
1329 \tl_put_right:Nn
1330 \l_tmpa_tl
1331 {
1332 .code:n = {
1333 \@@_get_option_value:nN
1334 { #2 }
1335 \l_tmpa_tl
1336 \clist_set:NV
1337 \l_tmpa_clist
1338 { \l_tmpa_tl , { ##1 } }
1339 \@@_set_option_value:nV
1340 { #2 }
1341 \l_tmpa_clist
1342 }
1343 }
1344 \keys_define:nV
1345 { markdown/options }
1346 \l_tmpa_tl
1347 }
1348 }
1349 \cs_generate_variant:Nn
1350 \clist_set:Nn
1351 { NV }
1352 \cs_generate_variant:Nn
1353 \keys_define:nn
1354 { nV }
1355 \prg_generate_conditional_variant:Nnn
1356 \str_if_eq:nn
1357 { en }

```

```

1358 { p, F }
1359 \msg_new:nnn
1360 { markdown }
1361 { malformed-name-for-clist-option }
1362 {
1363 Clist-option-name~#1~does~not~end~with~-s.
1364 }

```

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

```

1365 \str_if_eq:VVT
1366 \c_@@_top_layer_tl
1367 \c_@@_option_layer_plain_tex_tl
1368 {
1369 \@@_define_option_commands_and_keyvals:
1370 }
1371 \ExplSyntaxOff

```

### 2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a  $\text{\TeX}$  document (further referred to as *a theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer  $\text{\LaTeX}$  package, which provides similar functionality with its `\usetheme` macro [11, Section 15.1].

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

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

If `@<theme version>` is specified after `<theme name>`, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded.



If  $\langle theme\ version \rangle$  is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [12].

```

1372 \ExplSyntaxOn
1373 \keys_define:nn
1374 { markdown/options }
1375 {
1376 theme .code:n = {
1377 \@@_set_theme:n
1378 { #1 }
1379 },
1380 import .code:n = {
1381 \tl_set:Nn
1382 \l_tmpa_tl
1383 { #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.

```

1384 \tl_replace_all:NnV
1385 \l_tmpa_tl
1386 { / }
1387 \c_backslash_str
1388 \keys_set:nV
1389 { markdown/options/import }
1390 \l_tmpa_tl
1391 },
1392 }

```

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

```

1393 \seq_new:N
1394 \g_@@_theme_names_seq
1395 \seq_new:N
1396 \g_@@_theme_versions_seq
1397 \tl_new:N
1398 \g_@@_current_theme_tl
1399 \tl_gset:Nn
1400 \g_@@_current_theme_tl
1401 { }
1402 \seq_gput_right:NV
1403 \g_@@_theme_names_seq
1404 \g_@@_current_theme_tl

```

```

1405 \cs_new:Npn
1406 \markdownThemeVersion
1407 { }
1408 \seq_gput_right:NV
1409 \g_@@_theme_versions_seq
1410 \g_@@_current_theme_tl
1411 \cs_new:Nn
1412 \@@_set_theme:n
1413 {

```

First, we validate the theme name.

```

1414 \str_if_in:nnF
1415 { #1 }
1416 { / }
1417 {
1418 \msg_error:nnn
1419 { markdown }
1420 { unqualified-theme-name }
1421 { #1 }
1422 }
1423 \str_if_in:nnT
1424 { #1 }
1425 { _ }
1426 {
1427 \msg_error:nnn
1428 { markdown }
1429 { underscores-in-theme-name }
1430 { #1 }
1431 }

```

Next, we extract the theme version.

```

1432 \str_if_in:nnTF
1433 { #1 }
1434 { @ }
1435 {
1436 \regex_extract_once:nnN
1437 { (.*?) @ (.*?) }
1438 { #1 }
1439 \l_tmpa_seq
1440 \seq_gpop_left:NN
1441 \l_tmpa_seq
1442 \l_tmpa_tl
1443 \seq_gpop_left:NN
1444 \l_tmpa_seq
1445 \l_tmpa_tl
1446 \tl_gset:NV
1447 \g_@@_current_theme_tl
1448 \l_tmpa_tl

```

```

1449 \seq_gpop_left:NN
1450 \l_tmpa_seq
1451 \l_tmpa_tl
1452 \cs_gset:Npe
1453 \markdownThemeVersion
1454 {
1455 \tl_use:N
1456 \l_tmpa_tl
1457 }
1458 }
1459 {
1460 \tl_gset:Nn
1461 \g_@@_current_theme_tl
1462 { #1 }
1463 \cs_gset:Npn
1464 \markdownThemeVersion
1465 { latest }
1466 }

```

Next, we munge the theme name.

```

1467 \str_set:NV
1468 \l_tmpa_str
1469 \g_@@_current_theme_tl
1470 \str_replace_all:Nnn
1471 \l_tmpa_str
1472 { / }
1473 { _ }

```

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

```

1474 \tl_set:NV
1475 \l_tmpa_tl
1476 \g_@@_current_theme_tl
1477 \tl_put_right:Nn
1478 \g_@@_current_theme_tl
1479 { / }
1480 \seq_gput_right:NV
1481 \g_@@_theme_names_seq
1482 \g_@@_current_theme_tl
1483 \seq_gput_right:NV
1484 \g_@@_theme_versions_seq
1485 \markdownThemeVersion
1486 \@@_load_theme:VeV
1487 \l_tmpa_tl
1488 { \markdownThemeVersion }
1489 \l_tmpa_str

```

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

```

1490 \seq_gpop_right:NN
1491 \g_@@_theme_names_seq
1492 \l_tmpa_tl
1493 \seq_get_right:NN
1494 \g_@@_theme_names_seq
1495 \l_tmpa_tl
1496 \tl_gset:NV
1497 \g_@@_current_theme_tl
1498 \l_tmpa_tl
1499 \seq_gpop_right:NN
1500 \g_@@_theme_versions_seq
1501 \l_tmpa_tl
1502 \seq_get_right:NN
1503 \g_@@_theme_versions_seq
1504 \l_tmpa_tl
1505 \cs_gset:Npe
1506 \markdownThemeVersion
1507 {
1508 \tl_use:N
1509 \l_tmpa_tl
1510 }
1511 }
1512 \msg_new:nnnn
1513 { markdown }
1514 { unqualified-theme-name }
1515 { Won't~load~theme~with~unqualified~name~#1 }
1516 { Theme~names~must~contain~at~least~one~forward~slash }
1517 \msg_new:nnnn
1518 { markdown }
1519 { underscores-in-theme-name }
1520 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1521 { Theme~names~must~not~contain~underscores~in~their~names }
1522 \cs_generate_variant:Nn
1523 \tl_replace_all:Nnn
1524 { NnV }
1525 \cs_generate_variant:Nn
1526 \cs_gset:Npn
1527 { Npe }

```

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

```

1528 \prop_new:N
1529 \g_@@_plain_tex_built_in_themes_prop

```

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

**witiko/diagrams** A theme that typesets fenced code blocks with the infostrings `dot`, `mermaid`, and `plantuml` as figures with diagrams produced with the command `dot` from Graphviz tools, the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`, and the command `plantuml` from the package PlantUML, respectively.

The key-value attribute `caption` can be used to specify the caption of the figure and for the infostrings `dot` and `plantuml`, the key-value attribute `format` can be used to specify the output image format. The remaining attributes are treated as image attributes.

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

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

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

``` mermaid {caption="An example mindmap" width=9cm #mermaid}
mindmap
    root )base-idea(
```

```

        sub<br/>idea 1
            ((?))
        sub<br/>idea 2
            ((?))
        sub<br/>idea 3
            ((?))
        sub<br/>idea 4
            ((?))
    ...

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

 ' Diagram title
 title Simple Request-Response Flow

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

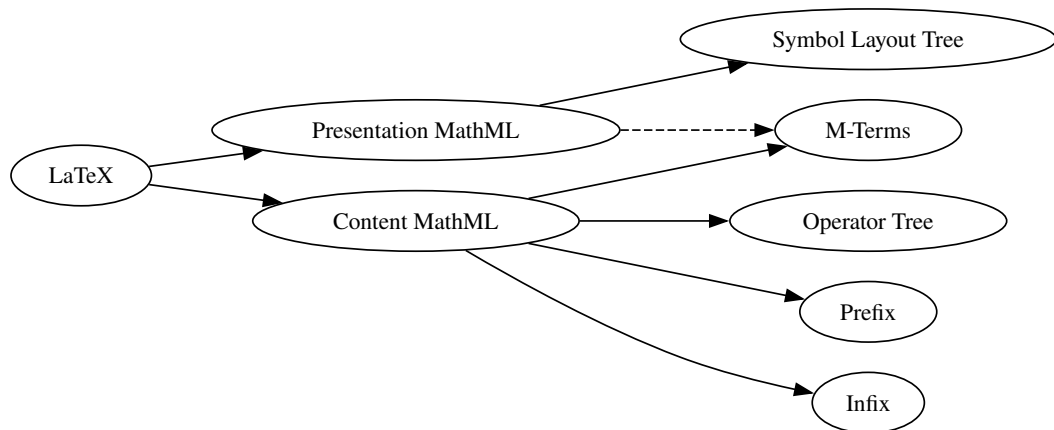
 alt Request is valid
 S -> DB: Query Data
 DB -> S: Return Data
 S -> C: Respond with Data
 else Request is invalid
 S -> C: Return Error
 end
 @enduml
 ...

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

```

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

The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package [@mermaid-js/mermaid-cli](https://www.npmjs.com/package/mermaid-cli), and PlantUML installed. All



**Figure 4: An example directed graph**

these packages are already included in the Docker image [witiko/markdown](#); consult [Dockerfile](#) to see how they are installed. The theme also requires shell access unless the [frozenCache](#) plain  $\text{\TeX}$  option is enabled.

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

```

\documentclass{article}
\usepackage[import=witiko/graphicx/http]{markdown}
\begin{document}
\begin{markdown}

 "The banner of the Markdown package"
\end{markdown}
\end{document}

```

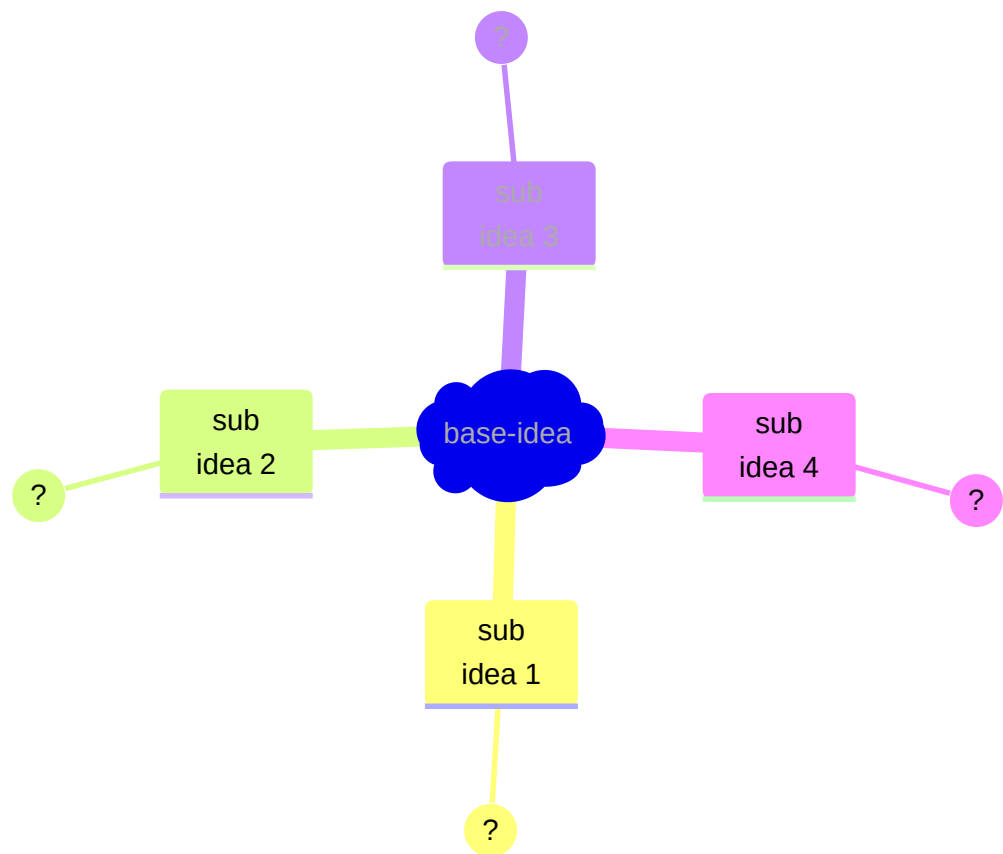
Typesetting the above document produces the output shown in [Figure 7](#). The theme requires the `catchfile`  $\text{\LaTeX}$  package and a Unix-like operating system with GNU Coreutils [md5sum](#) and either GNU Wget or cURL installed. The theme also requires shell access unless the [frozenCache](#)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

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

```

\input markdown

```



**Figure 5: An example mindmap**



## Simple Request-Response Flow

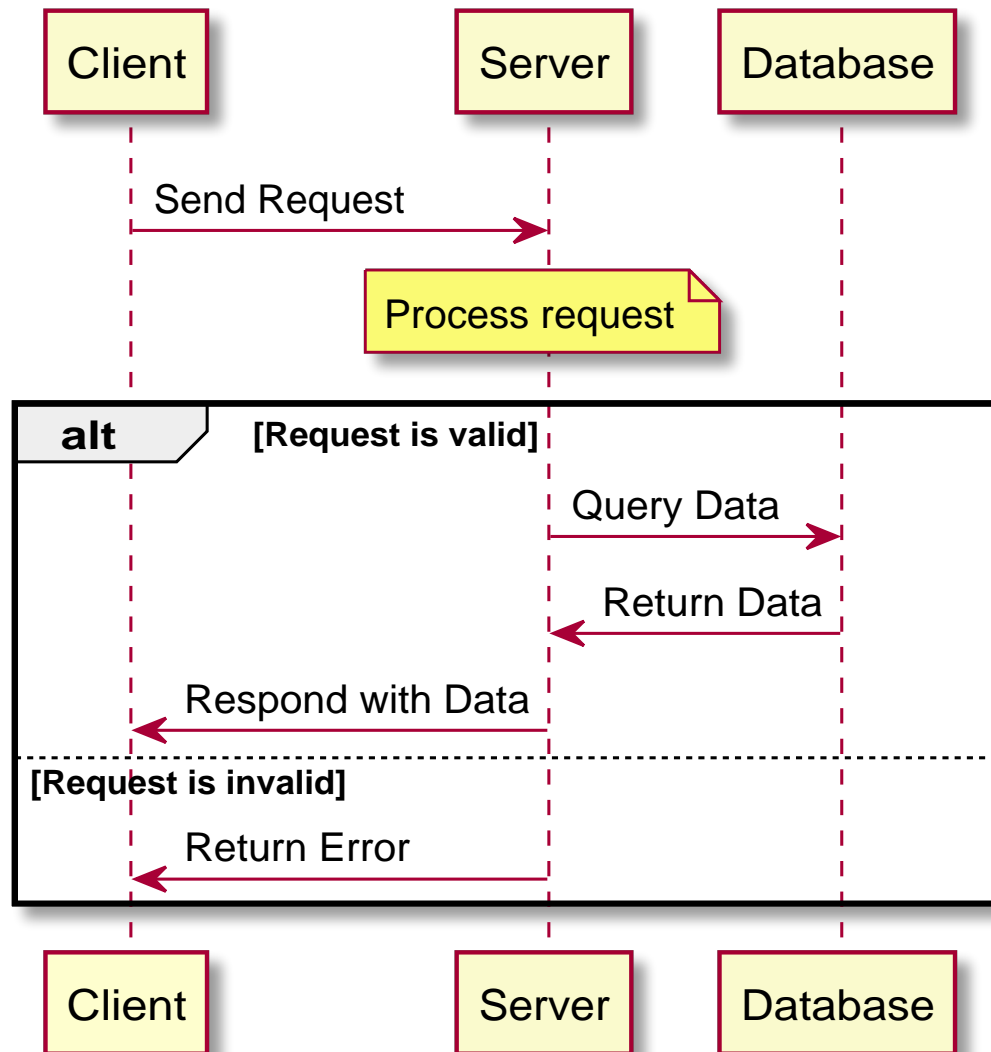


Figure 6: An example UML sequence diagram

```

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

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

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

```



## Chapter 1

# Introduction

### 1.1 Section

#### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

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

```

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

```

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

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

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

### 2.2.4 Snippets

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

```

1530 \prop_new:N
1531 \g_@@_snippets_prop
1532 \cs_new:Nn
1533 \@@_setup_snippet:nn
1534 {
1535 \tl_if_empty:nT
1536 { #1 }
1537 {
1538 \msg_error:nnn
1539 { markdown }
1540 { empty-snippet-name }
1541 { #1 }
1542 }
1543 \tl_set:NV
1544 \l_tmpa_tl
1545 \g_@@_current_theme_tl
1546 \tl_put_right:Nn
1547 \l_tmpa_tl
1548 { #1 }
1549 \@@_if_snippet_exists:nT
1550 { #1 }
1551 {
1552 \msg_warning:nnV
1553 { markdown }
1554 { redefined-snippet }
1555 \l_tmpa_tl
1556 }
1557 \keys_precompile:nnN
1558 { markdown/options }
1559 { #2 }
1560 \l_tmpb_tl
1561 \prop_gput:NVV
1562 \g_@@_snippets_prop
1563 \l_tmpa_tl
1564 \l_tmpb_tl
1565 }
1566 \cs_gset_eq:NN
1567 \markdownSetupSnippet
1568 \@@_setup_snippet:nn
1569 \msg_new:nnnn
1570 { markdown }
1571 { empty-snippet-name }
1572 { Empty~snippet~name~#1 }
1573 { Pick~a~non-empty~name~for~your~snippet }
1574 \msg_new:nnn
1575 { markdown }
1576 { redefined-snippet }

```

```
1577 { Redefined~snippet~#1 }
```

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

```
1578 \tl_new:N
1579 \l_@@_current_snippet_tl
1580 \prg_new_conditional:Nnn
1581 \@@_if_snippet_exists:n
1582 { TF, T }
1583 {
1584 \tl_set:NV
1585 \l_@@_current_snippet_tl
1586 \g_@@_current_theme_tl
1587 \tl_put_right:Nn
1588 \l_@@_current_snippet_tl
1589 { #1 }
1590 \prop_if_in:NVTF
1591 \g_@@_snippets_prop
1592 \l_@@_current_snippet_tl
1593 { \prg_return_true: }
1594 { \prg_return_false: }
1595 }
1596 \cs_gset_eq:NN
1597 \markdownIfSnippetExists
1598 \@@_if_snippet_exists:nTF
```

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

```
1599 \keys_define:nn
1600 { markdown/options }
1601 {
1602 snippet .code:n = {
1603 \tl_set:NV
1604 \l_tmpa_tl
1605 \g_@@_current_theme_tl
1606 \tl_put_right:Nn
1607 \l_tmpa_tl
1608 { #1 }
1609 \@@_if_snippet_exists:nTF
1610 { #1 }
1611 {
1612 \prop_get:NVN
1613 \g_@@_snippets_prop
1614 \l_tmpa_tl
1615 \l_tmpb_tl
1616 \tl_use:N
1617 \l_tmpb_tl
1618 }
1619 }
```

```

1620 \msg_error:nnV
1621 { markdown }
1622 { undefined-snippet }
1623 \l_tmpa_tl
1624 }
1625 }
1626 }
1627 \msg_new:nnn
1628 { markdown }
1629 { undefined-snippet }
1630 { Can't~invoke~undefined~snippet~#1 }
1631 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

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

```

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

```

The following ordered list will be preceded by roman numerals:

```
3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

```
3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

```
3. tres
4. quattuor

\end{markdown}
```

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

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

```
1632 \ExplSyntaxOn
1633 \tl_new:N
1634 \l_@@_import_current_theme_tl
1635 \keys_define:nn
1636 { markdown/options/import }
1637 {
```

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

```
1638 unknown .default:n = {},
1639 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.

```
1640 \tl_set_eq:NN
1641 \l_@@_import_current_theme_tl
1642 \l_keys_key_str
1643 \tl_replace_all:NVn
1644 \l_@@_import_current_theme_tl
1645 \c_backslash_str
1646 { / }
```

Here, we import the snippets.

```
1647 \clist_map_inline:nn
1648 { #1 }
```

```

1649 {
1650 \regex_extract_once:nnNTF
1651 { ^(.*)\s+as\s+(.*)$ }
1652 { ##1 }
1653 \l_tmpa_seq
1654 {
1655 \seq_pop:NN
1656 \l_tmpa_seq
1657 \l_tmpa_tl
1658 \seq_pop:NN
1659 \l_tmpa_seq
1660 \l_tmpa_tl
1661 \seq_pop:NN
1662 \l_tmpa_seq
1663 \l_tmpb_tl
1664 }
1665 {
1666 \tl_set:Nn
1667 \l_tmpa_tl
1668 { ##1 }
1669 \tl_set:Nn
1670 \l_tmpb_tl
1671 { ##1 }
1672 }
1673 \tl_put_left:Nn
1674 \l_tmpa_tl
1675 { / }
1676 \tl_put_left:NV
1677 \l_tmpa_tl
1678 \l_@@_import_current_theme_tl
1679 \@@_setup_snippet:Vx
1680 \l_tmpb_tl
1681 { snippet = { \l_tmpa_tl } }
1682 }

```

Here, we load the theme.

```

1683 \@@_set_theme:V
1684 \l_@@_import_current_theme_tl
1685 },
1686 }
1687 \cs_generate_variant:Nn
1688 \tl_replace_all:Nnn
1689 { NVn }
1690 \cs_generate_variant:Nn
1691 \@@_set_theme:n
1692 { V }
1693 \cs_generate_variant:Nn
1694 \@@_setup_snippet:nn

```



```
1695 { Vx }
```

## 2.2.5 Token Renderers

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

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

```
1696 \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.

```
1697 \prop_new:N \g_@@_renderer_arities_prop
```

```
1698 \ExplSyntaxOff
```

### 2.2.5.1 Attribute Renderers

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

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

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

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

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

```
1699 \ExplSyntaxOn
```

```
1700 \cs_gset_protected:Npn
```

```
1701 \markdownRendererAttributeIdentifier
```

```
1702 {
```

```
1703 \markdownRendererAttributeIdentifierPrototype
```

```
1704 }
```

```

1705 \seq_gput_right:Nn
1706 \g_@@_renderers_seq
1707 { attributeIdentifier }
1708 \prop_gput:Nnn
1709 \g_@@_renderer_arities_prop
1710 { attributeIdentifier }
1711 { 1 }
1712 \cs_gset_protected:Npn
1713 \markdownRendererAttributeClassName
1714 {
1715 \markdownRendererAttributeClassNamePrototype
1716 }
1717 \seq_gput_right:Nn
1718 \g_@@_renderers_seq
1719 { attributeClassName }
1720 \prop_gput:Nnn
1721 \g_@@_renderer_arities_prop
1722 { attributeClassName }
1723 { 1 }
1724 \cs_gset_protected:Npn
1725 \markdownRendererAttributeKeyValue
1726 {
1727 \markdownRendererAttributeKeyValuePrototype
1728 }
1729 \seq_gput_right:Nn
1730 \g_@@_renderers_seq
1731 { attributeKeyValue }
1732 \prop_gput:Nnn
1733 \g_@@_renderer_arities_prop
1734 { attributeKeyValue }
1735 { 2 }
1736 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1737 \ExplSyntaxOn
1738 \cs_gset_protected:Npn
1739 \markdownRendererBlockQuoteBegin
1740 {
1741 \markdownRendererBlockQuoteBeginPrototype
1742 }
1743 \seq_gput_right:Nn
1744 \g_@@_renderers_seq
1745 { blockQuoteBegin }
1746 \prop_gput:Nnn

```

```

1747 \g_@@_renderer_arities_prop
1748 { blockQuoteBegin }
1749 { 0 }
1750 \ExplSyntaxOff

```

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

```

1751 \ExplSyntaxOn
1752 \cs_gset_protected:Npn
1753 \markdownRendererBlockQuoteEnd
1754 {
1755 \markdownRendererBlockQuoteEndPrototype
1756 }
1757 \seq_gput_right:Nn
1758 \g_@@_renderers_seq
1759 { blockQuoteEnd }
1760 \prop_gput:Nnn
1761 \g_@@_renderer_arities_prop
1762 { blockQuoteEnd }
1763 { 0 }
1764 \ExplSyntaxOff

```

### 2.2.5.3 Bracketed Spans Attribute Context Renderers

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

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

```

1765 \ExplSyntaxOn
1766 \cs_gset_protected:Npn
1767 \markdownRendererBracketedSpanAttributeContextBegin
1768 {
1769 \markdownRendererBracketedSpanAttributeContextBeginPrototype
1770 }
1771 \seq_gput_right:Nn
1772 \g_@@_renderers_seq
1773 { bracketedSpanAttributeContextBegin }
1774 \prop_gput:Nnn
1775 \g_@@_renderer_arities_prop
1776 { bracketedSpanAttributeContextBegin }
1777 { 0 }
1778 \cs_gset_protected:Npn
1779 \markdownRendererBracketedSpanAttributeContextEnd
1780 {
1781 \markdownRendererBracketedSpanAttributeContextEndPrototype

```

```

1782 }
1783 \seq_gput_right:Nn
1784 \g_@@_renderers_seq
1785 { bracketedSpanAttributeContextEnd }
1786 \prop_gput:Nnn
1787 \g_@@_renderer_arities_prop
1788 { bracketedSpanAttributeContextEnd }
1789 { 0 }
1790 \ExplSyntaxOff

```

#### 2.2.5.4 Bullet List Renderers

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

```

1791 \ExplSyntaxOn
1792 \cs_gset_protected:Npn
1793 \markdownRendererUlBegin
1794 {
1795 \markdownRendererUlBeginPrototype
1796 }
1797 \seq_gput_right:Nn
1798 \g_@@_renderers_seq
1799 { ulBegin }
1800 \prop_gput:Nnn
1801 \g_@@_renderer_arities_prop
1802 { ulBegin }
1803 { 0 }
1804 \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.

```

1805 \ExplSyntaxOn
1806 \cs_gset_protected:Npn
1807 \markdownRendererUlBeginTight
1808 {
1809 \markdownRendererUlBeginTightPrototype
1810 }
1811 \seq_gput_right:Nn
1812 \g_@@_renderers_seq
1813 { ulBeginTight }
1814 \prop_gput:Nnn
1815 \g_@@_renderer_arities_prop
1816 { ulBeginTight }

```

```

1817 { 0 }
1818 \ExplSyntaxOff

```

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

```

1819 \ExplSyntaxOn
1820 \cs_gset_protected:Npn
1821 \markdownRenderUItem
1822 {
1823 \markdownRenderUItemPrototype
1824 }
1825 \seq_gput_right:Nn
1826 \g_@@_renderers_seq
1827 { ulItem }
1828 \prop_gput:Nnn
1829 \g_@@_render_arity_prop
1830 { ulItem }
1831 { 0 }
1832 \ExplSyntaxOff

```

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

```

1833 \ExplSyntaxOn
1834 \cs_gset_protected:Npn
1835 \markdownRenderUItemEnd
1836 {
1837 \markdownRenderUItemEndPrototype
1838 }
1839 \seq_gput_right:Nn
1840 \g_@@_renderers_seq
1841 { ulItemEnd }
1842 \prop_gput:Nnn
1843 \g_@@_render_arity_prop
1844 { ulItemEnd }
1845 { 0 }
1846 \ExplSyntaxOff

```

The `\markdownRenderUEnd` 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.

```

1847 \ExplSyntaxOn
1848 \cs_gset_protected:Npn
1849 \markdownRenderUEnd
1850 {
1851 \markdownRenderUEndPrototype
1852 }

```

```

1853 \seq_gput_right:Nn
1854 \g_@@_renderers_seq
1855 { ulEnd }
1856 \prop_gput:Nnn
1857 \g_@@_renderar_aritys_prop
1858 { ulEnd }
1859 { 0 }
1860 \ExplSyntaxOff

```

The `\markdownRenderarUleEndTight` 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.

```

1861 \ExplSyntaxOn
1862 \cs_gset_protected:Npn
1863 \markdownRenderarUleEndTight
1864 {
1865 \markdownRenderarUleEndTightPrototype
1866 }
1867 \seq_gput_right:Nn
1868 \g_@@_renderers_seq
1869 { ulEndTight }
1870 \prop_gput:Nnn
1871 \g_@@_renderar_aritys_prop
1872 { ulEndTight }
1873 { 0 }
1874 \ExplSyntaxOff

```

#### 2.2.5.5 Citation Renderers

The `\markdownRenderarCite` 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.

```

1875 \ExplSyntaxOn
1876 \cs_gset_protected:Npn
1877 \markdownRenderarCite
1878 {
1879 \markdownRenderarCitePrototype
1880 }
1881 \seq_gput_right:Nn
1882 \g_@@_renderers_seq
1883 { cite }
1884 \prop_gput:Nnn

```

```

1885 \g_@@_renderer_arities_prop
1886 { cite }
1887 { 1 }
1888 \ExplSyntaxOff

```

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

```

1889 \ExplSyntaxOn
1890 \cs_gset_protected:Npn
1891 \markdownRendererTextCite
1892 {
1893 \markdownRendererTextCitePrototype
1894 }
1895 \seq_gput_right:Nn
1896 \g_@@_renderers_seq
1897 { textCite }
1898 \prop_gput:Nnn
1899 \g_@@_renderer_arities_prop
1900 { textCite }
1901 { 1 }
1902 \ExplSyntaxOff

```

#### 2.2.5.6 Code Block Renderers

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

```

1903 \ExplSyntaxOn
1904 \cs_gset_protected:Npn
1905 \markdownRendererInputVerbatim
1906 {
1907 \markdownRendererInputVerbatimPrototype
1908 }
1909 \seq_gput_right:Nn
1910 \g_@@_renderers_seq
1911 { inputVerbatim }
1912 \prop_gput:Nnn
1913 \g_@@_renderer_arities_prop
1914 { inputVerbatim }
1915 { 1 }
1916 \ExplSyntaxOff

```

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

the code block contents, the fully escaped code fence infostring that can be directly typeset, and the raw code fence infostring that can be used outside typesetting.

```

1917 \ExplSyntaxOn
1918 \cs_gset_protected:Npn
1919 \markdownRendererInputFencedCode
1920 {
1921 \markdownRendererInputFencedCodePrototype
1922 }
1923 \seq_gput_right:Nn
1924 \g_@@_renderers_seq
1925 { inputFencedCode }
1926 \prop_gput:Nnn
1927 \g_@@_renderer_arities_prop
1928 { inputFencedCode }
1929 { 3 }
1930 \ExplSyntaxOff

```

### 2.2.5.7 Code Span Renderer

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

```

1931 \ExplSyntaxOn
1932 \cs_gset_protected:Npn
1933 \markdownRendererCodeSpan
1934 {
1935 \markdownRendererCodeSpanPrototype
1936 }
1937 \seq_gput_right:Nn
1938 \g_@@_renderers_seq
1939 { codeSpan }
1940 \prop_gput:Nnn
1941 \g_@@_renderer_arities_prop
1942 { codeSpan }
1943 { 1 }
1944 \ExplSyntaxOff

```

### 2.2.5.8 Code Span Attribute Context Renderers

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

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

```

1945 \ExplSyntaxOn
1946 \cs_gset_protected:Npn
1947 \markdownRendererCodeSpanAttributeContextBegin

```



```

1948 {
1949 \markdownRendererCodeSpanAttributeContextBeginPrototype
1950 }
1951 \seq_gput_right:Nn
1952 \g_@@_renderers_seq
1953 { codeSpanAttributeContextBegin }
1954 \prop_gput:Nnn
1955 \g_@@_renderer_arities_prop
1956 { codeSpanAttributeContextBegin }
1957 { 0 }
1958 \cs_gset_protected:Npn
1959 \markdownRendererCodeSpanAttributeContextEnd
1960 {
1961 \markdownRendererCodeSpanAttributeContextEndPrototype
1962 }
1963 \seq_gput_right:Nn
1964 \g_@@_renderers_seq
1965 { codeSpanAttributeContextEnd }
1966 \prop_gput:Nnn
1967 \g_@@_renderer_arities_prop
1968 { codeSpanAttributeContextEnd }
1969 { 0 }
1970 \ExplSyntaxOff

```

### 2.2.5.9 Content Block Renderers

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

```

1971 \ExplSyntaxOn
1972 \cs_gset_protected:Npn
1973 \markdownRendererContentBlock
1974 {
1975 \markdownRendererContentBlockPrototype
1976 }
1977 \seq_gput_right:Nn
1978 \g_@@_renderers_seq
1979 { contentBlock }
1980 \prop_gput:Nnn
1981 \g_@@_renderer_arities_prop
1982 { contentBlock }
1983 { 4 }
1984 \ExplSyntaxOff

```

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

```

1985 \ExplSyntaxOn
1986 \cs_gset_protected:Npn
1987 \markdownRendererContentBlockOnlineImage
1988 {
1989 \markdownRendererContentBlockOnlineImagePrototype
1990 }
1991 \seq_gput_right:Nn
1992 \g_@@_renderers_seq
1993 { contentBlockOnlineImage }
1994 \prop_gput:Nnn
1995 \g_@@_renderer_arities_prop
1996 { contentBlockOnlineImage }
1997 { 4 }
1998 \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>35</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s$ ,  $s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local T<sub>E</sub>X directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [6] is a good starting point.

```

1999 \ExplSyntaxOn
2000 \cs_gset_protected:Npn
2001 \markdownRendererContentBlockCode
2002 {
2003 \markdownRendererContentBlockCodePrototype
2004 }
2005 \seq_gput_right:Nn
2006 \g_@@_renderers_seq
2007 { contentBlockCode }
2008 \prop_gput:Nnn

```

---

<sup>35</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

2009 \g_@@_renderer_arities_prop
2010 { contentBlockCode }
2011 { 5 }
2012 \ExplSyntaxOff

```

#### 2.2.5.10 Definition List Renderers

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

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

```

2013 \ExplSyntaxOn
2014 \cs_gset_protected:Npn
2015 \markdownRendererDlBegin
2016 {
2017 \markdownRendererDlBeginPrototype
2018 }
2019 \seq_gput_right:Nn
2020 \g_@@_renderers_seq
2021 { dlBegin }
2022 \prop_gput:Nnn
2023 \g_@@_renderer_arities_prop
2024 { dlBegin }
2025 { 0 }
2026 \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.

```

2027 \ExplSyntaxOn
2028 \cs_gset_protected:Npn
2029 \markdownRendererDlBeginTight
2030 {
2031 \markdownRendererDlBeginTightPrototype
2032 }
2033 \seq_gput_right:Nn
2034 \g_@@_renderers_seq
2035 { dlBeginTight }
2036 \prop_gput:Nnn
2037 \g_@@_renderer_arities_prop
2038 { dlBeginTight }
2039 { 0 }
2040 \ExplSyntaxOff

```

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

```

2041 \ExplSyntaxOn
2042 \cs_gset_protected:Npn
2043 \markdownRendererDlItem
2044 {
2045 \markdownRendererDlItemPrototype
2046 }
2047 \seq_gput_right:Nn
2048 \g_@@_renderers_seq
2049 { dlItem }
2050 \prop_gput:Nnn
2051 \g_@@_renderer_arities_prop
2052 { dlItem }
2053 { 1 }
2054 \ExplSyntaxOff

```

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

```

2055 \ExplSyntaxOn
2056 \cs_gset_protected:Npn
2057 \markdownRendererDlItemEnd
2058 {
2059 \markdownRendererDlItemEndPrototype
2060 }
2061 \seq_gput_right:Nn
2062 \g_@@_renderers_seq
2063 { dlItemEnd }
2064 \prop_gput:Nnn
2065 \g_@@_renderer_arities_prop
2066 { dlItemEnd }
2067 { 0 }
2068 \ExplSyntaxOff

```

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

```

2069 \ExplSyntaxOn
2070 \cs_gset_protected:Npn
2071 \markdownRendererDlDefinitionBegin
2072 {
2073 \markdownRendererDlDefinitionBeginPrototype
2074 }
2075 \seq_gput_right:Nn
2076 \g_@@_renderers_seq
2077 { dlDefinitionBegin }
2078 \prop_gput:Nnn

```

```

2079 \g_@@_renderer_arities_prop
2080 { dlDefinitionBegin }
2081 { 0 }
2082 \ExplSyntaxOff

```

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

```

2083 \ExplSyntaxOn
2084 \cs_gset_protected:Npn
2085 \markdownRendererDlDefinitionEnd
2086 {
2087 \markdownRendererDlDefinitionEndPrototype
2088 }
2089 \seq_gput_right:Nn
2090 \g_@@_renderers_seq
2091 { dlDefinitionEnd }
2092 \prop_gput:Nnn
2093 \g_@@_renderer_arities_prop
2094 { dlDefinitionEnd }
2095 { 0 }
2096 \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.

```

2097 \ExplSyntaxOn
2098 \cs_gset_protected:Npn
2099 \markdownRendererDlEnd
2100 {
2101 \markdownRendererDlEndPrototype
2102 }
2103 \seq_gput_right:Nn
2104 \g_@@_renderers_seq
2105 { dlEnd }
2106 \prop_gput:Nnn
2107 \g_@@_renderer_arities_prop
2108 { dlEnd }
2109 { 0 }
2110 \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.

```

2111 \ExplSyntaxOn
2112 \cs_gset_protected:Npn

```

```

2113 \markdownRendererDlEndTight
2114 {
2115 \markdownRendererDlEndTightPrototype
2116 }
2117 \seq_gput_right:Nn
2118 \g_@@_renderers_seq
2119 { dlEndTight }
2120 \prop_gput:Nnn
2121 \g_@@_renderer_arities_prop
2122 { dlEndTight }
2123 { 0 }
2124 \ExplSyntaxOff

```

### 2.2.5.11 Ellipsis Renderer

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

```

2125 \ExplSyntaxOn
2126 \cs_gset_protected:Npn
2127 \markdownRendererEllipsis
2128 {
2129 \markdownRendererEllipsisPrototype
2130 }
2131 \seq_gput_right:Nn
2132 \g_@@_renderers_seq
2133 { ellipsis }
2134 \prop_gput:Nnn
2135 \g_@@_renderer_arities_prop
2136 { ellipsis }
2137 { 0 }
2138 \ExplSyntaxOff

```

### 2.2.5.12 Emphasis Renderers

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

```

2139 \ExplSyntaxOn
2140 \cs_gset_protected:Npn
2141 \markdownRendererEmphasis
2142 {
2143 \markdownRendererEmphasisPrototype
2144 }
2145 \seq_gput_right:Nn
2146 \g_@@_renderers_seq
2147 { emphasis }

```

```

2148 \prop_gput:Nnn
2149 \g_@@_renderer_arities_prop
2150 { emphasis }
2151 { 1 }
2152 \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.

```

2153 \ExplSyntaxOn
2154 \cs_gset_protected:Npn
2155 \markdownRendererStrongEmphasis
2156 {
2157 \markdownRendererStrongEmphasisPrototype
2158 }
2159 \seq_gput_right:Nn
2160 \g_@@_renderers_seq
2161 { strongEmphasis }
2162 \prop_gput:Nnn
2163 \g_@@_renderer_arities_prop
2164 { strongEmphasis }
2165 { 1 }
2166 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

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

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

```

2167 \ExplSyntaxOn
2168 \cs_gset_protected:Npn
2169 \markdownRendererFencedCodeAttributeContextBegin
2170 {
2171 \markdownRendererFencedCodeAttributeContextBeginPrototype
2172 }
2173 \seq_gput_right:Nn
2174 \g_@@_renderers_seq
2175 { fencedCodeAttributeContextBegin }
2176 \prop_gput:Nnn
2177 \g_@@_renderer_arities_prop
2178 { fencedCodeAttributeContextBegin }
2179 { 0 }
2180 \cs_gset_protected:Npn
2181 \markdownRendererFencedCodeAttributeContextEnd

```

```

2182 {
2183 \markdownRendererFencedCodeAttributeContextEndPrototype
2184 }
2185 \seq_gput_right:Nn
2186 \g_@@_renderers_seq
2187 { fencedCodeAttributeContextEnd }
2188 \prop_gput:Nnn
2189 \g_@@_renderer_arities_prop
2190 { fencedCodeAttributeContextEnd }
2191 { 0 }
2192 \ExplSyntaxOff

```

#### 2.2.5.14 Fenced Div Attribute Context Renderers

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

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

```

2193 \ExplSyntaxOn
2194 \cs_gset_protected:Npn
2195 \markdownRendererFencedDivAttributeContextBegin
2196 {
2197 \markdownRendererFencedDivAttributeContextBeginPrototype
2198 }
2199 \seq_gput_right:Nn
2200 \g_@@_renderers_seq
2201 { fencedDivAttributeContextBegin }
2202 \prop_gput:Nnn
2203 \g_@@_renderer_arities_prop
2204 { fencedDivAttributeContextBegin }
2205 { 0 }
2206 \cs_gset_protected:Npn
2207 \markdownRendererFencedDivAttributeContextEnd
2208 {
2209 \markdownRendererFencedDivAttributeContextEndPrototype
2210 }
2211 \seq_gput_right:Nn
2212 \g_@@_renderers_seq
2213 { fencedDivAttributeContextEnd }
2214 \prop_gput:Nnn
2215 \g_@@_renderer_arities_prop
2216 { fencedDivAttributeContextEnd }
2217 { 0 }
2218 \ExplSyntaxOff

```

#### 2.2.5.15 Header Attribute Context Renderers



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

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

```

2219 \ExplSyntaxOn
2220 \cs_gset_protected:Npn
2221 \markdownRendererHeaderAttributeContextBegin
2222 {
2223 \markdownRendererHeaderAttributeContextBeginPrototype
2224 }
2225 \seq_gput_right:Nn
2226 \g_@@_renderers_seq
2227 { headerAttributeContextBegin }
2228 \prop_gput:Nnn
2229 \g_@@_renderer_arities_prop
2230 { headerAttributeContextBegin }
2231 { 0 }
2232 \cs_gset_protected:Npn
2233 \markdownRendererHeaderAttributeContextEnd
2234 {
2235 \markdownRendererHeaderAttributeContextEndPrototype
2236 }
2237 \seq_gput_right:Nn
2238 \g_@@_renderers_seq
2239 { headerAttributeContextEnd }
2240 \prop_gput:Nnn
2241 \g_@@_renderer_arities_prop
2242 { headerAttributeContextEnd }
2243 { 0 }
2244 \ExplSyntaxOff

```

### 2.2.5.16 Heading Renderers

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

```

2245 \ExplSyntaxOn
2246 \cs_gset_protected:Npn
2247 \markdownRendererHeadingOne
2248 {
2249 \markdownRendererHeadingOnePrototype
2250 }
2251 \seq_gput_right:Nn
2252 \g_@@_renderers_seq
2253 { headingOne }
2254 \prop_gput:Nnn

```

```

2255 \g_@@_renderer_arities_prop
2256 { headingOne }
2257 { 1 }
2258 \ExplSyntaxOff

```

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

```

2259 \ExplSyntaxOn
2260 \cs_gset_protected:Npn
2261 \markdownRendererHeadingTwo
2262 {
2263 \markdownRendererHeadingTwoPrototype
2264 }
2265 \seq_gput_right:Nn
2266 \g_@@_renderers_seq
2267 { headingTwo }
2268 \prop_gput:Nnn
2269 \g_@@_renderer_arities_prop
2270 { headingTwo }
2271 { 1 }
2272 \ExplSyntaxOff

```

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

```

2273 \ExplSyntaxOn
2274 \cs_gset_protected:Npn
2275 \markdownRendererHeadingThree
2276 {
2277 \markdownRendererHeadingThreePrototype
2278 }
2279 \seq_gput_right:Nn
2280 \g_@@_renderers_seq
2281 { headingThree }
2282 \prop_gput:Nnn
2283 \g_@@_renderer_arities_prop
2284 { headingThree }
2285 { 1 }
2286 \ExplSyntaxOff

```

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

```

2287 \ExplSyntaxOn
2288 \cs_gset_protected:Npn
2289 \markdownRendererHeadingFour
2290 {
2291 \markdownRendererHeadingFourPrototype
2292 }

```

```

2293 \seq_gput_right:Nn
2294 \g_@@_renderers_seq
2295 { headingFour }
2296 \prop_gput:Nnn
2297 \g_@@_renderer_arities_prop
2298 { headingFour }
2299 { 1 }
2300 \ExplSyntaxOff

```

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

```

2301 \ExplSyntaxOn
2302 \cs_gset_protected:Npn
2303 \markdownRendererHeadingFive
2304 {
2305 \markdownRendererHeadingFivePrototype
2306 }
2307 \seq_gput_right:Nn
2308 \g_@@_renderers_seq
2309 { headingFive }
2310 \prop_gput:Nnn
2311 \g_@@_renderer_arities_prop
2312 { headingFive }
2313 { 1 }
2314 \ExplSyntaxOff

```

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

```

2315 \ExplSyntaxOn
2316 \cs_gset_protected:Npn
2317 \markdownRendererHeadingSix
2318 {
2319 \markdownRendererHeadingSixPrototype
2320 }
2321 \seq_gput_right:Nn
2322 \g_@@_renderers_seq
2323 { headingSix }
2324 \prop_gput:Nnn
2325 \g_@@_renderer_arities_prop
2326 { headingSix }
2327 { 1 }
2328 \ExplSyntaxOff

```

#### 2.2.5.17 Inline HTML Comment Renderer

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. This macro will only be produced, when the `html` option is

enabled. The macro receives a single argument that corresponds to the contents of the HTML comment.

```

2329 \ExplSyntaxOn
2330 \cs_gset_protected:Npn
2331 \markdownRendererInlineHtmlComment
2332 {
2333 \markdownRendererInlineHtmlCommentPrototype
2334 }
2335 \seq_gput_right:Nn
2336 \g_@@_renderers_seq
2337 { inlineHtmlComment }
2338 \prop_gput:Nnn
2339 \g_@@_renderer_arities_prop
2340 { inlineHtmlComment }
2341 { 1 }
2342 \ExplSyntaxOff

```

#### 2.2.5.18 HTML Tag and Element Renderers

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

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

```

2343 \ExplSyntaxOn
2344 \cs_gset_protected:Npn
2345 \markdownRendererInlineHtmlTag
2346 {
2347 \markdownRendererInlineHtmlTagPrototype
2348 }
2349 \seq_gput_right:Nn
2350 \g_@@_renderers_seq
2351 { inlineHtmlTag }
2352 \prop_gput:Nnn
2353 \g_@@_renderer_arities_prop
2354 { inlineHtmlTag }
2355 { 1 }
2356 \cs_gset_protected:Npn
2357 \markdownRendererInputBlockHtmlElement
2358 {
2359 \markdownRendererInputBlockHtmlElementPrototype
2360 }
2361 \seq_gput_right:Nn

```

```

2362 \g_@@_renderers_seq
2363 { inputBlockHtmlElement }
2364 \prop_gput:Nnn
2365 \g_@@_renderer_arities_prop
2366 { inputBlockHtmlElement }
2367 { 1 }
2368 \ExplSyntaxOff

```

### 2.2.5.19 Image Renderer

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

```

2369 \ExplSyntaxOn
2370 \cs_gset_protected:Npn
2371 \markdownRendererImage
2372 {
2373 \markdownRendererImagePrototype
2374 }
2375 \seq_gput_right:Nn
2376 \g_@@_renderers_seq
2377 { image }
2378 \prop_gput:Nnn
2379 \g_@@_renderer_arities_prop
2380 { image }
2381 { 4 }
2382 \ExplSyntaxOff

```

### 2.2.5.20 Image Attribute Context Renderers

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

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

```

2383 \ExplSyntaxOn
2384 \cs_gset_protected:Npn
2385 \markdownRendererImageAttributeContextBegin
2386 {
2387 \markdownRendererImageAttributeContextBeginPrototype
2388 }
2389 \seq_gput_right:Nn
2390 \g_@@_renderers_seq
2391 { imageAttributeContextBegin }
2392 \prop_gput:Nnn
2393 \g_@@_renderer_arities_prop

```

```

2394 { imageAttributeContextBegin }
2395 { 0 }
2396 \cs_gset_protected:Npn
2397 \markdownRendererImageAttributeContextEnd
2398 {
2399 \markdownRendererImageAttributeContextEndPrototype
2400 }
2401 \seq_gput_right:Nn
2402 \g_@@_renderers_seq
2403 { imageAttributeContextEnd }
2404 \prop_gput:Nnn
2405 \g_@@_renderer_arities_prop
2406 { imageAttributeContextEnd }
2407 { 0 }
2408 \ExplSyntaxOff

```

### 2.2.5.21 Interblock Separator Renderers

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

```

2409 \ExplSyntaxOn
2410 \cs_gset_protected:Npn
2411 \markdownRendererInterblockSeparator
2412 {
2413 \markdownRendererInterblockSeparatorPrototype
2414 }
2415 \seq_gput_right:Nn
2416 \g_@@_renderers_seq
2417 { interblockSeparator }
2418 \prop_gput:Nnn
2419 \g_@@_renderer_arities_prop
2420 { interblockSeparator }
2421 { 0 }
2422 \ExplSyntaxOff

```

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

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

```

2423 \ExplSyntaxOn
2424 \cs_gset_protected:Npn
2425 \markdownRendererParagraphSeparator
2426 {
2427 \markdownRendererParagraphSeparatorPrototype

```

```

2428 }
2429 \seq_gput_right:Nn
2430 \g_@@_renderers_seq
2431 { paragraphSeparator }
2432 \prop_gput:Nnn
2433 \g_@@_renderer_arities_prop
2434 { paragraphSeparator }
2435 { 0 }
2436 \ExplSyntaxOff

```

### 2.2.5.22 Line Block Renderers

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

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

```

2437 \ExplSyntaxOn
2438 \cs_gset_protected:Npn
2439 \markdownRendererLineBlockBegin
2440 {
2441 \markdownRendererLineBlockBeginPrototype
2442 }
2443 \seq_gput_right:Nn
2444 \g_@@_renderers_seq
2445 { lineBlockBegin }
2446 \prop_gput:Nnn
2447 \g_@@_renderer_arities_prop
2448 { lineBlockBegin }
2449 { 0 }
2450 \cs_gset_protected:Npn
2451 \markdownRendererLineBlockEnd
2452 {
2453 \markdownRendererLineBlockEndPrototype
2454 }
2455 \seq_gput_right:Nn
2456 \g_@@_renderers_seq
2457 { lineBlockEnd }
2458 \prop_gput:Nnn
2459 \g_@@_renderer_arities_prop
2460 { lineBlockEnd }
2461 { 0 }
2462 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2463 \ExplSyntaxOn
2464 \cs_gset_protected:Npn
2465 \markdownRendererSoftLineBreak
2466 {
2467 \markdownRendererSoftLineBreakPrototype
2468 }
2469 \seq_gput_right:Nn
2470 \g_@@_renderers_seq
2471 { softLineBreak }
2472 \prop_gput:Nnn
2473 \g_@@_renderer_arities_prop
2474 { softLineBreak }
2475 { 0 }
2476 \ExplSyntaxOff

```

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

```

2477 \ExplSyntaxOn
2478 \cs_gset_protected:Npn
2479 \markdownRendererHardLineBreak
2480 {
2481 \markdownRendererHardLineBreakPrototype
2482 }
2483 \seq_gput_right:Nn
2484 \g_@@_renderers_seq
2485 { hardLineBreak }
2486 \prop_gput:Nnn
2487 \g_@@_renderer_arities_prop
2488 { hardLineBreak }
2489 { 0 }
2490 \ExplSyntaxOff

```

#### 2.2.5.24 Link Renderer

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

```

2491 \ExplSyntaxOn
2492 \cs_gset_protected:Npn
2493 \markdownRendererLink
2494 {
2495 \markdownRendererLinkPrototype
2496 }
2497 \seq_gput_right:Nn
2498 \g_@@_renderers_seq
2499 { link }
2500 \prop_gput:Nnn

```



```

2501 \g_@@_renderer_arities_prop
2502 { link }
2503 { 4 }
2504 \ExplSyntaxOff

```

### 2.2.5.25 Link Attribute Context Renderers

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

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

```

2505 \ExplSyntaxOn
2506 \cs_gset_protected:Npn
2507 \markdownRendererLinkAttributeContextBegin
2508 {
2509 \markdownRendererLinkAttributeContextBeginPrototype
2510 }
2511 \seq_gput_right:Nn
2512 \g_@@_renderers_seq
2513 { linkAttributeContextBegin }
2514 \prop_gput:Nnn
2515 \g_@@_renderer_arities_prop
2516 { linkAttributeContextBegin }
2517 { 0 }
2518 \cs_gset_protected:Npn
2519 \markdownRendererLinkAttributeContextEnd
2520 {
2521 \markdownRendererLinkAttributeContextEndPrototype
2522 }
2523 \seq_gput_right:Nn
2524 \g_@@_renderers_seq
2525 { linkAttributeContextEnd }
2526 \prop_gput:Nnn
2527 \g_@@_renderer_arities_prop
2528 { linkAttributeContextEnd }
2529 { 0 }
2530 \ExplSyntaxOff

```

### 2.2.5.26 Marked Text Renderer

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

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

```

2531 \ExplSyntaxOn
2532 \cs_gset_protected:Npn

```

```

2533 \markdownRendererMark
2534 {
2535 \markdownRendererMarkPrototype
2536 }
2537 \seq_gput_right:Nn
2538 \g_@@_renderers_seq
2539 { mark }
2540 \prop_gput:Nnn
2541 \g_@@_renderer_arities_prop
2542 { mark }
2543 { 1 }
2544 \ExplSyntaxOff

```

### 2.2.5.27 Markdown Document Renderers

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

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

```

2545 \ExplSyntaxOn
2546 \cs_gset_protected:Npn
2547 \markdownRendererDocumentBegin
2548 {
2549 \markdownRendererDocumentBeginPrototype
2550 }
2551 \seq_gput_right:Nn
2552 \g_@@_renderers_seq
2553 { documentBegin }
2554 \prop_gput:Nnn
2555 \g_@@_renderer_arities_prop
2556 { documentBegin }
2557 { 0 }
2558 \cs_gset_protected:Npn
2559 \markdownRendererDocumentEnd
2560 {
2561 \markdownRendererDocumentEndPrototype
2562 }
2563 \seq_gput_right:Nn
2564 \g_@@_renderers_seq
2565 { documentEnd }
2566 \prop_gput:Nnn
2567 \g_@@_renderer_arities_prop
2568 { documentEnd }

```

```

2569 { 0 }
2570 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2571 \ExplSyntaxOn
2572 \cs_gset_protected:Npn
2573 \markdownRendererNbsp
2574 {
2575 \markdownRendererNbspPrototype
2576 }
2577 \seq_gput_right:Nn
2578 \g_@@_renderers_seq
2579 { nbsp }
2580 \prop_gput:Nnn
2581 \g_@@_renderer_arities_prop
2582 { nbsp }
2583 { 0 }
2584 \ExplSyntaxOff

```

### 2.2.5.29 Note Renderer

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

```

2585 \def\markdownRendererNote{%
2586 \markdownRendererNotePrototype}%
2587 \ExplSyntaxOn
2588 \seq_gput_right:Nn
2589 \g_@@_renderers_seq
2590 { note }
2591 \prop_gput:Nnn
2592 \g_@@_renderer_arities_prop
2593 { note }
2594 { 1 }
2595 \ExplSyntaxOff

```

### 2.2.5.30 Ordered List Renderers

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

```

2596 \ExplSyntaxOn
2597 \cs_gset_protected:Npn

```

```

2598 \markdownRendererOlBegin
2599 {
2600 \markdownRendererOlBeginPrototype
2601 }
2602 \seq_gput_right:Nn
2603 \g_@@_renderers_seq
2604 { olBegin }
2605 \prop_gput:Nnn
2606 \g_@@_renderer_arities_prop
2607 { olBegin }
2608 { 0 }
2609 \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.

```

2610 \ExplSyntaxOn
2611 \cs_gset_protected:Npn
2612 \markdownRendererOlBeginTight
2613 {
2614 \markdownRendererOlBeginTightPrototype
2615 }
2616 \seq_gput_right:Nn
2617 \g_@@_renderers_seq
2618 { olBeginTight }
2619 \prop_gput:Nnn
2620 \g_@@_renderer_arities_prop
2621 { olBeginTight }
2622 { 0 }
2623 \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`).

```

2624 \ExplSyntaxOn
2625 \cs_gset_protected:Npn
2626 \markdownRendererFancyOlBegin
2627 {
2628 \markdownRendererFancyOlBeginPrototype
2629 }
2630 \seq_gput_right:Nn
2631 \g_@@_renderers_seq

```

```

2632 { fancyOlBegin }
2633 \prop_gput:Nnn
2634 \g_@@_renderer_arities_prop
2635 { fancyOlBegin }
2636 { 2 }
2637 \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.

```

2638 \ExplSyntaxOn
2639 \cs_gset_protected:Npn
2640 \markdownRendererFancyOlBeginTight
2641 {
2642 \markdownRendererFancyOlBeginTightPrototype
2643 }
2644 \seq_gput_right:Nn
2645 \g_@@_renderers_seq
2646 { fancyOlBeginTight }
2647 \prop_gput:Nnn
2648 \g_@@_renderer_arities_prop
2649 { fancyOlBeginTight }
2650 { 2 }
2651 \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.

```

2652 \ExplSyntaxOn
2653 \cs_gset_protected:Npn
2654 \markdownRendererOlItem
2655 {
2656 \markdownRendererOlItemPrototype
2657 }
2658 \seq_gput_right:Nn
2659 \g_@@_renderers_seq
2660 { olItem }
2661 \prop_gput:Nnn
2662 \g_@@_renderer_arities_prop
2663 { olItem }
2664 { 0 }
2665 \ExplSyntaxOff

```

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

```

2666 \ExplSyntaxOn
2667 \cs_gset_protected:Npn
2668 \markdownRendererOlItemEnd
2669 {
2670 \markdownRendererOlItemEndPrototype
2671 }
2672 \seq_gput_right:Nn
2673 \g_@@_renderers_seq
2674 { olItemEnd }
2675 \prop_gput:Nnn
2676 \g_@@_renderer_arities_prop
2677 { olItemEnd }
2678 { 0 }
2679 \ExplSyntaxOff

```

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

```

2680 \ExplSyntaxOn
2681 \cs_gset_protected:Npn
2682 \markdownRendererOlItemWithNumber
2683 {
2684 \markdownRendererOlItemWithNumberPrototype
2685 }
2686 \seq_gput_right:Nn
2687 \g_@@_renderers_seq
2688 { olItemWithNumber }
2689 \prop_gput:Nnn
2690 \g_@@_renderer_arities_prop
2691 { olItemWithNumber }
2692 { 1 }
2693 \ExplSyntaxOff

```

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

```

2694 \ExplSyntaxOn
2695 \cs_gset_protected:Npn
2696 \markdownRendererFancyOlItem
2697 {
2698 \markdownRendererFancyOlItemPrototype
2699 }

```

```

2700 \seq_gput_right:Nn
2701 \g_@@_renderers_seq
2702 { fancy01Item }
2703 \prop_gput:Nnn
2704 \g_@@_renderer_arities_prop
2705 { fancy01Item }
2706 { 0 }
2707 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemEnd` 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.

```

2708 \ExplSyntaxOn
2709 \cs_gset_protected:Npn
2710 \markdownRendererFancy01ItemEnd
2711 {
2712 \markdownRendererFancy01ItemEndPrototype
2713 }
2714 \seq_gput_right:Nn
2715 \g_@@_renderers_seq
2716 { fancy01ItemEnd }
2717 \prop_gput:Nnn
2718 \g_@@_renderer_arities_prop
2719 { fancy01ItemEnd }
2720 { 0 }
2721 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemWithNumber` 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.

```

2722 \ExplSyntaxOn
2723 \cs_gset_protected:Npn
2724 \markdownRendererFancy01ItemWithNumber
2725 {
2726 \markdownRendererFancy01ItemWithNumberPrototype
2727 }
2728 \seq_gput_right:Nn
2729 \g_@@_renderers_seq
2730 { fancy01ItemWithNumber }
2731 \prop_gput:Nnn
2732 \g_@@_renderer_arities_prop
2733 { fancy01ItemWithNumber }
2734 { 1 }
2735 \ExplSyntaxOff

```

The `\markdownRendererOlEnd` 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.

```

2736 \ExplSyntaxOn
2737 \cs_gset_protected:Npn
2738 \markdownRendererOlEnd
2739 {
2740 \markdownRendererOlEndPrototype
2741 }
2742 \seq_gput_right:Nn
2743 \g_@@_renderers_seq
2744 { olEnd }
2745 \prop_gput:Nnn
2746 \g_@@_renderer_arities_prop
2747 { olEnd }
2748 { 0 }
2749 \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.

```

2750 \ExplSyntaxOn
2751 \cs_gset_protected:Npn
2752 \markdownRendererOlEndTight
2753 {
2754 \markdownRendererOlEndTightPrototype
2755 }
2756 \seq_gput_right:Nn
2757 \g_@@_renderers_seq
2758 { olEndTight }
2759 \prop_gput:Nnn
2760 \g_@@_renderer_arities_prop
2761 { olEndTight }
2762 { 0 }
2763 \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.

```

2764 \ExplSyntaxOn
2765 \cs_gset_protected:Npn
2766 \markdownRendererFancyOlEnd

```



```

2767 {
2768 \markdownRendererFancyO1EndPrototype
2769 }
2770 \seq_gput_right:Nn
2771 \g_@@_renderers_seq
2772 { fancyO1End }
2773 \prop_gput:Nnn
2774 \g_@@_renderer_arities_prop
2775 { fancyO1End }
2776 { 0 }
2777 \ExplSyntaxOff

```

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

```

2778 \ExplSyntaxOn
2779 \cs_gset_protected:Npn
2780 \markdownRendererFancyO1EndTight
2781 {
2782 \markdownRendererFancyO1EndTightPrototype
2783 }
2784 \seq_gput_right:Nn
2785 \g_@@_renderers_seq
2786 { fancyO1EndTight }
2787 \prop_gput:Nnn
2788 \g_@@_renderer_arities_prop
2789 { fancyO1EndTight }
2790 { 0 }
2791 \ExplSyntaxOff

```

### 2.2.5.31 Raw Content Renderers

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

```

2792 \ExplSyntaxOn
2793 \cs_gset_protected:Npn
2794 \markdownRendererInputRawInline
2795 {
2796 \markdownRendererInputRawInlinePrototype
2797 }
2798 \seq_gput_right:Nn
2799 \g_@@_renderers_seq
2800 { inputRawInline }

```

```

2801 \prop_gput:Nnn
2802 \g_@@_renderer_arities_prop
2803 { inputRawInline }
2804 { 2 }
2805 \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.

```

2806 \ExplSyntaxOn
2807 \cs_gset_protected:Npn
2808 \markdownRendererInputRawBlock
2809 {
2810 \markdownRendererInputRawBlockPrototype
2811 }
2812 \seq_gput_right:Nn
2813 \g_@@_renderers_seq
2814 { inputRawBlock }
2815 \prop_gput:Nnn
2816 \g_@@_renderer_arities_prop
2817 { inputRawBlock }
2818 { 2 }
2819 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2820 \ExplSyntaxOn
2821 \cs_gset_protected:Npn
2822 \markdownRendererSectionBegin
2823 {
2824 \markdownRendererSectionBeginPrototype
2825 }
2826 \seq_gput_right:Nn
2827 \g_@@_renderers_seq
2828 { sectionBegin }
2829 \prop_gput:Nnn
2830 \g_@@_renderer_arities_prop
2831 { sectionBegin }
2832 { 0 }
2833 \cs_gset_protected:Npn
2834 \markdownRendererSectionEnd
2835 {
2836 \markdownRendererSectionEndPrototype
2837 }

```

```

2838 \seq_gput_right:Nn
2839 \g_@@_renderers_seq
2840 { sectionEnd }
2841 \prop_gput:Nnn
2842 \g_@@_renderer_arities_prop
2843 { sectionEnd }
2844 { 0 }
2845 \ExplSyntaxOff

```

### 2.2.5.33 Replacement Character Renderers

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

```

2846 \ExplSyntaxOn
2847 \cs_gset_protected:Npn
2848 \markdownRendererReplacementCharacter
2849 {
2850 \markdownRendererReplacementCharacterPrototype
2851 }
2852 \seq_gput_right:Nn
2853 \g_@@_renderers_seq
2854 { replacementCharacter }
2855 \prop_gput:Nnn
2856 \g_@@_renderer_arities_prop
2857 { replacementCharacter }
2858 { 0 }
2859 \ExplSyntaxOff

```

### 2.2.5.34 Special Character Renderers

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

```

2860 \ExplSyntaxOn
2861 \cs_gset_protected:Npn
2862 \markdownRendererLeftBrace
2863 {
2864 \markdownRendererLeftBracePrototype
2865 }
2866 \seq_gput_right:Nn
2867 \g_@@_renderers_seq
2868 { leftBrace }
2869 \prop_gput:Nnn
2870 \g_@@_renderer_arities_prop
2871 { leftBrace }
2872 { 0 }
2873 \cs_gset_protected:Npn

```

```

2874 \markdownRendererRightBrace
2875 {
2876 \markdownRendererRightBracePrototype
2877 }
2878 \seq_gput_right:Nn
2879 \g_@@_renderers_seq
2880 { rightBrace }
2881 \prop_gput:Nnn
2882 \g_@@_renderer_arities_prop
2883 { rightBrace }
2884 { 0 }
2885 \cs_gset_protected:Npn
2886 \markdownRendererDollarSign
2887 {
2888 \markdownRendererDollarSignPrototype
2889 }
2890 \seq_gput_right:Nn
2891 \g_@@_renderers_seq
2892 { dollarSign }
2893 \prop_gput:Nnn
2894 \g_@@_renderer_arities_prop
2895 { dollarSign }
2896 { 0 }
2897 \cs_gset_protected:Npn
2898 \markdownRendererPercentSign
2899 {
2900 \markdownRendererPercentSignPrototype
2901 }
2902 \seq_gput_right:Nn
2903 \g_@@_renderers_seq
2904 { percentSign }
2905 \prop_gput:Nnn
2906 \g_@@_renderer_arities_prop
2907 { percentSign }
2908 { 0 }
2909 \cs_gset_protected:Npn
2910 \markdownRendererAmpersand
2911 {
2912 \markdownRendererAmpersandPrototype
2913 }
2914 \seq_gput_right:Nn
2915 \g_@@_renderers_seq
2916 { ampersand }
2917 \prop_gput:Nnn
2918 \g_@@_renderer_arities_prop
2919 { ampersand }
2920 { 0 }

```

```

2921 \cs_gset_protected:Npn
2922 \markdownRendererUnderscore
2923 {
2924 \markdownRendererUnderscorePrototype
2925 }
2926 \seq_gput_right:Nn
2927 \g_@@_renderers_seq
2928 { underscore }
2929 \prop_gput:Nnn
2930 \g_@@_renderer_arities_prop
2931 { underscore }
2932 { 0 }
2933 \cs_gset_protected:Npn
2934 \markdownRendererHash
2935 {
2936 \markdownRendererHashPrototype
2937 }
2938 \seq_gput_right:Nn
2939 \g_@@_renderers_seq
2940 { hash }
2941 \prop_gput:Nnn
2942 \g_@@_renderer_arities_prop
2943 { hash }
2944 { 0 }
2945 \cs_gset_protected:Npn
2946 \markdownRendererCircumflex
2947 {
2948 \markdownRendererCircumflexPrototype
2949 }
2950 \seq_gput_right:Nn
2951 \g_@@_renderers_seq
2952 { circumflex }
2953 \prop_gput:Nnn
2954 \g_@@_renderer_arities_prop
2955 { circumflex }
2956 { 0 }
2957 \cs_gset_protected:Npn
2958 \markdownRendererBackslash
2959 {
2960 \markdownRendererBackslashPrototype
2961 }
2962 \seq_gput_right:Nn
2963 \g_@@_renderers_seq
2964 { backslash }
2965 \prop_gput:Nnn
2966 \g_@@_renderer_arities_prop
2967 { backslash }

```

```

2968 { 0 }
2969 \cs_gset_protected:Npn
2970 \markdownRendererTilde
2971 {
2972 \markdownRendererTildePrototype
2973 }
2974 \seq_gput_right:Nn
2975 \g_@@_renderers_seq
2976 { tilde }
2977 \prop_gput:Nnn
2978 \g_@@_renderer_arities_prop
2979 { tilde }
2980 { 0 }
2981 \cs_gset_protected:Npn
2982 \markdownRendererPipe
2983 {
2984 \markdownRendererPipePrototype
2985 }
2986 \seq_gput_right:Nn
2987 \g_@@_renderers_seq
2988 { pipe }
2989 \prop_gput:Nnn
2990 \g_@@_renderer_arities_prop
2991 { pipe }
2992 { 0 }
2993 \ExplSyntaxOff

```

### 2.2.5.35 Strike-Through Renderer

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

```

2994 \ExplSyntaxOn
2995 \cs_gset_protected:Npn
2996 \markdownRendererStrikeThrough
2997 {
2998 \markdownRendererStrikeThroughPrototype
2999 }
3000 \seq_gput_right:Nn
3001 \g_@@_renderers_seq
3002 { strikeThrough }
3003 \prop_gput:Nnn
3004 \g_@@_renderer_arities_prop
3005 { strikeThrough }
3006 { 1 }
3007 \ExplSyntaxOff

```

### 2.2.5.36 Subscript Renderer

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

```
3008 \ExplSyntaxOn
3009 \cs_gset_protected:Npn
3010 \markdownRendererSubscript
3011 {
3012 \markdownRendererSubscriptPrototype
3013 }
3014 \seq_gput_right:Nn
3015 \g_@@_renderers_seq
3016 { subscript }
3017 \prop_gput:Nnn
3018 \g_@@_renderer_arities_prop
3019 { subscript }
3020 { 1 }
```

### 2.2.5.37 Superscript Renderer

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

```
3021 \cs_gset_protected:Npn
3022 \markdownRendererSuperscript
3023 {
3024 \markdownRendererSuperscriptPrototype
3025 }
3026 \seq_gput_right:Nn
3027 \g_@@_renderers_seq
3028 { superscript }
3029 \prop_gput:Nnn
3030 \g_@@_renderer_arities_prop
3031 { superscript }
3032 { 1 }
3033 \ExplSyntaxOff
```

### 2.2.5.38 Table Attribute Context Renderers

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

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

```
3034 \ExplSyntaxOn
```

```

3035 \cs_gset_protected:Npn
3036 \markdownRendererTableAttributeContextBegin
3037 {
3038 \markdownRendererTableAttributeContextBeginPrototype
3039 }
3040 \seq_gput_right:Nn
3041 \g_@@_renderers_seq
3042 { tableAttributeContextBegin }
3043 \prop_gput:Nnn
3044 \g_@@_renderer_arities_prop
3045 { tableAttributeContextBegin }
3046 { 0 }
3047 \cs_gset_protected:Npn
3048 \markdownRendererTableAttributeContextEnd
3049 {
3050 \markdownRendererTableAttributeContextEndPrototype
3051 }
3052 \seq_gput_right:Nn
3053 \g_@@_renderers_seq
3054 { tableAttributeContextEnd }
3055 \prop_gput:Nnn
3056 \g_@@_renderer_arities_prop
3057 { tableAttributeContextEnd }
3058 { 0 }
3059 \ExplSyntaxOff

```

### 2.2.5.39 Table Renderer

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

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

```

3060 \ExplSyntaxOn
3061 \cs_gset_protected:Npn
3062 \markdownRendererTable
3063 {
3064 \markdownRendererTablePrototype
3065 }
3066 \seq_gput_right:Nn

```



```

3067 \g_@@_renderers_seq
3068 { table }
3069 \prop_gput:Nnn
3070 \g_@@_renderer_arities_prop
3071 { table }
3072 { 3 }
3073 \ExplSyntaxOff

```

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

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

```

3074 \ExplSyntaxOn
3075 \cs_gset_protected:Npn
3076 \markdownRendererInlineMath
3077 {
3078 \markdownRendererInlineMathPrototype
3079 }
3080 \seq_gput_right:Nn
3081 \g_@@_renderers_seq
3082 { inlineMath }
3083 \prop_gput:Nnn
3084 \g_@@_renderer_arities_prop
3085 { inlineMath }
3086 { 1 }
3087 \cs_gset_protected:Npn
3088 \markdownRendererDisplayMath
3089 {
3090 \markdownRendererDisplayMathPrototype
3091 }
3092 \seq_gput_right:Nn
3093 \g_@@_renderers_seq
3094 { displayMath }
3095 \prop_gput:Nnn
3096 \g_@@_renderer_arities_prop
3097 { displayMath }
3098 { 1 }
3099 \ExplSyntaxOff

```

#### 2.2.5.41 Thematic Break Renderer

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

```

3100 \ExplSyntaxOn
3101 \cs_gset_protected:Npn
3102 \markdownRendererThematicBreak
3103 {
3104 \markdownRendererThematicBreakPrototype
3105 }
3106 \seq_gput_right:Nn
3107 \g_@@_renderers_seq
3108 { thematicBreak }
3109 \prop_gput:Nnn
3110 \g_@@_renderer_arities_prop
3111 { thematicBreak }
3112 { 0 }
3113 \ExplSyntaxOff

```

#### 2.2.5.42 Tickbox Renderers

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

```

3114 \ExplSyntaxOn
3115 \cs_gset_protected:Npn
3116 \markdownRendererTickedBox
3117 {
3118 \markdownRendererTickedBoxPrototype
3119 }
3120 \seq_gput_right:Nn
3121 \g_@@_renderers_seq
3122 { tickedBox }
3123 \prop_gput:Nnn
3124 \g_@@_renderer_arities_prop
3125 { tickedBox }
3126 { 0 }
3127 \cs_gset_protected:Npn
3128 \markdownRendererHalfTickedBox
3129 {
3130 \markdownRendererHalfTickedBoxPrototype
3131 }
3132 \seq_gput_right:Nn
3133 \g_@@_renderers_seq
3134 { halfTickedBox }
3135 \prop_gput:Nnn
3136 \g_@@_renderer_arities_prop
3137 { halfTickedBox }
3138 { 0 }

```

```

3139 \cs_gset_protected:Npn
3140 \markdownRendererUntickedBox
3141 {
3142 \markdownRendererUntickedBoxPrototype
3143 }
3144 \seq_gput_right:Nn
3145 \g_@@_renderers_seq
3146 { untickedBox }
3147 \prop_gput:Nnn
3148 \g_@@_renderer_arities_prop
3149 { untickedBox }
3150 { 0 }
3151 \ExplSyntaxOff

```

### 2.2.5.43 Warning and Error Renderers

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

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

```

3152 \ExplSyntaxOn
3153 \cs_gset_protected:Npn
3154 \markdownRendererWarning
3155 {
3156 \markdownRendererWarningPrototype
3157 }
3158 \cs_gset_protected:Npn
3159 \markdownRendererError
3160 {
3161 \markdownRendererErrorPrototype
3162 }
3163 \seq_gput_right:Nn
3164 \g_@@_renderers_seq
3165 { warning }
3166 \prop_gput:Nnn
3167 \g_@@_renderer_arities_prop
3168 { warning }
3169 { 4 }

```

```

3170 \seq_gput_right:Nn
3171 \g_@@_renderers_seq
3172 { error }
3173 \prop_gput:Nnn
3174 \g_@@_renderer_arities_prop
3175 { error }
3176 { 4 }
3177 \ExplSyntaxOff

```

#### 2.2.5.44 YAML Metadata Renderers

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

```

3178 \ExplSyntaxOn
3179 \cs_gset_protected:Npn
3180 \markdownRendererJekyllDataBegin
3181 {
3182 \markdownRendererJekyllDataBeginPrototype
3183 }
3184 \seq_gput_right:Nn
3185 \g_@@_renderers_seq
3186 { jekyllDataBegin }
3187 \prop_gput:Nnn
3188 \g_@@_renderer_arities_prop
3189 { jekyllDataBegin }
3190 { 0 }
3191 \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.

```

3192 \ExplSyntaxOn
3193 \cs_gset_protected:Npn
3194 \markdownRendererJekyllDataEnd
3195 {
3196 \markdownRendererJekyllDataEndPrototype
3197 }
3198 \seq_gput_right:Nn
3199 \g_@@_renderers_seq
3200 { jekyllDataEnd }
3201 \prop_gput:Nnn
3202 \g_@@_renderer_arities_prop
3203 { jekyllDataEnd }
3204 { 0 }
3205 \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.

```

3206 \ExplSyntaxOn
3207 \cs_gset_protected:Npn
3208 \markdownRendererJekyllDataMappingBegin
3209 {
3210 \markdownRendererJekyllDataMappingBeginPrototype
3211 }
3212 \seq_gput_right:Nn
3213 \g_@@_renderers_seq
3214 { jekyllDataMappingBegin }
3215 \prop_gput:Nnn
3216 \g_@@_renderer_arities_prop
3217 { jekyllDataMappingBegin }
3218 { 2 }
3219 \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.

```

3220 \ExplSyntaxOn
3221 \cs_gset_protected:Npn
3222 \markdownRendererJekyllDataMappingEnd
3223 {
3224 \markdownRendererJekyllDataMappingEndPrototype
3225 }
3226 \seq_gput_right:Nn
3227 \g_@@_renderers_seq
3228 { jekyllDataMappingEnd }
3229 \prop_gput:Nnn
3230 \g_@@_renderer_arities_prop
3231 { jekyllDataMappingEnd }
3232 { 0 }
3233 \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.

```

3234 \ExplSyntaxOn
3235 \cs_gset_protected:Npn

```

```

3236 \markdownRendererJekyllDataSequenceBegin
3237 {
3238 \markdownRendererJekyllDataSequenceBeginPrototype
3239 }
3240 \seq_gput_right:Nn
3241 \g_@@_renderers_seq
3242 { jekyllDataSequenceBegin }
3243 \prop_gput:Nnn
3244 \g_@@_renderer_arities_prop
3245 { jekyllDataSequenceBegin }
3246 { 2 }
3247 \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.

```

3248 \ExplSyntaxOn
3249 \cs_gset_protected:Npn
3250 \markdownRendererJekyllDataSequenceEnd
3251 {
3252 \markdownRendererJekyllDataSequenceEndPrototype
3253 }
3254 \seq_gput_right:Nn
3255 \g_@@_renderers_seq
3256 { jekyllDataSequenceEnd }
3257 \prop_gput:Nnn
3258 \g_@@_renderer_arities_prop
3259 { jekyllDataSequenceEnd }
3260 { 0 }
3261 \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.

```

3262 \ExplSyntaxOn
3263 \cs_gset_protected:Npn
3264 \markdownRendererJekyllDataBoolean
3265 {
3266 \markdownRendererJekyllDataBooleanPrototype
3267 }
3268 \seq_gput_right:Nn
3269 \g_@@_renderers_seq
3270 { jekyllDataBoolean }
3271 \prop_gput:Nnn

```

```

3272 \g_@@_renderer_arities_prop
3273 { jekyllDataBoolean }
3274 { 2 }
3275 \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.

```

3276 \ExplSyntaxOn
3277 \cs_gset_protected:Npn
3278 \markdownRendererJekyllDataNumber
3279 {
3280 \markdownRendererJekyllDataNumberPrototype
3281 }
3282 \seq_gput_right:Nn
3283 \g_@@_renderers_seq
3284 { jekyllDataNumber }
3285 \prop_gput:Nnn
3286 \g_@@_renderer_arities_prop
3287 { jekyllDataNumber }
3288 { 2 }
3289 \ExplSyntaxOff

```

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

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

```

3290 \ExplSyntaxOn
3291 \cs_gset_protected:Npn
3292 \markdownRendererJekyllDataTypographicString
3293 {
3294 \markdownRendererJekyllDataTypographicStringPrototype
3295 }
3296 \cs_gset_protected:Npn

```

```

3297 \markdownRendererJekyllDataProgrammaticString
3298 {
3299 \markdownRendererJekyllDataProgrammaticStringPrototype
3300 }
3301 \seq_gput_right:Nn
3302 \g_@@_renderers_seq
3303 { jekyllDataTypographicString }
3304 \prop_gput:Nnn
3305 \g_@@_renderer_arities_prop
3306 { jekyllDataTypographicString }
3307 { 2 }
3308 \seq_gput_right:Nn
3309 \g_@@_renderers_seq
3310 { jekyllDataProgrammaticString }
3311 \prop_gput:Nnn
3312 \g_@@_renderer_arities_prop
3313 { jekyllDataProgrammaticString }
3314 { 2 }
3315 \ExplSyntaxOff

```

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

```

3316 \ExplSyntaxOn
3317 \cs_gset:Npn
3318 \markdownRendererJekyllDataTypographicString
3319 {
3320 \cs_if_exist:NTF
3321 \markdownRendererJekyllDataString
3322 {
3323 \@@_if_option:nTF
3324 { experimental }
3325 {
3326 \markdownError
3327 {
3328 The~jekyllDataString~renderer~has~been~deprecated,~
3329 to~be~removed~in~Markdown~4.0.0
3330 }
3331 }
3332 {
3333 \markdownWarning
3334 {
3335 The~jekyllDataString~renderer~has~been~deprecated,~
3336 to~be~removed~in~Markdown~4.0.0
3337 }
3338 }
3339 }
3340 }

```



```

3339 }
3340 }
3341 {
3342 \cs_if_exist:NTF
3343 \markdownRendererJekyllDataStringPrototype
3344 {
3345 \@@_if_option:NTF
3346 { experimental }
3347 {
3348 \markdownError
3349 {
3350 The~jekyllDataString~renderer~prototype~
3351 has~been~deprecated,~
3352 to~be~removed~in~Markdown~4.0.0
3353 }
3354 }
3355 {
3356 \markdownWarning
3357 {
3358 The~jekyllDataString~renderer~prototype~
3359 has~been~deprecated,~
3360 to~be~removed~in~Markdown~4.0.0
3361 }
3362 \markdownRendererJekyllDataStringPrototype
3363 }
3364 }
3365 {
3366 \markdownRendererJekyllDataTypographicStringPrototype
3367 }
3368 }
3369 }
3370 \seq_gput_right:Nn
3371 \g_@@_renderers_seq
3372 { jekyllDataString }
3373 \prop_gput:Nnn
3374 \g_@@_renderer_arities_prop
3375 { jekyllDataString }
3376 { 2 }
3377 \ExplSyntaxOff

```

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

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

```

3378 \ExplSyntaxOn
3379 \cs_gset_protected:Npn
3380 \markdownRendererJekyllDataEmpty
3381 {
3382 \markdownRendererJekyllDataEmptyPrototype
3383 }
3384 \seq_gput_right:Nn
3385 \g_@@_renderers_seq
3386 { jekyllDataEmpty }
3387 \prop_gput:Nnn
3388 \g_@@_renderer_arities_prop
3389 { jekyllDataEmpty }
3390 { 1 }
3391 \ExplSyntaxOff

```

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

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

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

```

3392 \ExplSyntaxOn
3393 \cs_new:Nn \@@_define_renderers:
3394 {
3395 \seq_map_inline:Nn
3396 \g_@@_renderers_seq
3397 {
3398 \@@_define_renderer:n
3399 { ##1 }
3400 }
3401 }
3402 \cs_new:Nn \@@_define_renderer:n
3403 {
3404 \@@_renderer_tl_to_csname:nN
3405 { #1 }
3406 \l_tmpa_tl
3407 \prop_get:NnN
3408 \g_@@_renderer_arities_prop
3409 { #1 }
3410 \l_tmpb_tl
3411 \@@_define_renderer:ncV
3412 { #1 }

```

```

3413 { \l_tmpa_tl }
3414 \l_tmpb_tl
3415 }
3416 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3417 {
3418 \tl_set:Nn
3419 \l_tmpa_tl
3420 { \str_uppercase:n { #1 } }
3421 \tl_set:Nx
3422 #2
3423 {
3424 markdownRenderer
3425 \tl_head:f { \l_tmpa_tl }
3426 \tl_tail:n { #1 }
3427 }
3428 }
3429 \tl_new:N
3430 \l_@@_renderer_definition_tl
3431 \bool_new:N
3432 \g_@@_appending_renderer_bool
3433 \bool_new:N
3434 \g_@@_unprotected_renderer_bool
3435 \cs_new:Nn \@@_define_renderer:nNn
3436 {
3437 \keys_define:nn
3438 { markdown/options/renderers }
3439 {
3440 #1 .code:n = {
3441 \tl_set:Nn
3442 \l_@@_renderer_definition_tl
3443 { ##1 }
3444 \regex_replace_all:nnN
3445 { \cP\#0 }
3446 { #1 }
3447 \l_@@_renderer_definition_tl
3448 \bool_if:NT
3449 \g_@@_appending_renderer_bool
3450 {
3451 \@@_tl_set_from_cs:NNn
3452 \l_tmpa_tl
3453 #2
3454 { #3 }
3455 \tl_put_left:NV
3456 \l_@@_renderer_definition_tl
3457 \l_tmpa_tl
3458 }
3459 \bool_if:NTF

```

```

3460 \g_@@_unprotected_renderer_bool
3461 {
3462 \tl_set:Nn
3463 \l_tmpa_tl
3464 { \cs_set:Npn }
3465 }
3466 {
3467 \tl_set:Nn
3468 \l_tmpa_tl
3469 { \cs_set_protected:Npn }
3470 }
3471 \exp_last_unbraced:NNV
3472 \cs_generate_from_arg_count:NNnV
3473 #2
3474 \l_tmpa_tl
3475 { #3 }
3476 \l_@@_renderer_definition_tl
3477 },
3478 }

```

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

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

```

3479 \str_if_eq:nnT
3480 { #1 }
3481 { jekyllDataString }
3482 {
3483 \cs_undefine:N
3484 #2
3485 }
3486 }

```

We define the function `\@@_tl_set_from_cs:NNn` [13]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```

3487 \cs_new_protected:Nn
3488 \@@_tl_set_from_cs:NNn
3489 {
3490 \tl_set:Nn
3491 \l_tmpa_tl
3492 { #2 }
3493 \int_step_inline:nn
3494 { #3 }
3495 {
3496 \exp_args:NNc
3497 \tl_put_right:Nn

```

```

3498 \l_tmpa_tl
3499 { @@_tl_set_from_cs_parameter_ ##1 }
3500 }
3501 \exp_args:NNV
3502 \tl_set:No
3503 \l_tmpb_tl
3504 \l_tmpa_tl
3505 \regex_replace_all:nnN
3506 { \cP. }
3507 { \0\0 }
3508 \l_tmpb_tl
3509 \int_step_inline:nn
3510 { #3 }
3511 {
3512 \regex_replace_all:nnN
3513 { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3514 { \cP\# ##1 }
3515 \l_tmpb_tl
3516 }
3517 \tl_set:NV
3518 #1
3519 \l_tmpb_tl
3520 }
3521 \cs_generate_variant:Nn
3522 \@@_define_renderer:nNn
3523 { ncV }
3524 \cs_generate_variant:Nn
3525 \cs_generate_from_arg_count:NNnn
3526 { NNnV }
3527 \cs_generate_variant:Nn
3528 \tl_put_left:Nn
3529 { Nv }
3530 \keys_define:nn
3531 { markdown/options }
3532 {
3533 renderers .code:n = {
3534 \bool_gset_false:N
3535 \g_@@_unprotected_renderer_bool
3536 \keys_set:nn
3537 { markdown/options/renderers }
3538 { #1 }
3539 },
3540 unprotectedRenderers .code:n = {
3541 \bool_gset_true:N
3542 \g_@@_unprotected_renderer_bool
3543 \keys_set:nn
3544 { markdown/options/renderers }

```

```

3545 { #1 }
3546 },
3547 }

```

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

```

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

```

```

3548 \tl_new:N
3549 \l_@@_renderer_glob_definition_tl
3550 \seq_new:N
3551 \l_@@_renderer_glob_results_seq
3552 \regex_const:Nn
3553 \c_@@_appending_key_regex
3554 { \s*+$ }
3555 \keys_define:nn
3556 { markdown/options/renderers }
3557 {
3558 unknown .code:n = {

```

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

```

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

```

```

headerAttributeContextBegin += {
 \markdownSetup{
 renderers = {
 attributeIdentifier += {...},
 },
 },
},
}

```

```

3559 % TODO: Use `\\regex_if_match` in TeX Live 2025.
3560 \\regex_match:NVTF
3561 \\c_@@_appending_key_regex
3562 \\l_keys_key_str
3563 {
3564 \\bool_gset_true:N
3565 \\g_@@_appending_renderer_bool
3566 \\tl_set:NV
3567 \\l_tmpa_tl
3568 \\l_keys_key_str
3569 \\regex_replace_once:NnN
3570 \\c_@@_appending_key_regex
3571 { }
3572 \\l_tmpa_tl
3573 \\tl_set:Nx
3574 \\l_tmpb_tl
3575 { { \\l_tmpa_tl } = }
3576 \\tl_put_right:Nn
3577 \\l_tmpb_tl
3578 { { #1 } }
3579 \\keys_set:nV
3580 { markdown/options/renderers }
3581 \\l_tmpb_tl
3582 \\bool_gset_false:N
3583 \\g_@@_appending_renderer_bool
3584 }

```

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

```

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

```

```

 },
 }
}

```

Wildcards and enumerations can be combined:

```

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

```

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

```

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

```

```

3585 {
3586 \@@_glob_seq:VnN
3587 \l_keys_key_str
3588 { g_@@_renderers_seq }
3589 \l_@@_renderer_glob_results_seq
3590 \seq_if_empty:NTF
3591 \l_@@_renderer_glob_results_seq
3592 {
3593 \msg_error:nnV
3594 { markdown }
3595 { undefined-renderer }
3596 \l_keys_key_str
3597 }
3598 {
3599 \tl_set:Nn
3600 \l_@@_renderer_glob_definition_tl
3601 { \exp_not:n { #1 } }
3602 \seq_map_inline:Nn
3603 \l_@@_renderer_glob_results_seq
3604 {
3605 \tl_set:Nn
3606 \l_tmpa_tl
3607 { { ##1 } = }
3608 \tl_put_right:Nx

```



```

3609 \l_tmpa_tl
3610 { { \l_@@_renderer_glob_definition_tl } }
3611 \keys_set:nV
3612 { markdown/options/renderers }
3613 \l_tmpa_tl
3614 }
3615 }
3616 }
3617 },
3618 }
3619 \msg_new:nnn
3620 { markdown }
3621 { undefined-renderer }
3622 {
3623 Renderer~#1~is~undefined.
3624 }
3625 \cs_generate_variant:Nn
3626 \@@_glob_seq:nnN
3627 { VnN }
3628 \cs_generate_variant:Nn
3629 \cs_generate_from_arg_count:NNnn
3630 { cNVV }
3631 \cs_generate_variant:Nn
3632 \msg_error:nnn
3633 { nnV }
3634 \prg_generate_conditional_variant:Nnn
3635 % TODO: Use `regex_if_match` in TeX Live 2025.
3636 \regex_match:Nn % noqa: w202
3637 { NV }
3638 { TF }
3639 \prop_new:N
3640 \g_@@_glob_cache_prop
3641 \tl_new:N
3642 \l_@@_current_glob_tl
3643 \cs_new:Nn
3644 \@@_glob_seq:nnN
3645 {
3646 \tl_set:Nn
3647 \l_@@_current_glob_tl
3648 { ^ #1 $ }
3649 \prop_get:NeNTF
3650 \g_@@_glob_cache_prop
3651 { #2 / \l_@@_current_glob_tl }
3652 \l_tmpa_clist
3653 {
3654 \seq_set_from_clist:NN
3655 #3

```

```

3656 \l_tmpa_clist
3657 }
3658 {
3659 \seq_clear:N
3660 #3
3661 \regex_replace_all:nnN
3662 { * }
3663 { .* }
3664 \l_@@_current_glob_tl
3665 \regex_set:NV
3666 \l_tmpa_regex
3667 \l_@@_current_glob_tl
3668 \seq_map_inline:cn
3669 { #2 }
3670 {
3671 % TODO: Use \regex_if_match in TeX Live 2025.
3672 \regex_match:NnT % noqa: w202
3673 \l_tmpa_regex
3674 { ##1 }
3675 {
3676 \seq_put_right:Nn
3677 #3
3678 { ##1 }
3679 }
3680 }
3681 \clist_set_from_seq:NN
3682 \l_tmpa_clist
3683 #3
3684 \prop_gput:NeV
3685 \g_@@_glob_cache_prop
3686 { #2 / \l_@@_current_glob_tl }
3687 \l_tmpa_clist
3688 }
3689 }
3690 \cs_generate_variant:Nn
3691 \regex_set:Nn
3692 { NV }
3693 \cs_generate_variant:Nn
3694 \prop_gput:Nnn
3695 { NeV }

```

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

```

3696 \str_if_eq:VVT
3697 \c_@@_top_layer_tl
3698 \c_@@_option_layer_plain_tex_tl

```

```

3699 {
3700 \@@_define_renderers:
3701 }
3702 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

For simple YAML metadata, a simple high-level interface is provided that can be programmed by setting the `expl3` key-values [3] for the module `markdown/jekyllData`.

```

3703 \ExplSyntaxOn
3704 \keys_define:nn
3705 { markdown/jekyllData }
3706 { }
3707 \ExplSyntaxOff

```

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

```

3708 \ExplSyntaxOn
3709 \@@_with_various_cases:nn
3710 { jekyllDataRenderers }
3711 {
3712 \keys_define:nn
3713 { markdown/options }
3714 {
3715 #1 .code:n = {
3716 \tl_set:Nn
3717 \l_tmpa_tl
3718 { ##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.

```

3719 \tl_replace_all:NnV
3720 \l_tmpa_tl
3721 { / }
3722 \c_backslash_str
3723 \keys_set:nV
3724 { markdown/options/jekyll-data-renderers }
3725 \l_tmpa_tl
3726 },
3727 }
3728 }
3729 \keys_define:nn
3730 { markdown/options/jekyll-data-renderers }

```

```

3731 {
3732 unknown .code:n = {
3733 \tl_set_eq:NN
3734 \l_tmpa_tl
3735 \l_keys_key_str
3736 \tl_replace_all:NVn
3737 \l_tmpa_tl
3738 \c_backslash_str
3739 { / }
3740 \tl_put_right:Nn
3741 \l_tmpa_tl
3742 {
3743 .code:n = { #1 }
3744 }
3745 \keys_define:nV
3746 { markdown/jekyllData }
3747 \l_tmpa_tl
3748 }
3749 }
3750 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [14] can be used to route the processing of all YAML metadata in the current  $\text{\TeX}$  group to the key-values from  $\langle module \rangle$ .

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

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

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

```

3751 \ExplSyntaxOn
3752 \cs_new:Nn \@@_define_renderer_prototypes:
3753 {
3754 \seq_map_inline:Nn
3755 \g_@@_renderers_seq
3756 {
3757 \@@_define_renderer_prototype:n
3758 { ##1 }

```

```

3759 }
3760 }
3761 \cs_new:Nn \@@_define_renderer_prototype:n
3762 {
3763 \@@_renderer_prototype_tl_to_csname:nN
3764 { #1 }
3765 \l_tmpa_tl
3766 \prop_get:NnN
3767 \g_@@_renderer_arities_prop
3768 { #1 }
3769 \l_tmpb_tl
3770 \@@_define_renderer_prototype:ncV
3771 { #1 }
3772 { \l_tmpa_tl }
3773 \l_tmpb_tl
3774 }
3775 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3776 {
3777 \tl_set:Nn
3778 \l_tmpa_tl
3779 { \str_uppercase:n { #1 } }
3780 \tl_set:Nx
3781 #2
3782 {
3783 markdownRenderer
3784 \tl_head:f { \l_tmpa_tl }
3785 \tl_tail:n { #1 }
3786 Prototype
3787 }
3788 }
3789 \tl_new:N
3790 \l_@@_renderer_prototype_definition_tl
3791 \bool_new:N
3792 \g_@@_appending_renderer_prototype_bool
3793 \bool_new:N
3794 \g_@@_unprotected_renderer_prototype_bool
3795 \cs_new:Nn \@@_define_renderer_prototype:nNn
3796 {
3797 \keys_define:nn
3798 { markdown/options/renderer-prototypes }
3799 {
3800 #1 .code:n = {
3801 \tl_set:Nn
3802 \l_@@_renderer_prototype_definition_tl
3803 { ##1 }
3804 \regex_replace_all:nnN
3805 { \cP\#0 }

```

```

3806 { #1 }
3807 \l_@@_renderer_prototype_definition_tl
3808 \bool_if:NT
3809 \g_@@_appending_renderer_prototype_bool
3810 {
3811 \@@_tl_set_from_cs:NNn
3812 \l_tmpa_tl
3813 #2
3814 { #3 }
3815 \tl_put_left:NV
3816 \l_@@_renderer_prototype_definition_tl
3817 \l_tmpa_tl
3818 }
3819 \bool_if:NTF
3820 \g_@@_unprotected_renderer_prototype_bool
3821 {
3822 \tl_set:Nn
3823 \l_tmpa_tl
3824 { \cs_set:Npn }
3825 }
3826 {
3827 \tl_set:Nn
3828 \l_tmpa_tl
3829 { \cs_set_protected:Npn }
3830 }
3831 \exp_last_unbraced:NNV
3832 \cs_generate_from_arg_count:NNnV
3833 #2
3834 \l_tmpa_tl
3835 { #3 }
3836 \l_@@_renderer_prototype_definition_tl
3837 },
3838 }

```

Unless the token `renderer` prototype macro has already been defined or unless, it has been deprecated, we provide an empty definition.

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

```

3839 \str_if_eq:nnF
3840 { #1 }
3841 { jekyllDataString }
3842 {
3843 \cs_if_free:NT
3844 #2
3845 {
3846 \cs_generate_from_arg_count:NNnn
3847 #2

```

```

3848 \cs_gset_protected:Npn
3849 { #3 }
3850 { }
3851 }
3852 }
3853 }
3854 \cs_generate_variant:Nn
3855 \@@_define_renderer_prototype:nNn
3856 { ncV }

```

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

```

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

```

```

3857 \keys_define:nn
3858 { markdown/options/renderer-prototypes }
3859 {
3860 unknown .code:n = {

```

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

```

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

```

```

\markdownSetup{
 rendererPrototypes = {
 attributeIdentifier += {...},
 },
}
},
},
}
}

```

```

3861 % TODO: Use `\\regex_if_match` in TeX Live 2025.
3862 \\regex_match:NVTF
3863 \\c_@@_appending_key_regex
3864 \\l_keys_key_str
3865 {
3866 \\bool_gset_true:N
3867 \\g_@@_appending_renderer_prototype_bool
3868 \\tl_set:NV
3869 \\l_tmpa_tl
3870 \\l_keys_key_str
3871 \\regex_replace_once:NnN
3872 \\c_@@_appending_key_regex
3873 { }
3874 \\l_tmpa_tl
3875 \\tl_set:Nx
3876 \\l_tmpb_tl
3877 { { \\l_tmpa_tl } = }
3878 \\tl_put_right:Nn
3879 \\l_tmpb_tl
3880 { { #1 } }
3881 \\keys_set:nV
3882 { markdown/options/renderer-prototypes }
3883 \\l_tmpb_tl
3884 \\bool_gset_false:N
3885 \\g_@@_appending_renderer_prototype_bool
3886 }

```

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

```

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

```



```

}
}

```

Wildcards and enumerations can be combined:

```

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

```

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

```

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

```

```

3887 {
3888 \@@_glob_seq:VnN
3889 \l_keys_key_str
3890 { g_@@_renderers_seq }
3891 \l_@@_renderer_glob_results_seq
3892 \seq_if_empty:NTF
3893 \l_@@_renderer_glob_results_seq
3894 {
3895 \msg_error:nnV
3896 { markdown }
3897 { undefined-renderer-prototype }
3898 \l_keys_key_str
3899 }
3900 {
3901 \tl_set:Nn
3902 \l_@@_renderer_glob_definition_tl
3903 { \exp_not:n { #1 } }
3904 \seq_map_inline:Nn
3905 \l_@@_renderer_glob_results_seq
3906 {
3907 \tl_set:Nn
3908 \l_tmpa_tl
3909 { { ##1 } = }
3910 \tl_put_right:Nx

```

```

3911 \l_tmpa_tl
3912 { { \l_@@_renderer_glob_definition_tl } }
3913 \keys_set:nV
3914 { markdown/options/renderer-prototypes }
3915 \l_tmpa_tl
3916 }
3917 }
3918 }
3919 },
3920 }
3921 \msg_new:nnn
3922 { markdown }
3923 { undefined-renderer-prototype }
3924 {
3925 Renderer~prototype~#1~is~undefined.
3926 }
3927 \@@_with_various_cases:nn
3928 { rendererPrototypes }
3929 {
3930 \keys_define:nn
3931 { markdown/options }
3932 {
3933 #1 .code:n = {
3934 \bool_gset_false:N
3935 \g_@@_unprotected_renderer_prototype_bool
3936 \keys_set:nn
3937 { markdown/options/renderer-prototypes }
3938 { ##1 }
3939 },
3940 }
3941 }
3942 \@@_with_various_cases:nn
3943 { unprotectedRendererPrototypes }
3944 {
3945 \keys_define:nn
3946 { markdown/options }
3947 {
3948 #1 .code:n = {
3949 \bool_gset_true:N
3950 \g_@@_unprotected_renderer_prototype_bool
3951 \keys_set:nn
3952 { markdown/options/renderer-prototypes }
3953 { ##1 }
3954 },
3955 }
3956 }

```

If plain T<sub>E</sub>X is the top layer, we use the `\@@_define_renderer_prototypes:` macro

to define plain T<sub>E</sub>X token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3957 \str_if_eq:VVT
3958 \c_@@_top_layer_tl
3959 \c_@@_option_layer_plain_tex_tl
3960 {
3961 \@@_define_renderer_prototypes:
3962 }
3963 \ExplSyntaxOff

```

### 2.2.7 Logging Facilities

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

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

### 2.2.8 Miscellanea

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

```

3964 \let\markdownMakeOther\relax

```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` and `\yamlBegin` macros. The first argument specifies the token sequence that will terminate the markdown input when the plain T<sub>E</sub>X special characters have had their category changed to *other*: `\markdownEnd` for the `\markdownBegin` macro and `\yamlEnd` for the `\yamlBegin` macro. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```

3965 \let\markdownReadAndConvert\relax
3966 \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*.

```

3967 \catcode`\|=0\catcode`\=12%
3968 |gdef|markdownBegin{%
3969 |markdownReadAndConvert{\markdownEnd}%

```

```

3970 {|markdownEnd}}}%
3971 \gdef|yamlBegin{%
3972 \begingroup
3973 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3974 \markdownReadAndConvert{\yamlEnd}%
3975 {|yamlEnd}}}%
3976 \endgroup

```

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

The `code` key, which can be used to immediately expand and execute code.

```

3977 \ExplSyntaxOn
3978 \keys_define:nn
3979 { markdown/options }
3980 {
3981 code .code:n = { #1 },
3982 }
3983 \ExplSyntaxOff

```

This can be especially useful in snippets.

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

To determine whether L<sup>A</sup>T<sub>E</sub>X is the top layer or if there are other layers above L<sup>A</sup>T<sub>E</sub>X, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that L<sup>A</sup>T<sub>E</sub>X is the top layer.

```

3984 \ExplSyntaxOn
3985 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3986 \cs_generate_variant:Nn
3987 \tl_const:Nn
3988 { NV }
3989 \tl_if_exist:NF
3990 \c_@@_top_layer_tl
3991 {
3992 \tl_const:NV
3993 \c_@@_top_layer_tl
3994 \c_@@_option_layer_latex_tl
3995 }
3996 \ExplSyntaxOff
3997 \input markdown/markdown

```

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

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

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

### 2.3.1 Typesetting Markdown

The interface exposes the `markdown`, `markdown*`, and `yaml`  $\text{\LaTeX}$  environments, and redefines the `\markinline`, `\markdownInput`, and `\yamlInput` commands.

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*`  $\text{\LaTeX}$  environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain  $\text{\TeX}$  interface.

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

```
3998 \newenvironment{markdown}\relax\relax
3999 \newenvironment{markdown*}[1]\relax\relax
```

Furthermore, both environments accept  $\text{\LaTeX}$  interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

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

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

<code>\documentclass{article}</code>	<code>\documentclass{article}</code>
<code>\usepackage{markdown}</code>	<code>\usepackage{markdown}</code>
<code>\begin{document}</code>	<code>\begin{document}</code>
<code>\begin{markdown}[smartEllipses]</code>	<code>\begin{markdown*}{smartEllipses}</code>
<code>_Hello_ **world** ...</code>	<code>_Hello_ **world** ...</code>
<code>\end{markdown}</code>	<code>\end{markdown*}</code>
<code>\end{document}</code>	<code>\end{document}</code>

You can't directly extend the `markdown`  $\text{\LaTeX}$  environment by using it in other environments as follows:

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

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

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

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

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown`  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  environment.

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

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

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

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

The `yaml`  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain  $\text{T}_{\text{E}}\text{X}$  interface.

```
4000 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  interface options (see Section 2.3.3) as the only optional argument.

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

The following example  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  code showcases the usage of the `yaml` environment:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
```

```

\begin{yaml}[smartEllipses]
title: _Hello_ world ...
author: John Doe
\end{yaml}
\end{document}

```

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

```

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

```

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

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

### 2.3.1.2 Typesetting Markdown and YAML from external documents

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

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

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

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain  $\text{\TeX}$ . Unlike the `\yamlInput` macro provided by the plain  $\text{\TeX}$  interface, this macro also accepts  $\text{\LaTeX}$  interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

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

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

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

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

### 2.3.2 Using $\text{\LaTeX}$ hooks with the Markdown package

$\text{\LaTeX}$  provides an intricate hook management system that allows users to insert extra material before and after certain  $\text{\TeX}$  macros and  $\text{\LaTeX}$  environments, among other things. [15, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before  $\text{\TeX}$  commands and before/after  $\text{\LaTeX}$  environments without restriction:



```

\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with L<sup>A</sup>T<sub>E</sub>X will produce the text “foo emphasis: \_bar\_ baz!”, as expected.

However, using hooks to insert extra material after T<sub>E</sub>X commands only works for commands with a fixed number of parameters that don’t use currying.

If, in the above example, you explicitly defined the renderer for emphasis using `\markdownSetup` or another method that does not use currying, then you would be able to insert extra material even after the renderer:

```

\documentclass{article}
\usepackage{markdown}
\markdownSetup{renderers={emphasis={\emph{#1}}}}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{<emphasis>}
\AddToHook{cmd/markdownRendererEmphasis/after}{</emphasis>}
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with L<sup>A</sup>T<sub>E</sub>X will produce the text “foo \_bar\_ baz!”, as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.12. In such a case, you would need to redefine the renderer in a way that does not use currying before you would be able to use hooks to insert extra material after it.

Hooks also cannot be used to insert extra material after renderers with a variable number of parameters such as the renderer for tables, see Section 2.2.5.39.

### 2.3.3 Options

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

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

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

#### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [16, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\TeX$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

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

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

```
4001 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
4002 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

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

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

```
4003 \ExplSyntaxOn
4004 \str_if_eq:VVT
4005 \c_@@_top_layer_tl
4006 \c_@@_option_layer_latex_tl
```

```

4007 {
4008 \@@_define_option_commands_and_keyvals:
4009 \@@_define_renderers:
4010 \@@_define_renderer_prototypes:
4011 }
4012 \ExplSyntaxOff

```

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

```

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

```

### 2.3.4 Themes

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

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

If the  $\text{\LaTeX}$  option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown  $\text{\LaTeX}$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, the following code would first load the Markdown package, then the theme `witiko/example/foo`, and finally the theme `witiko/example/bar`:

```

\usepackage[
 import=witiko/example/foo,
 import=witiko/example/bar,
]{markdown}

```

```

4013 \newif\ifmarkdownLaTeXLoaded
4014 \markdownLaTeXLoadedfalse

```

Due to limitations of  $\text{\LaTeX}$ , themes may not be loaded after the beginning of a  $\text{\LaTeX}$  document.

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

```
4015 \ExplSyntaxOn
4016 \prop_new:N
4017 \g_@@_latex_built_in_themes_prop
4018 \ExplSyntaxOff
```

Built-in  $\text{\LaTeX}$  themes provided with the Markdown package include:

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

```
4019 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

At the end of the  $\text{\LaTeX}$  module, we load the **witiko/markdown/defaults**  $\text{\LaTeX}$  theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option **noDefaults** has been enabled (see Section 2.2.2.3).

```
4020 \ExplSyntaxOn
4021 \str_if_eq:VVT
4022 \c_@@_top_layer_tl
4023 \c_@@_option_layer_latex_tl
4024 {
4025 \use:c
4026 { ExplSyntaxOff }
4027 \AtEndOfPackage
4028 {
4029 \@@_if_option:nF
4030 { noDefaults }
4031 {
4032 \@@_if_option:nTF
4033 { experimental }
4034 {
4035 \@@_setup:n
4036 { theme = witiko/markdown/defaults@experimental }
4037 }
4038 {
4039 \@@_setup:n
4040 { theme = witiko/markdown/defaults }
4041 }
4042 }
4043 }
4044 \use:c
4045 { ExplSyntaxOn }
```

```

4046 }
4047 \ExplSyntaxOff

```

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

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

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

```

4048 \ExplSyntaxOn
4049 \tl_const:Nn \c_@@_option_layer_context_tl { context }
4050 \cs_generate_variant:Nn
4051 \tl_const:Nn
4052 { NV }
4053 \tl_if_exist:NF
4054 \c_@@_top_layer_tl
4055 {
4056 \tl_const:NV
4057 \c_@@_top_layer_tl
4058 \c_@@_option_layer_context_tl
4059 }
4060 \ExplSyntaxOff

```

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

```

4061 \writestatus{loading}{ConTEXt User Module / markdown}%
4062 \startmodule[markdown]
4063 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
4064 \do\#\do\^\do_do\%do\~}%
4065 \input markdown/markdown

```

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

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

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

### 2.4.1 Typesetting Markdown and YAML

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

### 2.4.1.1 Typesetting Markdown and YAML directly

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

```
4066 \let\startmarkdown\relax
```

```
4067 \let\stopmarkdown\relax
```

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

The macros `\startmarkdown` and `\stopmarkdown` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

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

The `\startyaml` and `\stopyaml` macros are aliases for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```
4068 \let\startyaml\relax
```

```
4069 \let\stopyaml\relax
```

You may prepend your own code to the `\startyaml` macro and append your own code to the `\stopyaml` macro to produce special effects before and after the YAML document.

The macros `\startyaml` and `\stopyaml` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTeXt code showcases the usage of the `\startyaml` and `\stopyaml` macros:

```
\usemodule[t][markdown]
\starttext
\startyaml
title: _Hello_ **world** ...
author: John Doe
\stopyaml
\stoptext
```

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

```

\usemodule[t][markdown]
\starttext
\setupyaml[jekyllData, expectJekyllData, ensureJekyllData]
\startyaml
title: _Hello_ world ...
author: John Doe
\stopyaml
\stoptext

```

#### 2.4.1.2 Typesetting Markdown and YAML from external documents

The `\inputmarkdown` macro aliases the macro `\markdownInput` exposed by the plain  $\text{\TeX}$  interface.

```
4070 \let\inputmarkdown\relax
```

Furthermore, the `\inputmarkdown` macro also accepts Con $\text{\TeX}$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example Con $\text{\TeX}$ t code showcases the usage of the `\inputmarkdown` macro:

```

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

```

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

```

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

```

The `\inputyaml` macro aliases the macro `\yamlInput` exposed by the plain  $\text{\TeX}$  interface.

```
4071 \let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts Con $\text{\TeX}$ t interface options (see Section 2.4.2) as its optional argument. These options will only influence this YAML document.

The following example Con $\text{\TeX}$ t code showcases the usage of the `\inputyaml` macro:

```

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

```

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

```

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

```

## 2.4.2 Options

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

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

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

```

4072 \ExplSyntaxOn
4073 \cs_new:Npn
4074 \setupmarkdown
4075 [#1]
4076 {
4077 \@@_setup:n
4078 { #1 }
4079 }

```

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

```

4080 \cs_gset_eq:NN
4081 \setupyaml
4082 \setupmarkdown

```

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

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

```

4083 \cs_new:Nn \@@_caseless:N

```



```

4084 {
4085 \regex_replace_all:nnN
4086 { ([a-z])([A-Z]) }
4087 { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
4088 #1
4089 \tl_set:Nx
4090 #1
4091 { #1 }
4092 }
4093 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

4094 \str_if_eq:VVT
4095 \c_@@_top_layer_tl
4096 \c_@@_option_layer_context_tl
4097 {
4098 \@@_define_option_commands_and_keyvals:
4099 \@@_define_renderers:
4100 \@@_define_renderer_prototypes:
4101 }

```

### 2.4.3 Themes

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

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

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

```

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

```

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

```
4102 \prop_new:N
4103 \g_@@_context_built_in_themes_prop
4104 \ExplSyntaxOff
```

Built-in ConT<sub>E</sub>Xt themes provided with the Markdown package include:

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

```
4105 \startmodule[markdownthemewitiko_markdown_defaults]
4106 \unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConT<sub>E</sub>Xt themes.

## 3 Implementation

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

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to T<sub>E</sub>X *token renderers* is performed by the Lua layer. The plain T<sub>E</sub>X layer provides default definitions for the token renderers. The L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt layers correct idiosyncrasies of the respective T<sub>E</sub>X formats, and provide format-specific default definitions for the token renderers.

### 3.1 Lua Implementation

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

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

Furthermore, here are some abbreviations that we inherited from the Lunamark library and which are used throughout the Lua implementation.

```
4107 local upper, format, length =
4108 string.upper, string.format, string.len
4109 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
4110 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
4111 lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

### 3.1.1 Unicode Support

To start off, we load a Lua file named `markdown-unicode-data.lua` that contains our implementation of various Unicode algorithms.

```
4112 local early_warnings = {}
4113 local unicode_data = require("markdown-unicode-data")
4114 if metadata.version ~= unicode_data.metadata.version then
4115 table.insert(
4116 early_warnings,
4117 "markdown.lua " .. metadata.version .. " used with " ..
4118 "markdown-unicode-data.lua " .. unicode_data.metadata.version .. "."
4119)
4120 end
```

In the following subsections, we'll write a second-order file named `markdown-unicode-data-generator.lua`. The code from this file will be executed during the compilation of the Markdown package and the standard output will be stored in a generated file named `markdown-unicode-data.lua`. First, let's define a couple helper functions.

The function `depth_first_search` performs the depth first search algorithm on a tree with nodes being tables with the key `_type` equal to either `intermediate` or `leaf` and with edges labeled with bytes.

Since the algorithm is implemented using recursion, it should only be used for the traversal of shallow trees to prevent exceeding the maximum recursion depth (usually 100).

```
4121 local function depth_first_search(node, path, visit, leave)
4122 assert(node._type == "intermediate")
4123 visit(node, path)
4124 for label, child in pairs(node) do
4125 if label == "_type" then
4126 goto continue
4127 end
4128 if child._type == "intermediate" then
4129 depth_first_search(child, path .. label, visit, leave)
4130 else
4131 assert(child._type == "leaf")
4132 visit(child, path)
4133 end
4134 ::continue::
4135 end
4136 leave(node, path)
4137 end
```

The function `serialize_byte_parser` produces the Lua code for a PEG parser that matches a single byte.

```
4138 local function serialize_byte_parser(byte)
4139 if byte == '"' then
4140 return "P(' " .. byte .. " ')"
4141 end
```

```

4141 elseif byte == "\\" then
4142 return 'P("\\\\")'
4143 else
4144 return 'P("'" .. byte .. "'")'
4145 end
4146 end

```

The function `serialize_byte_range_parser` produces the Lua code for a PEG parser that matches a contiguous range of bytes.

```

4147 local function serialize_byte_range_parser(start_byte, end_byte)
4148 assert(start_byte <= end_byte)
4149 if start_byte == "\\" then
4150 start_byte = "\\\\"
4151 end
4152 if end_byte == "\\" then
4153 end_byte = "\\\\"
4154 end
4155 if start_byte == "'" or end_byte == "'" then
4156 return "R('" .. start_byte .. end_byte .. "'")"
4157 else
4158 return 'R("'" .. start_byte .. end_byte .. "'")'
4159 end
4160 end

```

The function `serialize_replacement` produces the Lua code for a string literal with one or more UTF-8-encoded Unicode characters.

```

4161 local function serialize_replacement(codepoints)
4162 assert(#codepoints > 0)
4163 local buffer = {}
4164 for _, codepoint in ipairs(codepoints) do
4165 local code = utf8.char(codepoint)
4166 for i = 1, #code do
4167 local byte = code:sub(i, i)
4168 if byte == "'" then
4169 table.insert(buffer, '\\\\')
4170 elseif byte == "\\" then
4171 table.insert(buffer, "\\\\")
4172 else
4173 table.insert(buffer, byte)
4174 end
4175 end
4176 end
4177 table.insert(buffer, "'")
4178 return table.concat(buffer)
4179 end

```

The function `read_decompositions` is an iterator that reads a file `UnicodeData.txt` that has previously been opened for reading and yields all canonical and compatibility decompositions from that file.

```

4180 local function read_decompositions(file)
4181 return function()
4182 local from_codepoint, mapping
4183 for line in file:lines() do
4184 from_codepoint, mapping
4185 = line:match("^(%x+);[~;]*;%a*;%d+;%a*;([<%a>%x]*)")
4186 assert(from_codepoint ~= nil)
4187 if #mapping > 0 then
4188 break
4189 end
4190 end
4191 if #mapping == 0 then
4192 return nil
4193 end
4194 from_codepoint = tonumber(from_codepoint, 16)
4195 local to_codepoints, decomposition_type = {}, "canonical"
4196 for raw_codepoint in mapping:gmatch("%S+") do
4197 assert(#raw_codepoint > 0)
4198 if raw_codepoint:sub(1, 1) == "<" then
4199 decomposition_type = "compatibility"
4200 else
4201 local codepoint = tonumber(raw_codepoint, 16)
4202 table.insert(to_codepoints, codepoint)
4203 end
4204 end
4205 assert(#to_codepoints > 0)
4206 return decomposition_type, from_codepoint, to_codepoints
4207 end
4208 end

```

Next, let's define some aliases in the generated file.

```

4209 print("local P, R, Cc, C = lpeg.P, lpeg.R, lpeg.Cc, lpeg.C")
4210 print("local fail = P(false)")
4211 print("-- luacheck: push no max line length")

```

### 3.1.1.1 Canonical and Compatibility Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using either the canonical or the compatibility decomposition [17, Section 3.7] are organized in table `unicode_data.decomposition_mapping` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```

4212 ;(function()
4213 local file = assert(io.open("UnicodeData.txt", "r"),

```

```
4214 [[Could not open file "UnicodeData.txt"]])
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given decomposition type and code length.

```
4215 local decomposition_types = {"canonical", "compatibility"}
4216 local prefix_trees = {}
4217 for _, decomposition_type in ipairs(decomposition_types) do
4218 prefix_trees[decomposition_type] = {}
4219 for char_length = 1, 4 do
4220 prefix_trees[decomposition_type][char_length]
4221 = {_type = "intermediate"}
4222 end
4223 end
4224 for decomposition_type, from_codepoint, to_codepoints
4225 in read_decompositions(file) do
4226 local from_code = utf8.char(from_codepoint)
4227 local node = prefix_trees[decomposition_type][#from_code]
4228 for i = 1, #from_code do
4229 local from_byte = from_code:sub(i, i)
4230 if i < #from_code then
4231 if node[from_byte] == nil then
4232 node[from_byte] = {_type = "intermediate"}
4233 end
4234 node = node[from_byte]
4235 else
4236 table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4237 end
4238 end
4239 end
4240 assert(file:close())
```

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

```
4241 print("M.decomposition_mapping = {}")
4242 for _, decomposition_type in ipairs(decomposition_types) do
4243 print("M.decomposition_mapping." .. decomposition_type .. " = {}")
4244 for length, prefix_tree in pairs(prefix_trees[decomposition_type])
4245 do
4246 local subparsers = {}
4247 depth_first_search(prefix_tree, "", function(node, path)
4248 if node._type == "leaf" then
4249 local from_byte, to_codepoints = table.unpack(node)
4250 local suffix = serialize_byte_parser(from_byte)
4251 .. " / " .. serialize_replacement(to_codepoints)
4252 if subparsers[path] ~= nil then
4253 subparsers[path] = subparsers[path] .. " + " .. suffix
4254 else
4255 subparsers[path] = suffix
```

```

4256 end
4257 end
4258 end, function(_, path)
4259 if #path > 0 then
4260 local byte = path:sub(#path, #path)
4261 local parent_path = path:sub(1, #path-1)
4262 local prefix = serialize_byte_parser(byte)
4263 local suffix
4264 if subparsers[path]:find(" %+ ") then
4265 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4266 else
4267 suffix = prefix .. " * " .. subparsers[path]
4268 end
4269 if subparsers[parent_path] ~= nil then
4270 subparsers[parent_path] = subparsers[parent_path]
4271 .. " + " .. suffix
4272 else
4273 subparsers[parent_path] = suffix
4274 end
4275 else
4276 print(
4277 "M.decomposition_mapping." .. decomposition_type
4278 .. "[" .. length .. "]" = " .. (subparsers[path] or "fail")
4279)
4280 end
4281 end)
4282 end
4283 end
4284 end)()

```

### 3.1.1.2 Hangul Syllable Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using the Hangul syllable decomposition [17, Section 3.12.2] are also organized in table [unicode\\_data.decomposition\\_mapping](#), previously defined in Section 3.1.1.1 based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file [HangulSyllableType.txt](#).

```

4285 ;(function()
4286 local file = assert(io.open("HangulSyllableType.txt", "r"),
4287 [[Could not open file "HangulSyllableType.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given syllable type and code length.

```

4288 local syllable_types = {"LV", "LVT"}
4289 local prefix_trees = {}
4290 for _, syllable_type in ipairs(syllable_types) do
4291 prefix_trees[syllable_type] = {}

```

```

4292 for char_length = 1, 4 do
4293 prefix_trees[syllable_type][char_length]
4294 = {_type = "intermediate"}
4295 end
4296 end
4297 for line in file:lines() do
4298 if #line == 0 or line:sub(1, 1) == "#" then
4299 goto continue
4300 end
4301 local codepoint, syllable_type = line:match("^([%x.]+)%s*;%s*(%a*)")
4302 assert(codepoint ~= nil)
4303 if prefix_trees[syllable_type] == nil then
4304 goto continue
4305 end
4306 local codepoint_start, codepoint_end
4307 if codepoint:find("%.%.") then
4308 codepoint_start, codepoint_end
4309 = codepoint:match("^(%x+)%.%.(%x+)$")
4310 else
4311 codepoint_start, codepoint_end = codepoint, codepoint
4312 end
4313 codepoint_start = tonumber(codepoint_start, 16)
4314 codepoint_end = tonumber(codepoint_end, 16)
4315 local prev_code, prev_leaf_node
4316 for codepoint = codepoint_start, codepoint_end do
4317 local code = utf8.char(codepoint)
4318 local node = prefix_trees[syllable_type][#code]
4319 for i = 1, #code do
4320 local byte = code:sub(i, i)
4321 if i < #code then
4322 if node[byte] == nil then
4323 node[byte] = {_type = "intermediate"}
4324 end
4325 node = node[byte]
4326 else
4327 local leaf_node
4328 if (
4329 prev_code ~= nil
4330 and #prev_code == #code
4331 and code:sub(1, #code - 1)
4332 == prev_code:sub(1, #code - 1)
4333) then
4334 assert(prev_leaf_node ~= nil)
4335 leaf_node = prev_leaf_node
4336 leaf_node[2] = byte
4337 else
4338 leaf_node = {byte, byte, _type = "leaf"}

```



```

4339 table.insert(node, leaf_node)
4340 end
4341 prev_code, prev_leaf_node = code, leaf_node
4342 end
4343 end
4344 end
4345 ::continue::
4346 end
4347 assert(file:close())

```

Next, we will define constants and functions with the Hangul syllable (de)composition algorithm.

```

4348 print(string.format("local s_base = %d", tonumber("AC00", 16)))
4349 print(string.format("local l_base = %d", tonumber("1100", 16)))
4350 print(string.format("local v_base = %d", tonumber("1161", 16)))
4351 print(string.format("local t_base = %d", tonumber("11A7", 16)))
4352 print(string.format("local l_count = %d", 19))
4353 print(string.format("local v_count = %d", 21))
4354 print(string.format("local t_count = %d", 28))
4355 print("local n_count = v_count * t_count")
4356 print("local s_count = l_count * n_count")
4357
4358 print("local function hangul_decompose_LV(byte)")
4359 print(" local s = utf8.codepoint(byte)")
4360 print(" local s_index = s - s_base")
4361 print(" local l_index = s_index // n_count")
4362 print(" local v_index = (s_index % n_count) // t_count")
4363 print(" local l_part = l_base + l_index")
4364 print(" local v_part = v_base + v_index")
4365 print(" return utf8.char(l_part) .. utf8.char(v_part)")
4366 print("end")
4367
4368 print("local function hangul_decompose_LVT(byte)")
4369 print(" local s = utf8.codepoint(byte)")
4370 print(" local s_index = s - s_base")
4371 print(" local lv_index = (s_index // t_count) * t_count")
4372 print(" local t_index = s_index % t_count")
4373 print(" local lv_part = s_base + lv_index")
4374 print(" local t_part = t_base + t_index")
4375 print(" return utf8.char(lv_part) .. utf8.char(t_part)")
4376 print("end")
4377
4378 print("function M.hangul_compose(first_byte, second_byte)")
4379 print(" local last = utf8.codepoint(first_byte)")
4380 print(" local ch = utf8.codepoint(second_byte)")
4381 print(" local l_index = last - l_base")
4382 print(" if 0 <= l_index and l_index < l_count then")

```

```

4383 print(" local v_index = ch - v_base")
4384 print(" if 0 <= v_index and v_index < v_count then")
4385 print(" return utf8.char("")
4386 print(" s_base + (l_index * v_count + v_index) * t_count")
4387 print(" ")
4388 print(" end")
4389 print("end")
4390 print(" local s_index = last - s_base")
4391 print(" if 0 <= s_index and s_index < s_count")
4392 print(" and s_index % t_count == 0 then")
4393 print(" local t_index = ch - t_base")
4394 print(" if 0 < t_index and t_index < t_count then")
4395 print(" return utf8.char(last + t_index)")
4396 print(" end")
4397 print(" end")
4398 print(" return nil")
4399 print("end")

```

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

```

4400 print("M.decomposition_mapping.hangul = {}")
4401 for _, syllable_type in ipairs(syllable_types) do
4402 print("M.decomposition_mapping.hangul." .. syllable_type .. " = {}")
4403 for length, prefix_tree in pairs(prefix_trees[syllable_type]) do
4404 local subparsers = {}
4405 depth_first_search(prefix_tree, "", function(node, path)
4406 if node._type == "leaf" then
4407 local start_byte, end_byte = table.unpack(node)
4408 local suffix
4409 if start_byte == end_byte then
4410 suffix = serialize_byte_parser(start_byte)
4411 else
4412 assert(start_byte < end_byte)
4413 suffix = serialize_byte_range_parser(start_byte, end_byte)
4414 end
4415 if subparsers[path] ~= nil then
4416 subparsers[path] = subparsers[path] .. " + " .. suffix
4417 else
4418 subparsers[path] = suffix
4419 end
4420 end
4421 end, function(_, path)
4422 if #path > 0 then
4423 local byte = path:sub(#path, #path)
4424 local parent_path = path:sub(1, #path-1)
4425 local prefix = serialize_byte_parser(byte)
4426 local suffix
4427 if subparsers[path]:find(" %+ ") then
4428 suffix = prefix .. " * (" .. subparsers[path] .. ")"

```

```

4429 else
4430 suffix = prefix .. " * " .. subparsers[path]
4431 end
4432 if subparsers[parent_path] ~= nil then
4433 subparsers[parent_path] = subparsers[parent_path]
4434 .. " + " .. suffix
4435 else
4436 subparsers[parent_path] = suffix
4437 end
4438 else
4439 if subparsers[path] then
4440 print(
4441 "M.decomposition_mapping.hangul." .. syllable_type
4442 .. "[" .. length .. "] = C(" .. subparsers[path] .. ")"
4443 .. " / hangul_decompose_" .. syllable_type
4444)
4445 else
4446 print(
4447 "M.decomposition_mapping.hangul." .. syllable_type
4448 .. "[" .. length .. "] = fail"
4449)
4450 end
4451 end
4452 end)
4453 end
4454 end
4455 end)()

```

### 3.1.1.3 Canonical Composition

Low-level parsers that map pairs of UTF-8-encoded Unicode characters from a canonical or compatibility decomposition into their primary composites [17, Section 3.11.6] are organized in table [unicode\\_data.composition\\_mapping](#) based on the number of bytes the characters occupy after conversion to UTF-8.

First, let's read the file [DerivedNormalizationProps.txt](#) and record all canonical decomposable characters that are not full composition exclusions.

```

4456 ;(function()
4457 local file = assert(io.open("DerivedNormalizationProps.txt", "r"),
4458 [[Could not open file "DerivedNormalizationProps.txt"]])
4459 local full_composition_exclusions = {}
4460 for line in file:lines() do
4461 if #line == 0 or line:sub(1, 1) == "#" then
4462 goto continue
4463 end
4464 local codepoint, property = line:match("^([%x.]+)%s*;%s*([%a_]+)")
4465 assert(codepoint ~= nil)
4466 if property ~= "Full_Composition_Exclusion" then

```

```

4467 goto continue
4468 end
4469 local codepoint_start, codepoint_end
4470 if codepoint:find("%.%.") then
4471 codepoint_start, codepoint_end
4472 = codepoint:match("^(%x+)%.%.(%x+)$")
4473 else
4474 codepoint_start, codepoint_end = codepoint, codepoint
4475 end
4476 codepoint_start = tonumber(codepoint_start, 16)
4477 codepoint_end = tonumber(codepoint_end, 16)
4478 for codepoint = codepoint_start, codepoint_end do
4479 full_composition_exclusions[codepoint] = true
4480 end
4481 ::continue::
4482 end
4483 assert(file:close())

```

Next, let's also read the file [UnicodeData.txt](#).

```

4484 file = assert(io.open("UnicodeData.txt", "r"),
4485 [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all pairs of codepoints of given code lengths.

```

4486 local prefix_trees = {starters = {}, both = {}}
4487 for first_char_length = 1, 4 do
4488 prefix_trees.starters[first_char_length]
4489 = {_type = "intermediate"}
4490 prefix_trees.both[first_char_length] = {}
4491 for second_char_length = 1, 4 do
4492 prefix_trees.both[first_char_length][second_char_length]
4493 = {_type = "intermediate"}
4494 end
4495 end
4496 local seen_starter_codes = {}
4497 for decomposition_type, to_codepoint, from_codepoints
4498 in read_decompositions(file) do
4499 if (
4500 decomposition_type ~= "canonical"
4501 or #from_codepoints ~= 2
4502 or full_composition_exclusions[to_codepoint]
4503) then
4504 goto continue
4505 end
4506 local starter_code = utf8.char(from_codepoints[1])
4507 local combining_character_code = utf8.char(from_codepoints[2])
4508 local starter_node = prefix_trees.starters[#starter_code]
4509 local both_node

```

```

4510 = prefix_trees.both[#starter_code][#combining_character_code]
4511 for i = 1, #starter_code do
4512 local from_byte = starter_code:sub(i, i)
4513 if both_node[from_byte] == nil then
4514 both_node[from_byte] = {_type = "intermediate"}
4515 end
4516 both_node = both_node[from_byte]
4517 if i < #starter_code then
4518 if starter_node[from_byte] == nil then
4519 starter_node[from_byte] = {_type = "intermediate"}
4520 end
4521 starter_node = starter_node[from_byte]
4522 elseif seen_starter_codes[starter_code] == nil then
4523 seen_starter_codes[starter_code] = true
4524 table.insert(starter_node, {from_byte, _type = "leaf"})
4525 end
4526 end
4527 for i = 1, #combining_character_code do
4528 local from_byte = combining_character_code:sub(i, i)
4529 if i < #combining_character_code then
4530 if both_node[from_byte] == nil then
4531 both_node[from_byte] = {_type = "intermediate"}
4532 end
4533 both_node = both_node[from_byte]
4534 else
4535 table.insert(
4536 both_node,
4537 {from_byte, to_codepoint, _type = "leaf"}
4538)
4539 end
4540 end
4541 ::continue::
4542 end
4543 assert(file:close())

```

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

```

4544 print("M.composition_mapping = {starters = {}, both = {}}")
4545 for first_char_length = 1, 4 do
4546 local prefix_tree = prefix_trees.starters[first_char_length]
4547 local subparsers = {}
4548 depth_first_search(prefix_tree, "", function(node, path)
4549 if node._type == "leaf" then
4550 local from_byte = table.unpack(node)
4551 local suffix = serialize_byte_parser(from_byte)
4552 if subparsers[path] ~= nil then
4553 subparsers[path] = subparsers[path] .. " + " .. suffix
4554 else
4555 subparsers[path] = suffix

```

```

4556 end
4557 end
4558 end, function(_, path)
4559 if #path > 0 then
4560 local byte = path:sub(#path, #path)
4561 local parent_path = path:sub(1, #path-1)
4562 local prefix = serialize_byte_parser(byte)
4563 local suffix
4564 if subparsers[path]:find(" %+ ") then
4565 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4566 else
4567 suffix = prefix .. " * " .. subparsers[path]
4568 end
4569 if subparsers[parent_path] ~= nil then
4570 subparsers[parent_path] = subparsers[parent_path]
4571 .. " + " .. suffix
4572 else
4573 subparsers[parent_path] = suffix
4574 end
4575 else
4576 print(
4577 "M.composition_mapping.starters["
4578 .. first_char_length .. "] = " .. (subparsers[path] or "fail")
4579)
4580 end
4581 end)
4582 print(
4583 string.format(
4584 "M.composition_mapping.both[%d] = {}",
4585 first_char_length
4586)
4587)
4588 for second_char_length = 1, 4 do
4589 prefix_tree
4590 = prefix_trees.both[first_char_length][second_char_length]
4591 subparsers = {}
4592 depth_first_search(prefix_tree, "", function(node, path)
4593 if node._type == "leaf" then
4594 local from_byte, to_codepoint = table.unpack(node)
4595 local suffix = serialize_byte_parser(from_byte)
4596 .. " / " .. serialize_replacement({to_codepoint})
4597 if subparsers[path] ~= nil then
4598 subparsers[path] = subparsers[path] .. " + " .. suffix
4599 else
4600 subparsers[path] = suffix
4601 end
4602 end

```

```

4603 end, function(_, path)
4604 if #path > 0 then
4605 local byte = path:sub(#path, #path)
4606 local parent_path = path:sub(1, #path-1)
4607 local prefix = serialize_byte_parser(byte)
4608 local suffix
4609 if subparsers[path]:find(" %+ ") then
4610 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4611 else
4612 suffix = prefix .. " * " .. subparsers[path]
4613 end
4614 if subparsers[parent_path] ~= nil then
4615 subparsers[parent_path] = subparsers[parent_path]
4616 .. " + " .. suffix
4617 else
4618 subparsers[parent_path] = suffix
4619 end
4620 else
4621 print(
4622 "M.composition_mapping.both[" .. first_char_length .. "]["
4623 .. second_char_length .. "] = "
4624 .. (subparsers[path] or "fail")
4625)
4626 end
4627 end)
4628 end
4629 end
4630 end)()

```

#### 3.1.1.4 Case Folding

Low-level parsers that case-fold UTF-8-encoded Unicode characters using the full mapping (C and F) [17, Section 3.13.3] are organized in table [unicode\\_data.casefold\\_mapping](#) based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file [CaseFolding.txt](#).

```

4631 ;(function()
4632 local file = assert(io.open("CaseFolding.txt", "r"),
4633 [[Could not open file "CaseFolding.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given code length.

```

4634 local prefix_trees = {}
4635 for char_length = 1, 4 do
4636 prefix_trees[char_length] = {_type = "intermediate"}
4637 end
4638 for line in file:lines() do

```

```

4639 if #line == 0 or line:sub(1, 1) == "#" then
4640 goto continue
4641 end
4642 local raw_from_codepoint, status, raw_to_codepoints
4643 = line:match("^(%x+); ([CFST]); ([%x]+);")
4644 assert(raw_from_codepoint ~= nil)
4645 assert(status ~= nil)
4646 assert(raw_to_codepoints ~= nil)
4647 if status ~= "C" and status ~= "F" then
4648 goto continue
4649 end
4650 local from_codepoint = tonumber(raw_from_codepoint, 16)
4651 local to_codepoints = {}
4652 for raw_codepoint in raw_to_codepoints:gmatch('%x+') do
4653 local codepoint = tonumber(raw_codepoint, 16)
4654 table.insert(to_codepoints, codepoint)
4655 end
4656 local from_code = utf8.char(from_codepoint)
4657 local node = prefix_trees[#from_code]
4658 for i = 1, #from_code do
4659 local from_byte = from_code:sub(i, i)
4660 if i < #from_code then
4661 if node[from_byte] == nil then
4662 node[from_byte] = {_type = "intermediate"}
4663 end
4664 node = node[from_byte]
4665 else
4666 table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4667 end
4668 end
4669 ::continue::
4670 end
4671 assert(file:close())

```

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

```

4672 print("M.casefold_mapping = {}")
4673 for length, prefix_tree in pairs(prefix_trees) do
4674 local subparsers = {}
4675 depth_first_search(prefix_tree, "", function(node, path)
4676 if node._type == "leaf" then
4677 local from_byte, to_codepoints = table.unpack(node)
4678 local suffix = serialize_byte_parser(from_byte)
4679 .. " / " .. serialize_replacement(to_codepoints)
4680 if subparsers[path] ~= nil then
4681 subparsers[path] = subparsers[path] .. " + " .. suffix
4682 else
4683 subparsers[path] = suffix
4684 end
4685 end
4686 end)
4687 end

```



```

4685 end
4686 end, function(_, path)
4687 if #path > 0 then
4688 local byte = path:sub(#path, #path)
4689 local parent_path = path:sub(1, #path-1)
4690 local prefix = serialize_byte_parser(byte)
4691 local suffix
4692 if subparsers[path]:find(" %+ ") then
4693 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4694 else
4695 suffix = prefix .. " * " .. subparsers[path]
4696 end
4697 if subparsers[parent_path] ~= nil then
4698 subparsers[parent_path] = subparsers[parent_path]
4699 .. " + " .. suffix
4700 else
4701 subparsers[parent_path] = suffix
4702 end
4703 else
4704 print(
4705 "M.casefold_mapping[" .. length .. "] = "
4706 .. (subparsers[path] or "fail")
4707)
4708 end
4709 end)
4710 end
4711 end)()

```

### 3.1.1.5 Character Categories

Low-level parsers of UTF-8-encoded Unicode characters from different general categories [17, Section 4.5] are organized in table `unicode_data.categories` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```

4712 ;(function()
4713 local file = assert(io.open("UnicodeData.txt", "r"),
4714 [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given Unicode category and code length.

```

4715 local categories = {"L", "N", "P", "Pc", "S", "Z"}
4716 local prefix_trees = {}
4717 for _, category in ipairs(categories) do
4718 prefix_trees[category] = {}
4719 for char_length = 1, 4 do
4720 prefix_trees[category][char_length] = {_type = "intermediate"}

```

```

4721 end
4722 end
4723 for line in file:lines() do
4724 local codepoint, full_category = line:match("^(%x+);[~;]*;(%a*)")
4725 assert(#full_category >= 1)
4726 local major_category = full_category:sub(1, 1)
4727 for _, category in ipairs({full_category, major_category}) do
4728 if prefix_trees[category] == nil then
4729 goto continue
4730 end
4731 local code = utf8.char(tonumber(codepoint, 16))
4732 local node = prefix_trees[category][#code]
4733 for i = 1, #code do
4734 local byte = code:sub(i, i)
4735 if i < #code then
4736 if node[byte] == nil then
4737 node[byte] = {_type = "intermediate"}
4738 end
4739 node = node[byte]
4740 else
4741 table.insert(node, {byte, _type = "leaf"})
4742 end
4743 end
4744 ::continue::
4745 end
4746 end
4747 assert(file:close())

```

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

```

4748 print("M.categories = {}")
4749 for _, category in ipairs(categories) do
4750 print("M.categories." .. category .. " = {}")
4751 for length, prefix_tree in pairs(prefix_trees[category]) do
4752 local subparsers = {}
4753 depth_first_search(prefix_tree, "", function(node, path)
4754 if node._type == "leaf" then
4755 local byte = node[1]
4756 local suffix = serialize_byte_parser(byte)
4757 if subparsers[path] ~= nil then
4758 subparsers[path] = subparsers[path] .. " + " .. suffix
4759 else
4760 subparsers[path] = suffix
4761 end
4762 end
4763 end, function(_, path)
4764 if #path > 0 then
4765 local byte = path:sub(#path, #path)
4766 local parent_path = path:sub(1, #path-1)

```

```

4767 local prefix = serialize_byte_parser(byte)
4768 local suffix
4769 if subparsers[path]:find(" %+ ") then
4770 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4771 else
4772 suffix = prefix .. " * " .. subparsers[path]
4773 end
4774 if subparsers[parent_path] ~= nil then
4775 subparsers[parent_path] = subparsers[parent_path]
4776 .. " + " .. suffix
4777 else
4778 subparsers[parent_path] = suffix
4779 end
4780 else
4781 print(
4782 "M.categories." .. category .. "[" .. length .. "] = "
4783 .. (subparsers[path] or "fail")
4784)
4785 end
4786 end)
4787 end
4788 end
4789 end)()

```

### 3.1.1.6 Canonical Ordering Classes

Low-level parsers of UTF-8-encoded Unicode characters from different character classes [17, Section 3.11] are organized in table `unicode_data.ccc` based on the number of bytes they occupy after conversion to UTF-8.

First, let's read the file `UnicodeData.txt`.

```

4790 ;(function()
4791 local file = assert(io.open("UnicodeData.txt", "r"),
4792 [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct prefix trees of UTF-8 encodings for all codepoints of a given Unicode combining class and code length.

```

4793 local prefix_trees = {}
4794 for char_length = 1, 4 do
4795 prefix_trees[char_length] = { _type = "intermediate" }
4796 end
4797 for line in file:lines() do
4798 local codepoint, combining_class
4799 = line:match("^(%x+);[~;]*;%a*;%d+")
4800 combining_class = tonumber(combining_class)
4801 if combining_class == 0 then
4802 goto continue

```

```

4803 end
4804 local code = utf8.char(tonumber(codepoint, 16))
4805 local node = prefix_trees[#code]
4806 for i = 1, #code do
4807 local byte = code:sub(i, i)
4808 if i < #code then
4809 if node[byte] == nil then
4810 node[byte] = {_type = "intermediate"}
4811 end
4812 node = node[byte]
4813 else
4814 table.insert(node, {byte, combining_class, _type = "leaf"})
4815 end
4816 end
4817 ::continue::
4818 end
4819 assert(file:close())

```

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

```

4820 print("M.ccc = {}")
4821 for length, prefix_tree in pairs(prefix_trees) do
4822 local subparsers = {}
4823 depth_first_search(prefix_tree, "", function(node, path)
4824 if node._type == "leaf" then
4825 local byte, combining_class = table.unpack(node)
4826 local suffix = serialize_byte_parser(byte)
4827 .. " * Cc(" .. tostring(combining_class) .. ")"
4828 if subparsers[path] ~= nil then
4829 subparsers[path] = subparsers[path] .. " + " .. suffix
4830 else
4831 subparsers[path] = suffix
4832 end
4833 end
4834 end, function(_, path)
4835 if #path > 0 then
4836 local byte = path:sub(#path, #path)
4837 local parent_path = path:sub(1, #path-1)
4838 local prefix = serialize_byte_parser(byte)
4839 local suffix
4840 if subparsers[path]:find(" %+ ") then
4841 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4842 else
4843 suffix = prefix .. " * " .. subparsers[path]
4844 end
4845 if subparsers[parent_path] ~= nil then
4846 subparsers[parent_path] = subparsers[parent_path]
4847 .. " + " .. suffix
4848 else

```

```

4849 subparsers[parent_path] = suffix
4850 end
4851 else
4852 print(
4853 "M.ccc[" .. length .. "] = " .. (subparsers[path] or "fail")
4854)
4855 end
4856 end)
4857 end
4858 end)()
4859 print("-- luacheck: pop")
4860 print("return M")

```

### 3.1.2 Utility Functions

This section documents the utility functions back in the file `markdown-parser.lua` used by the plain  $\text{\TeX}$  writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```

4861 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.

```

4862 function util.err(msg, exit_code)
4863 io.stderr:write("markdown.lua: " .. msg .. "\n")
4864 os.exit(exit_code or 1)
4865 end

```

The `util.cache` method used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exist, it gets created with `transform(string)` as its content and the result of `transform(string)` is returned as the second return value in case it's useful to the caller. Regardless, the pathname is always returned as the first return value.

```

4866 function util.cache(dir, string, salt, transform, suffix)
4867 local digest = md5.sumhexa(string .. (salt or ""))
4868 local name = util.pathname(dir, digest .. suffix)
4869 local file = io.open(name, "r")
4870 local result = nil
4871 if file == nil then -- If no cache entry exists, create a new one.
4872 file = assert(io.open(name, "w"),
4873 [[Could not open file]] .. name .. [[for writing]])
4874 result = string
4875 if transform ~= nil then
4876 result = transform(result)
4877 end
4878 assert(file:write(result))

```

```

4879 assert(file:close())
4880 end
4881 return name, result
4882 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.

```

4883 function util.cache_verbatim(dir, string)
4884 local name = util.cache(dir, string, nil, nil, ".verbatim")
4885 return name
4886 end

```

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

```

4887 function util.table_copy(t)
4888 local u = { }
4889 for k, v in pairs(t) do u[k] = v end
4890 return setmetatable(u, getmetatable(t))
4891 end

```

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

```

4892 function util.encode_json_string(s)
4893 s = s:gsub([[\\]], [[\\]])
4894 s = s:gsub([[\"]], [[\"]])
4895 return [[\"]] .. s .. [[\"]]
4896 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 Ierusalimsky [18, Chapter 21].

```

4897 function util.expand_tabs_in_line(s, tabstop)
4898 local tab = tabstop or 4
4899 local corr = 0
4900 return (s:gsub("(\\t)", function(p)
4901 local sp = tab - (p - 1 + corr) % tab
4902 corr = corr - 1 + sp
4903 return string.rep(" ", sp)
4904 end))
4905 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.

```

4906 function util.walk(t, f)
4907 local typ = type(t)
4908 if typ == "string" then

```

```

4909 f(t)
4910 elseif typ == "table" then
4911 local i = 1
4912 local n
4913 n = t[i]
4914 while n do
4915 util.walk(n, f)
4916 i = i + 1
4917 n = t[i]
4918 end
4919 elseif typ == "function" then
4920 local ok, val = pcall(t)
4921 if ok then
4922 util.walk(val, f)
4923 end
4924 else
4925 f(tostring(t))
4926 end
4927 end

```

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

```

4928 function util.flatten(ary)
4929 local new = {}
4930 for _,v in ipairs(ary) do
4931 if type(v) == "table" then
4932 for _,w in ipairs(util.flatten(v)) do
4933 new[#new + 1] = w
4934 end
4935 else
4936 new[#new + 1] = v
4937 end
4938 end
4939 return new
4940 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.

```

4941 function util.rope_to_string(rope)
4942 local buffer = {}
4943 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4944 return table.concat(buffer)
4945 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.

```

4946 function util.rope_last(rope)
4947 if #rope == 0 then

```

```

4948 return nil
4949 else
4950 local l = rope[#rope]
4951 if type(l) == "table" then
4952 return util.rope_last(l)
4953 else
4954 return l
4955 end
4956 end
4957 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  $1 \leq i \leq \#ary$ .

```

4958 function util.intersperse(ary, x)
4959 local new = {}
4960 local l = #ary
4961 for i,v in ipairs(ary) do
4962 local n = #new
4963 new[n + 1] = v
4964 if i ~= l then
4965 new[n + 2] = x
4966 end
4967 end
4968 return new
4969 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  $1 \leq i \leq \#ary$ .

```

4970 function util.map(ary, f)
4971 local new = {}
4972 for i,v in ipairs(ary) do
4973 new[i] = f(v)
4974 end
4975 return new
4976 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.

```

4977 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

4978 local char_escapes_list = ""
4979 for i,_ in pairs(char_escapes) do
4980 char_escapes_list = char_escapes_list .. i

```



```
4981 end
```

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

```
4982 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.

```
4983 if string_escapes then
4984 for k,v in pairs(string_escapes) do
4985 escapable = P(k) / v + escapable
4986 end
4987 end
```

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

```
4988 local escape_string = Cs((escapable + any)^0)
```

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

```
4989 return function(s)
4990 return lpeg.match(escape_string, s)
4991 end
4992 end
```

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

```
4993 function util.pathname(dir, file)
4994 if #dir == 0 then
4995 return file
4996 else
4997 return dir .. "/" .. file
4998 end
4999 end
```

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

```
5000 function util.salt(options)
5001 local opt_string = {}
5002 for k, _ in pairs(defaultOptions) do
5003 local v = options[k]
5004 if type(v) == "table" then
5005 for _, i in ipairs(v) do
```

```

5006 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
5007 end

```

The `cacheDir` option is disregarded.

```

5008 elseif k ~= "cacheDir" then
5009 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5010 end
5011 end
5012 table.sort(opt_string)
5013 local salt = table.concat(opt_string, ",")
5014 .. "," .. metadata.version
5015 return salt
5016 end

```

The `util.warning` method produces a warning `s` that is unrelated to any specific markdown text being processed. For warnings that are specific to a markdown text, use `writer->warning` function.

```

5017 util.warning = (function()
5018 local function warning(s)
5019 io.stderr:write("Warning: " .. s .. "\n")
5020 end

5021 for _, message in ipairs(early_warnings) do
5022 warning(message)
5023 end

5024 return warning
5025 end)()

```

The `util.casefold` method performs a full case-folding of a UTF-8-encoded Unicode string `s` based on the low-level parsers in `unicode_data.casefold_mapping`, defined in Section 3.1.1.4. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5026 util.casefold = (function()
5027 local fail, any = P(false), P(1)
5028 local eof = -any

```

First, define a parser that will case-fold a character.

```

5029 local fold_character = fail
5030 for n = 1, 4 do
5031 fold_character
5032 = fold_character
5033 + unicode_data.casefold_mapping[n]
5034 end
5035 fold_character
5036 = fold_character
5037 + C(any)

```

Next, define a parser that will case-fold a string.

```

5038 local fold_string = Ct(fold_character~0) * eof
5039 return function(s, form)
5040 local result = table.concat(lpeg.match(fold_string, s))
5041 assert(result ~= nil)

```

For NFD and NFKD normalization forms, normalize the case-folded string and then repeat the fold-and-normalize operation.

```

5042 if form == "nfd" or form == "nfkd" then
5043 result = util.normalize(result, form)
5044 result = table.concat(lpeg.match(fold_string, result))
5045 assert(result ~= nil)
5046 result = util.normalize(result, form)
5047 end
5048 return result
5049 end
5050 end)()

```

The `util.canonically_order` method performs a Unicode canonical ordering of a string UTF-8-encoded Unicode `s` based on the low-level parsers in `unicode_data.ccc`, defined in Section 3.1.1.6. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5051 util.canonically_order = (function()
5052 local fail, any = P(false), P(1)
5053 local eof = -any
5054 local cont = R("\128\191")
5055 local utf8_character
5056 = R("\0\127")
5057 + R("\194\223") * cont
5058 + R("\224\239") * cont * cont
5059 + R("\240\244") * cont * cont * cont

```

First, define a parser that will determine the combining class of a character.

```

5060 local classify_character = fail
5061 for n = 1, 4 do
5062 classify_character
5063 = classify_character
5064 + unicode_data.ccc[n]
5065 end
5066 classify_character
5067 = classify_character
5068 + utf8_character * Cc(0)

```

Next, define a parser that will determine the combining classes of all characters in a string.

```

5069 local classify_string = Ct(classify_character~0) * eof

```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```

5070 return function(s)
5071 local s_len = utf8.len(s)
5072 if s == false or s_len <= 1 then
5073 return s
5074 end

```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```

5075 local classes = lpeg.match(classify_string, s)
5076 if classes == nil then
5077 return s
5078 end
5079 assert(#classes == s_len)

```

Again, check whether the string is trivially ordered. If it is, return it without any changes. Otherwise, construct a list of ranges of non-starter characters that must be ordered.

```

5080 local non_starter_ranges = {}
5081 local first_non_starter, last_non_starter = nil, nil
5082 for i = 1, #classes do
5083 if first_non_starter == nil then
5084 if classes[i] ~= 0 then
5085 first_non_starter, last_non_starter = i, i
5086 end
5087 else
5088 if classes[i] == 0 then
5089 table.insert(
5090 non_starter_ranges,
5091 {first_non_starter, last_non_starter}
5092)
5093 first_non_starter, last_non_starter = nil, nil
5094 else
5095 last_non_starter = i
5096 end
5097 end
5098 end
5099 if first_non_starter ~= nil then
5100 table.insert(
5101 non_starter_ranges,
5102 {first_non_starter, last_non_starter}
5103)
5104 end
5105 if #non_starter_ranges == 0 then
5106 return s
5107 end
5108 local max_range_length = 0
5109 for _, range in ipairs(non_starter_ranges) do
5110 local range_start, range_end = table.unpack(range)

```

```

5111 local range_length = range_end - range_start + 1
5112 if range_length > max_range_length then
5113 max_range_length = range_length
5114 end
5115 end
5116 if max_range_length <= 1 then
5117 return s
5118 end

```

Then, construct a buffer of all characters in the string.

```

5119 local buffer = {}
5120 for _, code in utf8.codes(s) do
5121 local char = utf8.char(code)
5122 table.insert(buffer, char)
5123 end
5124 assert(#buffer == s_len)

```

Next, perform a local bubble sort over the ranges of non-starter characters.

```

5125 for _, range in ipairs(non_starter_ranges) do
5126 local range_start, range_end = table.unpack(range)
5127 local range_length = range_end - range_start + 1
5128 for _ = 1, range_length - 1 do
5129 local swapped = false
5130 for i = range_start, range_end - 1 do
5131 local j = i + 1
5132 if classes[i] > classes[j] then
5133 classes[i], classes[j] = classes[j], classes[i]
5134 buffer[i], buffer[j] = buffer[j], buffer[i]
5135 swapped = true
5136 end
5137 end
5138 if not swapped then
5139 break
5140 end
5141 end
5142 end

```

Finally, concatenate the buffer and return an ordered string.

```

5143 return table.concat(buffer, "")
5144 end
5145 end)()

```

The [util.decompose](#) method performs either the canonical or the compatibility decomposition of a UTF-8-encoded Unicode string `s` based on the low-level parsers in [unicode\\_data.decomposition\\_mapping](#), defined in sections 3.1.1.1 and 3.1.1.2. Unlike the low-level parsers, the high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5146 util.decompose = (function()

```

```

5147 local fail, any = P(false), P(1)
5148 local eof = -any
5149 local decomposition_types = {"canonical", "compatibility"}

```

First, define parsers that will decompose a character.

```

5150 local decompose_character = {}
5151 for _, decomposition_type in ipairs(decomposition_types) do
5152 decompose_character[decomposition_type] = fail
5153 for n = 1, 4 do
5154 decompose_character[decomposition_type]
5155 = decompose_character[decomposition_type]
5156 + unicode_data.decomposition_mapping[decomposition_type][n]
5157 end
5158 decompose_character[decomposition_type]
5159 = decompose_character[decomposition_type]
5160 + C(any)
5161 end
5162 local hangul = unicode_data.decomposition_mapping.hangul
5163 decompose_character.hangul = {}
5164 for syllable_type, _ in pairs(hangul) do
5165 decompose_character.hangul[syllable_type] = fail
5166 for n = 1, 4 do
5167 decompose_character.hangul[syllable_type]
5168 = decompose_character.hangul[syllable_type]
5169 + hangul[syllable_type][n]
5170 end
5171 decompose_character.hangul[syllable_type]
5172 = decompose_character.hangul[syllable_type]
5173 + C(any)
5174 end

```

Next, define a parser that will decompose a string.

```

5175 local decompose_string = {}
5176 for _, decomposition_type in ipairs(decomposition_types) do
5177 decompose_string[decomposition_type]
5178 = Ct(decompose_character[decomposition_type]^0) * eof
5179 end
5180 decompose_string.hangul = {}
5181 for syllable_type, _ in pairs(hangul) do
5182 decompose_string.hangul[syllable_type]
5183 = Ct(decompose_character.hangul[syllable_type]^0) * eof
5184 end
5185 local function _decompose(s, parser)
5186 assert(s ~= nil)
5187 local result = table.concat(lpeg.match(parser, s), "")
5188 assert(result ~= nil)
5189 return result
5190 end

```

```

5191 return function(s, decomposition_type)
5192 assert(
5193 decomposition_type == "canonical"
5194 or decomposition_type == "compatibility"
5195)
5196 local prev_s
5197 local next_s = s
5198 repeat
5199 prev_s = next_s
5200 local function decompose(...) next_s = _decompose(next_s, ...) end
5201 decompose(decompose_string.canonical)
5202 if decomposition_type == "compatibility" then
5203 decompose(decompose_string.compatibility)
5204 end
5205 decompose(decompose_string.hangul.LVT)
5206 decompose(decompose_string.hangul.LV)
5207 until prev_s == next_s
5208 return util.canonically_order(next_s)
5209 end
5210 end)()

```

The `util.compose` method performs the canonical composition of a UTF-8-encoded canonically ordered Unicode string `s` based on the low-level parsers in `unicode_data.composition_mapping`, defined in Section 3.1.1.3, and definitions from the Hangul syllable (de)composition algorithm, defined in Section 3.1.1.2. Unlike the low-level parsers, this high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5211 util.compose = (function()
5212 local fail, any = P(false), P(1)
5213 local eof = -any
5214 local cont = R("\128\191")
5215 local utf8_character
5216 = R("\0\127")
5217 + R("\194\223") * cont
5218 + R("\224\239") * cont * cont
5219 + R("\240\244") * cont * cont * cont

```

First, define a parser that will determine the combining class of a character.

```

5220 local classify_character = fail
5221 for n = 1, 4 do
5222 classify_character
5223 = classify_character
5224 + unicode_data.ccc[n]
5225 end
5226 classify_character
5227 = classify_character
5228 + utf8_character * Cc(0)

```

Next, define a parser that will determine the combining classes of all characters in a string.

```
5229 local classify_string = Ct(classify_character^0) * eof
```

First, define parsers that will compose a pair of UTF-8-encoded Unicode characters into their primary composite.

```
5230 local compose_characters = fail
5231 for m = 1, 4 do
5232 local starter = #unicode_data.composition_mapping.starters[m]
5233 local both = fail
5234 for n = 1, 4 do
5235 both = (
5236 both
5237 + #unicode_data.composition_mapping.both[m][n]
5238 * unicode_data.composition_mapping.both[m][n]
5239)
5240 end
5241 compose_characters = compose_characters + starter * both
5242 end
```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```
5243 return function(s)
5244 local s_len = utf8.len(s)
5245 if s == false or s_len <= 1 then
5246 return s
5247 end
```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```
5248 local classes = lpeg.match(classify_string, s)
5249 if classes == nil then
5250 return s
5251 end
5252 assert(#classes == s_len)
```

Otherwise, construct a buffer of all characters in the string.

```
5253 local buffer = {}
5254 for _, code in utf8.codes(s) do
5255 local char = utf8.char(code)
5256 table.insert(buffer, char)
5257 end
5258 assert(#buffer == s_len)
```

Finally, implement the composition algorithm.

First, find the first starter character in the string.

```
5259 local starter = 1
5260 while starter <= s_len and classes[starter] ~= 0 do
```



```

5261 starter = starter + 1
5262 end
5263 local candidate_combining_mark = starter + 1

```

Next, apply the composition rules until we reach the end of the string.

```

5264 while candidate_combining_mark <= s_len do
5265 local L = buffer[starter]
5266 local C = buffer[candidate_combining_mark]
5267 local P = lpeg.match(compose_characters, L .. C)
5268 if P ~= nil then
5269 buffer[starter] = P
5270 buffer[candidate_combining_mark] = ""
5271 else
5272 if classes[candidate_combining_mark] == 0 then
5273 starter = candidate_combining_mark
5274 end
5275 candidate_combining_mark = candidate_combining_mark + 1
5276 end
5277 assert(starter <= s_len)
5278 end

```

Next, iterate over the string once more and compose Hangul syllables.

```

5279 for i = 1, s_len - 1 do
5280 local last, ch = buffer[i], buffer[i + 1]
5281 if last ~= "" and ch ~= "" then
5282 local composite = unicode_data.hangul_compose(last, ch)
5283 if composite ~= nil then
5284 buffer[i] = ""
5285 buffer[i + 1] = composite
5286 end
5287 end
5288 end

```

Finally, concatenate the buffer and return an ordered string.

```

5289 return table.concat(buffer, "")
5290 end
5291 end)()

```

The `util.normalize` method normalizes a UTF-8-encoded canonically ordered Unicode string `s` using the normalization form `form`.

```

5292 function util.normalize(s, form)
5293 if form == "nfd" then
5294 return util.decompose(s, "canonical")
5295 elseif form == "nfkd" then
5296 return util.decompose(s, "compatibility")
5297 elseif form == "nfc" then
5298 return util.compose(util.decompose(s, "canonical"))
5299 elseif form == "nfkc" then
5300 return util.compose(util.decompose(s, "compatibility"))

```

```

5301 else
5302 error(string.format('Unexpected normal form "%s"', form))
5303 end
5304 end

```

The `util.find_file` and `util.find_files` method find the first or all locations of a file, respectively, according to either the resolvers API [1, Section 11.5] from the ConTeXt format or the Kpathsea library.

```

5305 function util.find_file(filename)
5306 if resolvers ~= nil then
5307 return resolvers.findfile(filename)
5308 else
5309 return kpse.find_file(filename)
5310 end
5311 end
5312 function util.find_files(filename)
5313 if resolvers ~= nil then
5314 return resolvers.findfiles(filename)
5315 else
5316 return {kpse.lookup(filename, {all=true})}
5317 end
5318 end

```

### 3.1.3 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.

```

5319 local entities = {}
5320
5321 local character_entities = {
5322 ["Tab"] = 9,
5323 ["NewLine"] = 10,
5324 ["excl"] = 33,
5325 ["QUOT"] = 34,
5326 ["quot"] = 34,
5327 ["num"] = 35,
5328 ["dollar"] = 36,
5329 ["percent"] = 37,
5330 ["AMP"] = 38,
5331 ["amp"] = 38,
5332 ["apos"] = 39,
5333 ["lpar"] = 40,
5334 ["rpar"] = 41,
5335 ["ast"] = 42,
5336 ["midast"] = 42,
5337 ["plus"] = 43,

```

```

5338 ["comma"] = 44,
5339 ["period"] = 46,
5340 ["sol"] = 47,
5341 ["colon"] = 58,
5342 ["semi"] = 59,
5343 ["LT"] = 60,
5344 ["lt"] = 60,
5345 ["nvlt"] = {60, 8402},
5346 ["bne"] = {61, 8421},
5347 ["equals"] = 61,
5348 ["GT"] = 62,
5349 ["gt"] = 62,
5350 ["nvgt"] = {62, 8402},
5351 ["quest"] = 63,
5352 ["commat"] = 64,
5353 ["lbrack"] = 91,
5354 ["lsqb"] = 91,
5355 ["bsol"] = 92,
5356 ["rbrack"] = 93,
5357 ["rsqb"] = 93,
5358 ["Hat"] = 94,
5359 ["UnderBar"] = 95,
5360 ["lowbar"] = 95,
5361 ["DiacriticalGrave"] = 96,
5362 ["grave"] = 96,
5363 ["fjlig"] = {102, 106},
5364 ["lbrace"] = 123,
5365 ["lcub"] = 123,
5366 ["VerticalLine"] = 124,
5367 ["verbar"] = 124,
5368 ["vert"] = 124,
5369 ["rbrace"] = 125,
5370 ["rcub"] = 125,
5371 ["NonBreakingSpace"] = 160,
5372 ["nbsp"] = 160,
5373 ["iexcl"] = 161,
5374 ["cent"] = 162,
5375 ["pound"] = 163,
5376 ["curren"] = 164,
5377 ["yen"] = 165,
5378 ["brvbar"] = 166,
5379 ["sect"] = 167,
5380 ["Dot"] = 168,
5381 ["DoubleDot"] = 168,
5382 ["die"] = 168,
5383 ["uml"] = 168,
5384 ["COPY"] = 169,

```

```

5385 ["copy"] = 169,
5386 ["ordf"] = 170,
5387 ["laquo"] = 171,
5388 ["not"] = 172,
5389 ["shy"] = 173,
5390 ["REG"] = 174,
5391 ["circledR"] = 174,
5392 ["reg"] = 174,
5393 ["macr"] = 175,
5394 ["strns"] = 175,
5395 ["deg"] = 176,
5396 ["PlusMinus"] = 177,
5397 ["plusmn"] = 177,
5398 ["pm"] = 177,
5399 ["sup2"] = 178,
5400 ["sup3"] = 179,
5401 ["DiacriticalAcute"] = 180,
5402 ["acute"] = 180,
5403 ["micro"] = 181,
5404 ["para"] = 182,
5405 ["CenterDot"] = 183,
5406 ["centerdot"] = 183,
5407 ["middot"] = 183,
5408 ["Cedilla"] = 184,
5409 ["cedil"] = 184,
5410 ["sup1"] = 185,
5411 ["ordm"] = 186,
5412 ["raquo"] = 187,
5413 ["frac14"] = 188,
5414 ["frac12"] = 189,
5415 ["half"] = 189,
5416 ["frac34"] = 190,
5417 ["iquest"] = 191,
5418 ["Agrave"] = 192,
5419 ["Aacute"] = 193,
5420 ["Acirc"] = 194,
5421 ["Atilde"] = 195,
5422 ["Auml"] = 196,
5423 ["Aring"] = 197,
5424 ["angst"] = 197,
5425 ["AElig"] = 198,
5426 ["Ccedil"] = 199,
5427 ["Egrave"] = 200,
5428 ["Eacute"] = 201,
5429 ["Ecirc"] = 202,
5430 ["Euml"] = 203,
5431 ["Igrave"] = 204,

```

5432 ["Iacute"] = 205,  
 5433 ["Icirc"] = 206,  
 5434 ["Iuml"] = 207,  
 5435 ["ETH"] = 208,  
 5436 ["Ntilde"] = 209,  
 5437 ["Ograve"] = 210,  
 5438 ["Oacute"] = 211,  
 5439 ["Ocirc"] = 212,  
 5440 ["Otilde"] = 213,  
 5441 ["Ouml"] = 214,  
 5442 ["times"] = 215,  
 5443 ["Oslash"] = 216,  
 5444 ["Ugrave"] = 217,  
 5445 ["Uacute"] = 218,  
 5446 ["Ucirc"] = 219,  
 5447 ["Uuml"] = 220,  
 5448 ["Yacute"] = 221,  
 5449 ["THORN"] = 222,  
 5450 ["szlig"] = 223,  
 5451 ["agrave"] = 224,  
 5452 ["aacute"] = 225,  
 5453 ["acirc"] = 226,  
 5454 ["atilde"] = 227,  
 5455 ["auml"] = 228,  
 5456 ["aring"] = 229,  
 5457 ["aelig"] = 230,  
 5458 ["ccedil"] = 231,  
 5459 ["egrave"] = 232,  
 5460 ["eacute"] = 233,  
 5461 ["ecirc"] = 234,  
 5462 ["euml"] = 235,  
 5463 ["igrave"] = 236,  
 5464 ["iacute"] = 237,  
 5465 ["icirc"] = 238,  
 5466 ["iuml"] = 239,  
 5467 ["eth"] = 240,  
 5468 ["ntilde"] = 241,  
 5469 ["ograve"] = 242,  
 5470 ["oacute"] = 243,  
 5471 ["ocirc"] = 244,  
 5472 ["otilde"] = 245,  
 5473 ["ouml"] = 246,  
 5474 ["div"] = 247,  
 5475 ["divide"] = 247,  
 5476 ["oslash"] = 248,  
 5477 ["ugrave"] = 249,  
 5478 ["uacute"] = 250,

```

5479 ["ucirc"] = 251,
5480 ["uuml"] = 252,
5481 ["yacute"] = 253,
5482 ["thorn"] = 254,
5483 ["yuml"] = 255,
5484 ["Amacr"] = 256,
5485 ["amacr"] = 257,
5486 ["Abreve"] = 258,
5487 ["abreve"] = 259,
5488 ["Aogon"] = 260,
5489 ["aogon"] = 261,
5490 ["Cacute"] = 262,
5491 ["cacute"] = 263,
5492 ["Ccirc"] = 264,
5493 ["ccirc"] = 265,
5494 ["Cdot"] = 266,
5495 ["cdot"] = 267,
5496 ["Ccaron"] = 268,
5497 ["ccaron"] = 269,
5498 ["Dcaron"] = 270,
5499 ["dcaron"] = 271,
5500 ["Dstrok"] = 272,
5501 ["dstrok"] = 273,
5502 ["Emacr"] = 274,
5503 ["emacr"] = 275,
5504 ["Edot"] = 278,
5505 ["edot"] = 279,
5506 ["Eogon"] = 280,
5507 ["eogon"] = 281,
5508 ["Ecaron"] = 282,
5509 ["ecaron"] = 283,
5510 ["Gcirc"] = 284,
5511 ["gcirc"] = 285,
5512 ["Gbreve"] = 286,
5513 ["gbreve"] = 287,
5514 ["Gdot"] = 288,
5515 ["gdot"] = 289,
5516 ["Gcedil"] = 290,
5517 ["Hcirc"] = 292,
5518 ["hcirc"] = 293,
5519 ["Hstrok"] = 294,
5520 ["hstrok"] = 295,
5521 ["Itilde"] = 296,
5522 ["itilde"] = 297,
5523 ["Imacr"] = 298,
5524 ["imacr"] = 299,
5525 ["Iogon"] = 302,

```

```

5526 ["iogon"] = 303,
5527 ["Idot"] = 304,
5528 ["imath"] = 305,
5529 ["inodot"] = 305,
5530 ["IJlig"] = 306,
5531 ["ijlig"] = 307,
5532 ["Jcirc"] = 308,
5533 ["jcirc"] = 309,
5534 ["Kcedil"] = 310,
5535 ["kcedil"] = 311,
5536 ["kgreen"] = 312,
5537 ["Lacute"] = 313,
5538 ["lacute"] = 314,
5539 ["Lcedil"] = 315,
5540 ["lcedil"] = 316,
5541 ["Lcaron"] = 317,
5542 ["lcaron"] = 318,
5543 ["Lmidot"] = 319,
5544 ["lmidot"] = 320,
5545 ["Lstrok"] = 321,
5546 ["lstrok"] = 322,
5547 ["Nacute"] = 323,
5548 ["nacute"] = 324,
5549 ["Ncedil"] = 325,
5550 ["ncedil"] = 326,
5551 ["Ncaron"] = 327,
5552 ["ncaron"] = 328,
5553 ["napos"] = 329,
5554 ["ENG"] = 330,
5555 ["eng"] = 331,
5556 ["Omacr"] = 332,
5557 ["omacr"] = 333,
5558 ["Odblac"] = 336,
5559 ["odblac"] = 337,
5560 ["OElig"] = 338,
5561 ["oelig"] = 339,
5562 ["Racute"] = 340,
5563 ["racute"] = 341,
5564 ["Rcedil"] = 342,
5565 ["rcedil"] = 343,
5566 ["Rcaron"] = 344,
5567 ["rcaron"] = 345,
5568 ["Sacute"] = 346,
5569 ["sacute"] = 347,
5570 ["Scirc"] = 348,
5571 ["scirc"] = 349,
5572 ["Scedil"] = 350,

```

```

5573 ["scedil"] = 351,
5574 ["Scaron"] = 352,
5575 ["scaron"] = 353,
5576 ["Tcedil"] = 354,
5577 ["tcedil"] = 355,
5578 ["Tcaron"] = 356,
5579 ["tcaron"] = 357,
5580 ["Tstrok"] = 358,
5581 ["tstrok"] = 359,
5582 ["Utilde"] = 360,
5583 ["utilde"] = 361,
5584 ["Umacr"] = 362,
5585 ["umacr"] = 363,
5586 ["Ubreve"] = 364,
5587 ["ubreve"] = 365,
5588 ["Uring"] = 366,
5589 ["uring"] = 367,
5590 ["Udblac"] = 368,
5591 ["udblac"] = 369,
5592 ["Uogon"] = 370,
5593 ["uogon"] = 371,
5594 ["Wcirc"] = 372,
5595 ["wcirc"] = 373,
5596 ["Ycirc"] = 374,
5597 ["ycirc"] = 375,
5598 ["Yuml"] = 376,
5599 ["Zacute"] = 377,
5600 ["zacute"] = 378,
5601 ["Zdot"] = 379,
5602 ["zdot"] = 380,
5603 ["Zcaron"] = 381,
5604 ["zcaron"] = 382,
5605 ["fnof"] = 402,
5606 ["imped"] = 437,
5607 ["gacute"] = 501,
5608 ["jmath"] = 567,
5609 ["circ"] = 710,
5610 ["Hacek"] = 711,
5611 ["caron"] = 711,
5612 ["Breve"] = 728,
5613 ["breve"] = 728,
5614 ["DiacriticalDot"] = 729,
5615 ["dot"] = 729,
5616 ["ring"] = 730,
5617 ["ogon"] = 731,
5618 ["DiacriticalTilde"] = 732,
5619 ["tilde"] = 732,

```



```

5620 ["DiacriticalDoubleAcute"] = 733,
5621 ["dblac"] = 733,
5622 ["DownBreve"] = 785,
5623 ["Alpha"] = 913,
5624 ["Beta"] = 914,
5625 ["Gamma"] = 915,
5626 ["Delta"] = 916,
5627 ["Epsilon"] = 917,
5628 ["Zeta"] = 918,
5629 ["Eta"] = 919,
5630 ["Theta"] = 920,
5631 ["Iota"] = 921,
5632 ["Kappa"] = 922,
5633 ["Lambda"] = 923,
5634 ["Mu"] = 924,
5635 ["Nu"] = 925,
5636 ["Xi"] = 926,
5637 ["Omicron"] = 927,
5638 ["Pi"] = 928,
5639 ["Rho"] = 929,
5640 ["Sigma"] = 931,
5641 ["Tau"] = 932,
5642 ["Upsilon"] = 933,
5643 ["Phi"] = 934,
5644 ["Chi"] = 935,
5645 ["Psi"] = 936,
5646 ["Omega"] = 937,
5647 ["ohm"] = 937,
5648 ["alpha"] = 945,
5649 ["beta"] = 946,
5650 ["gamma"] = 947,
5651 ["delta"] = 948,
5652 ["epsi"] = 949,
5653 ["epsilon"] = 949,
5654 ["zeta"] = 950,
5655 ["eta"] = 951,
5656 ["theta"] = 952,
5657 ["iota"] = 953,
5658 ["kappa"] = 954,
5659 ["lambda"] = 955,
5660 ["mu"] = 956,
5661 ["nu"] = 957,
5662 ["xi"] = 958,
5663 ["omicron"] = 959,
5664 ["pi"] = 960,
5665 ["rho"] = 961,
5666 ["sigmaf"] = 962,

```

```

5667 ["sigmav"] = 962,
5668 ["varsigma"] = 962,
5669 ["sigma"] = 963,
5670 ["tau"] = 964,
5671 ["upsi"] = 965,
5672 ["upsilon"] = 965,
5673 ["phi"] = 966,
5674 ["chi"] = 967,
5675 ["psi"] = 968,
5676 ["omega"] = 969,
5677 ["thetasym"] = 977,
5678 ["thetav"] = 977,
5679 ["vartheta"] = 977,
5680 ["Upsi"] = 978,
5681 ["upsih"] = 978,
5682 ["phiv"] = 981,
5683 ["straightphi"] = 981,
5684 ["varphi"] = 981,
5685 ["piv"] = 982,
5686 ["varpi"] = 982,
5687 ["Gammad"] = 988,
5688 ["digamma"] = 989,
5689 ["gammad"] = 989,
5690 ["kappav"] = 1008,
5691 ["varkappa"] = 1008,
5692 ["rhov"] = 1009,
5693 ["varrho"] = 1009,
5694 ["epsiv"] = 1013,
5695 ["straightepsilon"] = 1013,
5696 ["varepsilon"] = 1013,
5697 ["backepsilon"] = 1014,
5698 ["bepsi"] = 1014,
5699 ["IOcy"] = 1025,
5700 ["DJcy"] = 1026,
5701 ["GJcy"] = 1027,
5702 ["Jukcy"] = 1028,
5703 ["DScy"] = 1029,
5704 ["Iukcy"] = 1030,
5705 ["YIcy"] = 1031,
5706 ["Jsercy"] = 1032,
5707 ["LJcy"] = 1033,
5708 ["NJcy"] = 1034,
5709 ["TSHcy"] = 1035,
5710 ["KJcy"] = 1036,
5711 ["Ubrcy"] = 1038,
5712 ["DZcy"] = 1039,
5713 ["Acy"] = 1040,

```

```

5714 ["Bcy"] = 1041,
5715 ["Vcy"] = 1042,
5716 ["Gcy"] = 1043,
5717 ["Dcy"] = 1044,
5718 ["IEcy"] = 1045,
5719 ["ZHcy"] = 1046,
5720 ["Zcy"] = 1047,
5721 ["Icy"] = 1048,
5722 ["Jcy"] = 1049,
5723 ["Kcy"] = 1050,
5724 ["Lcy"] = 1051,
5725 ["Mcy"] = 1052,
5726 ["Ncy"] = 1053,
5727 ["Ocy"] = 1054,
5728 ["Pcy"] = 1055,
5729 ["Rcy"] = 1056,
5730 ["Scy"] = 1057,
5731 ["Tcy"] = 1058,
5732 ["Ucy"] = 1059,
5733 ["Fcy"] = 1060,
5734 ["KHcy"] = 1061,
5735 ["TScy"] = 1062,
5736 ["CHcy"] = 1063,
5737 ["SHcy"] = 1064,
5738 ["SHCHcy"] = 1065,
5739 ["HARDcy"] = 1066,
5740 ["Ycy"] = 1067,
5741 ["SOFTcy"] = 1068,
5742 ["Ecy"] = 1069,
5743 ["YUcy"] = 1070,
5744 ["YAcy"] = 1071,
5745 ["acy"] = 1072,
5746 ["bcy"] = 1073,
5747 ["vcy"] = 1074,
5748 ["gcy"] = 1075,
5749 ["dcy"] = 1076,
5750 ["iecy"] = 1077,
5751 ["zhcy"] = 1078,
5752 ["zcy"] = 1079,
5753 ["icy"] = 1080,
5754 ["jcy"] = 1081,
5755 ["kcy"] = 1082,
5756 ["lcy"] = 1083,
5757 ["mcy"] = 1084,
5758 ["ncy"] = 1085,
5759 ["ocy"] = 1086,
5760 ["pcy"] = 1087,

```

```

5761 ["rcy"] = 1088,
5762 ["scy"] = 1089,
5763 ["tcy"] = 1090,
5764 ["ucy"] = 1091,
5765 ["fcy"] = 1092,
5766 ["khcy"] = 1093,
5767 ["tscy"] = 1094,
5768 ["chcy"] = 1095,
5769 ["shcy"] = 1096,
5770 ["shchcy"] = 1097,
5771 ["hardcy"] = 1098,
5772 ["ycy"] = 1099,
5773 ["softcy"] = 1100,
5774 ["ecy"] = 1101,
5775 ["yucy"] = 1102,
5776 ["yacy"] = 1103,
5777 ["iocy"] = 1105,
5778 ["djcy"] = 1106,
5779 ["gjcy"] = 1107,
5780 ["jukcy"] = 1108,
5781 ["dscy"] = 1109,
5782 ["iukcy"] = 1110,
5783 ["yicy"] = 1111,
5784 ["jsercy"] = 1112,
5785 ["ljcy"] = 1113,
5786 ["njcy"] = 1114,
5787 ["tshcy"] = 1115,
5788 ["kjcy"] = 1116,
5789 ["ubrscy"] = 1118,
5790 ["dzcy"] = 1119,
5791 ["ensp"] = 8194,
5792 ["emsp"] = 8195,
5793 ["emsp13"] = 8196,
5794 ["emsp14"] = 8197,
5795 ["numsp"] = 8199,
5796 ["puncsp"] = 8200,
5797 ["ThinSpace"] = 8201,
5798 ["thinsp"] = 8201,
5799 ["VeryThinSpace"] = 8202,
5800 ["hairsp"] = 8202,
5801 ["NegativeMediumSpace"] = 8203,
5802 ["NegativeThickSpace"] = 8203,
5803 ["NegativeThinSpace"] = 8203,
5804 ["NegativeVeryThinSpace"] = 8203,
5805 ["ZeroWidthSpace"] = 8203,
5806 ["zwnj"] = 8204,
5807 ["zwj"] = 8205,

```

```

5808 ["lrm"] = 8206,
5809 ["rlm"] = 8207,
5810 ["dash"] = 8208,
5811 ["hyphen"] = 8208,
5812 ["ndash"] = 8211,
5813 ["mdash"] = 8212,
5814 ["horbar"] = 8213,
5815 ["Verbar"] = 8214,
5816 ["Vert"] = 8214,
5817 ["OpenCurlyQuote"] = 8216,
5818 ["lsquo"] = 8216,
5819 ["CloseCurlyQuote"] = 8217,
5820 ["rsquo"] = 8217,
5821 ["rsquor"] = 8217,
5822 ["lsquor"] = 8218,
5823 ["sbquo"] = 8218,
5824 ["OpenCurlyDoubleQuote"] = 8220,
5825 ["ldquo"] = 8220,
5826 ["CloseCurlyDoubleQuote"] = 8221,
5827 ["rdquo"] = 8221,
5828 ["rdquor"] = 8221,
5829 ["bdquo"] = 8222,
5830 ["ldquor"] = 8222,
5831 ["dagger"] = 8224,
5832 ["Dagger"] = 8225,
5833 ["ddagger"] = 8225,
5834 ["bull"] = 8226,
5835 ["bullet"] = 8226,
5836 ["nldr"] = 8229,
5837 ["hellip"] = 8230,
5838 ["mldr"] = 8230,
5839 ["permil"] = 8240,
5840 ["pertenk"] = 8241,
5841 ["prime"] = 8242,
5842 ["Prime"] = 8243,
5843 ["tprime"] = 8244,
5844 ["backprime"] = 8245,
5845 ["bprime"] = 8245,
5846 ["lsaquo"] = 8249,
5847 ["rsaquo"] = 8250,
5848 ["OverBar"] = 8254,
5849 ["oline"] = 8254,
5850 ["caret"] = 8257,
5851 ["hybull"] = 8259,
5852 ["frasl"] = 8260,
5853 ["bsemi"] = 8271,
5854 ["qprime"] = 8279,

```

```

5855 ["MediumSpace"] = 8287,
5856 ["ThickSpace"] = {8287, 8202},
5857 ["NoBreak"] = 8288,
5858 ["ApplyFunction"] = 8289,
5859 ["af"] = 8289,
5860 ["InvisibleTimes"] = 8290,
5861 ["it"] = 8290,
5862 ["InvisibleComma"] = 8291,
5863 ["ic"] = 8291,
5864 ["euro"] = 8364,
5865 ["TripleDot"] = 8411,
5866 ["tdot"] = 8411,
5867 ["DotDot"] = 8412,
5868 ["Copf"] = 8450,
5869 ["complexes"] = 8450,
5870 ["incare"] = 8453,
5871 ["gscr"] = 8458,
5872 ["HilbertSpace"] = 8459,
5873 ["Hscr"] = 8459,
5874 ["hamilt"] = 8459,
5875 ["Hfr"] = 8460,
5876 ["Poincareplane"] = 8460,
5877 ["Hopf"] = 8461,
5878 ["quaternions"] = 8461,
5879 ["planckh"] = 8462,
5880 ["hbar"] = 8463,
5881 ["hslash"] = 8463,
5882 ["planck"] = 8463,
5883 ["plankv"] = 8463,
5884 ["Iscr"] = 8464,
5885 ["imagline"] = 8464,
5886 ["Ifr"] = 8465,
5887 ["Im"] = 8465,
5888 ["image"] = 8465,
5889 ["imagpart"] = 8465,
5890 ["Laplacetrif"] = 8466,
5891 ["Lscr"] = 8466,
5892 ["lagran"] = 8466,
5893 ["ell"] = 8467,
5894 ["Nopf"] = 8469,
5895 ["naturals"] = 8469,
5896 ["numero"] = 8470,
5897 ["copysr"] = 8471,
5898 ["weierp"] = 8472,
5899 ["wp"] = 8472,
5900 ["Popf"] = 8473,
5901 ["primes"] = 8473,

```

```

5902 ["Qopf"] = 8474,
5903 ["rationals"] = 8474,
5904 ["Rscr"] = 8475,
5905 ["realine"] = 8475,
5906 ["Re"] = 8476,
5907 ["Rfr"] = 8476,
5908 ["real"] = 8476,
5909 ["realpart"] = 8476,
5910 ["Ropf"] = 8477,
5911 ["reals"] = 8477,
5912 ["rx"] = 8478,
5913 ["TRADE"] = 8482,
5914 ["trade"] = 8482,
5915 ["Zopf"] = 8484,
5916 ["integers"] = 8484,
5917 ["mho"] = 8487,
5918 ["Zfr"] = 8488,
5919 ["zeetrif"] = 8488,
5920 ["iiota"] = 8489,
5921 ["Bernoullis"] = 8492,
5922 ["Bscr"] = 8492,
5923 ["bernou"] = 8492,
5924 ["Cayleys"] = 8493,
5925 ["Cfr"] = 8493,
5926 ["escr"] = 8495,
5927 ["Escr"] = 8496,
5928 ["expectation"] = 8496,
5929 ["Fouriertrf"] = 8497,
5930 ["Fscr"] = 8497,
5931 ["Mellintrf"] = 8499,
5932 ["Mscr"] = 8499,
5933 ["phmmat"] = 8499,
5934 ["order"] = 8500,
5935 ["orderof"] = 8500,
5936 ["oscr"] = 8500,
5937 ["alefsym"] = 8501,
5938 ["aleph"] = 8501,
5939 ["beth"] = 8502,
5940 ["gimel"] = 8503,
5941 ["daleth"] = 8504,
5942 ["CapitalDifferentialD"] = 8517,
5943 ["DD"] = 8517,
5944 ["DifferentialD"] = 8518,
5945 ["dd"] = 8518,
5946 ["ExponentialE"] = 8519,
5947 ["ee"] = 8519,
5948 ["exponentiale"] = 8519,

```

```

5949 ["ImaginaryI"] = 8520,
5950 ["ii"] = 8520,
5951 ["frac13"] = 8531,
5952 ["frac23"] = 8532,
5953 ["frac15"] = 8533,
5954 ["frac25"] = 8534,
5955 ["frac35"] = 8535,
5956 ["frac45"] = 8536,
5957 ["frac16"] = 8537,
5958 ["frac56"] = 8538,
5959 ["frac18"] = 8539,
5960 ["frac38"] = 8540,
5961 ["frac58"] = 8541,
5962 ["frac78"] = 8542,
5963 ["LeftArrow"] = 8592,
5964 ["ShortLeftArrow"] = 8592,
5965 ["larr"] = 8592,
5966 ["leftarrow"] = 8592,
5967 ["slarr"] = 8592,
5968 ["ShortUpArrow"] = 8593,
5969 ["UpArrow"] = 8593,
5970 ["uarr"] = 8593,
5971 ["uparrow"] = 8593,
5972 ["RightArrow"] = 8594,
5973 ["ShortRightArrow"] = 8594,
5974 ["rarr"] = 8594,
5975 ["rightarrow"] = 8594,
5976 ["srarr"] = 8594,
5977 ["DownArrow"] = 8595,
5978 ["ShortDownArrow"] = 8595,
5979 ["darr"] = 8595,
5980 ["downarrow"] = 8595,
5981 ["LeftRightArrow"] = 8596,
5982 ["harr"] = 8596,
5983 ["leftrightarrow"] = 8596,
5984 ["UpDownArrow"] = 8597,
5985 ["updownarrow"] = 8597,
5986 ["varr"] = 8597,
5987 ["UpperLeftArrow"] = 8598,
5988 ["nwarr"] = 8598,
5989 ["nwarrow"] = 8598,
5990 ["UpperRightArrow"] = 8599,
5991 ["nearr"] = 8599,
5992 ["nearrow"] = 8599,
5993 ["LowerRightArrow"] = 8600,
5994 ["searr"] = 8600,
5995 ["searrow"] = 8600,

```



```

5996 ["LowerLeftArrow"] = 8601,
5997 ["swarr"] = 8601,
5998 ["swarrow"] = 8601,
5999 ["nlarr"] = 8602,
6000 ["nleftarrow"] = 8602,
6001 ["nrarr"] = 8603,
6002 ["nrightarrow"] = 8603,
6003 ["nrarrw"] = {8605, 824},
6004 ["rarrw"] = 8605,
6005 ["rightsquigarrow"] = 8605,
6006 ["Larr"] = 8606,
6007 ["twoheadleftarrow"] = 8606,
6008 ["Uarr"] = 8607,
6009 ["Rarr"] = 8608,
6010 ["twoheadrightarrow"] = 8608,
6011 ["Darr"] = 8609,
6012 ["larrtl"] = 8610,
6013 ["leftarrowtail"] = 8610,
6014 ["rarrtl"] = 8611,
6015 ["rightarrowtail"] = 8611,
6016 ["LeftTeeArrow"] = 8612,
6017 ["mapstoleft"] = 8612,
6018 ["UpTeeArrow"] = 8613,
6019 ["mapstoup"] = 8613,
6020 ["RightTeeArrow"] = 8614,
6021 ["map"] = 8614,
6022 ["mapsto"] = 8614,
6023 ["DownTeeArrow"] = 8615,
6024 ["mapstodown"] = 8615,
6025 ["hookleftarrow"] = 8617,
6026 ["larrhk"] = 8617,
6027 ["hookrightarrow"] = 8618,
6028 ["rarrhk"] = 8618,
6029 ["larrlp"] = 8619,
6030 ["looparrowleft"] = 8619,
6031 ["looparrowright"] = 8620,
6032 ["rarrlp"] = 8620,
6033 ["harrw"] = 8621,
6034 ["leftrightsquigarrow"] = 8621,
6035 ["nharr"] = 8622,
6036 ["nletrightarrow"] = 8622,
6037 ["Lsh"] = 8624,
6038 ["lsh"] = 8624,
6039 ["Rsh"] = 8625,
6040 ["rsh"] = 8625,
6041 ["ldsh"] = 8626,
6042 ["rdsh"] = 8627,

```

```

6043 ["crarr"] = 8629,
6044 ["cularr"] = 8630,
6045 ["curvearrowleft"] = 8630,
6046 ["curarr"] = 8631,
6047 ["curvearrowright"] = 8631,
6048 ["circlearrowleft"] = 8634,
6049 ["olarr"] = 8634,
6050 ["circlearrowright"] = 8635,
6051 ["orarr"] = 8635,
6052 ["LeftVector"] = 8636,
6053 ["leftharpoonup"] = 8636,
6054 ["lharu"] = 8636,
6055 ["DownLeftVector"] = 8637,
6056 ["leftharpoondown"] = 8637,
6057 ["lhard"] = 8637,
6058 ["RightUpVector"] = 8638,
6059 ["uharr"] = 8638,
6060 ["upharpoonright"] = 8638,
6061 ["LeftUpVector"] = 8639,
6062 ["uharl"] = 8639,
6063 ["upharpoonleft"] = 8639,
6064 ["RightVector"] = 8640,
6065 ["rharu"] = 8640,
6066 ["rightharpoonup"] = 8640,
6067 ["DownRightVector"] = 8641,
6068 ["rhard"] = 8641,
6069 ["rightharpoondown"] = 8641,
6070 ["RightDownVector"] = 8642,
6071 ["dharr"] = 8642,
6072 ["downharpoonright"] = 8642,
6073 ["LeftDownVector"] = 8643,
6074 ["dharl"] = 8643,
6075 ["downharpoonleft"] = 8643,
6076 ["RightArrowLeftArrow"] = 8644,
6077 ["rightleftarrows"] = 8644,
6078 ["rlarr"] = 8644,
6079 ["UpArrowDownArrow"] = 8645,
6080 ["udarr"] = 8645,
6081 ["LeftArrowRightArrow"] = 8646,
6082 ["leftrightarrows"] = 8646,
6083 ["lrarr"] = 8646,
6084 ["leftleftarrows"] = 8647,
6085 ["llarr"] = 8647,
6086 ["upuparrows"] = 8648,
6087 ["uuarr"] = 8648,
6088 ["rightrightarrows"] = 8649,
6089 ["rrarr"] = 8649,

```

```

6090 ["ddarr"] = 8650,
6091 ["downdownarrows"] = 8650,
6092 ["ReverseEquilibrium"] = 8651,
6093 ["leftrightharpoons"] = 8651,
6094 ["lrhar"] = 8651,
6095 ["Equilibrium"] = 8652,
6096 ["rightleftharpoons"] = 8652,
6097 ["rlhar"] = 8652,
6098 ["nLeftarrow"] = 8653,
6099 ["nlArr"] = 8653,
6100 ["nLeftrightarrow"] = 8654,
6101 ["nhArr"] = 8654,
6102 ["nRrightarrow"] = 8655,
6103 ["nrArr"] = 8655,
6104 ["DoubleLeftArrow"] = 8656,
6105 ["Leftarrow"] = 8656,
6106 ["lArr"] = 8656,
6107 ["DoubleUpArrow"] = 8657,
6108 ["Uparrow"] = 8657,
6109 ["uArr"] = 8657,
6110 ["DoubleRightArrow"] = 8658,
6111 ["Implies"] = 8658,
6112 ["Rrightarrow"] = 8658,
6113 ["rArr"] = 8658,
6114 ["DoubleDownArrow"] = 8659,
6115 ["Downarrow"] = 8659,
6116 ["dArr"] = 8659,
6117 ["DoubleLeftRightArrow"] = 8660,
6118 ["Leftrightarrow"] = 8660,
6119 ["hArr"] = 8660,
6120 ["iff"] = 8660,
6121 ["DoubleUpDownArrow"] = 8661,
6122 ["Updownarrow"] = 8661,
6123 ["vArr"] = 8661,
6124 ["nwArr"] = 8662,
6125 ["neArr"] = 8663,
6126 ["seArr"] = 8664,
6127 ["swArr"] = 8665,
6128 ["Lleftarrow"] = 8666,
6129 ["lAarr"] = 8666,
6130 ["Rrightarrow"] = 8667,
6131 ["rAarr"] = 8667,
6132 ["zigrarr"] = 8669,
6133 ["LeftArrowBar"] = 8676,
6134 ["larrb"] = 8676,
6135 ["RightArrowBar"] = 8677,
6136 ["rarrb"] = 8677,

```

```

6137 ["DownArrowUpArrow"] = 8693,
6138 ["duarr"] = 8693,
6139 ["loarr"] = 8701,
6140 ["roarr"] = 8702,
6141 ["hoarr"] = 8703,
6142 ["ForAll"] = 8704,
6143 ["forall"] = 8704,
6144 ["comp"] = 8705,
6145 ["complement"] = 8705,
6146 ["PartialD"] = 8706,
6147 ["npart"] = {8706, 824},
6148 ["part"] = 8706,
6149 ["Exists"] = 8707,
6150 ["exist"] = 8707,
6151 ["NotExists"] = 8708,
6152 ["nexist"] = 8708,
6153 ["nexists"] = 8708,
6154 ["empty"] = 8709,
6155 ["emptyset"] = 8709,
6156 ["emptyv"] = 8709,
6157 ["varnothing"] = 8709,
6158 ["Del"] = 8711,
6159 ["nabla"] = 8711,
6160 ["Element"] = 8712,
6161 ["in"] = 8712,
6162 ["isin"] = 8712,
6163 ["isinv"] = 8712,
6164 ["NotElement"] = 8713,
6165 ["notin"] = 8713,
6166 ["notinva"] = 8713,
6167 ["ReverseElement"] = 8715,
6168 ["SuchThat"] = 8715,
6169 ["ni"] = 8715,
6170 ["niv"] = 8715,
6171 ["NotReverseElement"] = 8716,
6172 ["notni"] = 8716,
6173 ["notniva"] = 8716,
6174 ["Product"] = 8719,
6175 ["prod"] = 8719,
6176 ["Coproduct"] = 8720,
6177 ["coprod"] = 8720,
6178 ["Sum"] = 8721,
6179 ["sum"] = 8721,
6180 ["minus"] = 8722,
6181 ["MinusPlus"] = 8723,
6182 ["mnplus"] = 8723,
6183 ["mp"] = 8723,

```

```

6184 ["dotplus"] = 8724,
6185 ["plusdo"] = 8724,
6186 ["Backslash"] = 8726,
6187 ["setminus"] = 8726,
6188 ["setmn"] = 8726,
6189 ["smallsetminus"] = 8726,
6190 ["ssetmn"] = 8726,
6191 ["lowast"] = 8727,
6192 ["SmallCircle"] = 8728,
6193 ["compfn"] = 8728,
6194 ["Sqrt"] = 8730,
6195 ["radic"] = 8730,
6196 ["Proportional"] = 8733,
6197 ["prop"] = 8733,
6198 ["propto"] = 8733,
6199 ["varpropto"] = 8733,
6200 ["vprop"] = 8733,
6201 ["infin"] = 8734,
6202 ["angrt"] = 8735,
6203 ["ang"] = 8736,
6204 ["angle"] = 8736,
6205 ["nang"] = {8736, 8402},
6206 ["angmsd"] = 8737,
6207 ["measuredangle"] = 8737,
6208 ["angsph"] = 8738,
6209 ["VerticalBar"] = 8739,
6210 ["mid"] = 8739,
6211 ["shortmid"] = 8739,
6212 ["smid"] = 8739,
6213 ["NotVerticalBar"] = 8740,
6214 ["nmid"] = 8740,
6215 ["nshortmid"] = 8740,
6216 ["nsmid"] = 8740,
6217 ["DoubleVerticalBar"] = 8741,
6218 ["par"] = 8741,
6219 ["parallel"] = 8741,
6220 ["shortparallel"] = 8741,
6221 ["spar"] = 8741,
6222 ["NotDoubleVerticalBar"] = 8742,
6223 ["npar"] = 8742,
6224 ["nparallel"] = 8742,
6225 ["nshortparallel"] = 8742,
6226 ["nspar"] = 8742,
6227 ["and"] = 8743,
6228 ["wedge"] = 8743,
6229 ["or"] = 8744,
6230 ["vee"] = 8744,

```

```

6231 ["cap"] = 8745,
6232 ["caps"] = {8745, 65024},
6233 ["cup"] = 8746,
6234 ["cups"] = {8746, 65024},
6235 ["Integral"] = 8747,
6236 ["int"] = 8747,
6237 ["Int"] = 8748,
6238 ["iiint"] = 8749,
6239 ["tint"] = 8749,
6240 ["ContourIntegral"] = 8750,
6241 ["conint"] = 8750,
6242 ["oint"] = 8750,
6243 ["Conint"] = 8751,
6244 ["DoubleContourIntegral"] = 8751,
6245 ["Cconint"] = 8752,
6246 ["cwint"] = 8753,
6247 ["ClockwiseContourIntegral"] = 8754,
6248 ["cwconint"] = 8754,
6249 ["CounterClockwiseContourIntegral"] = 8755,
6250 ["awconint"] = 8755,
6251 ["Therefore"] = 8756,
6252 ["there4"] = 8756,
6253 ["therefore"] = 8756,
6254 ["Because"] = 8757,
6255 ["because"] = 8757,
6256 ["because"] = 8757,
6257 ["ratio"] = 8758,
6258 ["Colon"] = 8759,
6259 ["Proportion"] = 8759,
6260 ["dotminus"] = 8760,
6261 ["minusd"] = 8760,
6262 ["mDDot"] = 8762,
6263 ["homtht"] = 8763,
6264 ["Tilde"] = 8764,
6265 ["nvsim"] = {8764, 8402},
6266 ["sim"] = 8764,
6267 ["thicksim"] = 8764,
6268 ["thksim"] = 8764,
6269 ["backsim"] = 8765,
6270 ["bsim"] = 8765,
6271 ["race"] = {8765, 817},
6272 ["ac"] = 8766,
6273 ["acE"] = {8766, 819},
6274 ["mstpos"] = 8766,
6275 ["acd"] = 8767,
6276 ["VerticalTilde"] = 8768,
6277 ["wr"] = 8768,

```

```

6278 ["wreath"] = 8768,
6279 ["NotTilde"] = 8769,
6280 ["nsim"] = 8769,
6281 ["EqualTilde"] = 8770,
6282 ["NotEqualTilde"] = {8770, 824},
6283 ["eqsim"] = 8770,
6284 ["esim"] = 8770,
6285 ["nesim"] = {8770, 824},
6286 ["TildeEqual"] = 8771,
6287 ["sime"] = 8771,
6288 ["simeq"] = 8771,
6289 ["NotTildeEqual"] = 8772,
6290 ["nsime"] = 8772,
6291 ["nsimeq"] = 8772,
6292 ["TildeFullEqual"] = 8773,
6293 ["cong"] = 8773,
6294 ["simne"] = 8774,
6295 ["NotTildeFullEqual"] = 8775,
6296 ["ncong"] = 8775,
6297 ["TildeTilde"] = 8776,
6298 ["ap"] = 8776,
6299 ["approx"] = 8776,
6300 ["asymp"] = 8776,
6301 ["thickapprox"] = 8776,
6302 ["thkap"] = 8776,
6303 ["NotTildeTilde"] = 8777,
6304 ["nap"] = 8777,
6305 ["napprox"] = 8777,
6306 ["ape"] = 8778,
6307 ["approxeq"] = 8778,
6308 ["apid"] = 8779,
6309 ["napid"] = {8779, 824},
6310 ["backcong"] = 8780,
6311 ["bcong"] = 8780,
6312 ["CupCap"] = 8781,
6313 ["asympeq"] = 8781,
6314 ["nvap"] = {8781, 8402},
6315 ["Bumpeq"] = 8782,
6316 ["HumpDownHump"] = 8782,
6317 ["NotHumpDownHump"] = {8782, 824},
6318 ["bump"] = 8782,
6319 ["nbump"] = {8782, 824},
6320 ["HumpEqual"] = 8783,
6321 ["NotHumpEqual"] = {8783, 824},
6322 ["bumpe"] = 8783,
6323 ["bumpeq"] = 8783,
6324 ["nbumpe"] = {8783, 824},

```

```

6325 ["DotEqual"] = 8784,
6326 ["doteq"] = 8784,
6327 ["esdot"] = 8784,
6328 ["nedot"] = {8784, 824},
6329 ["doteqdot"] = 8785,
6330 ["eDot"] = 8785,
6331 ["efDot"] = 8786,
6332 ["fallingdotseq"] = 8786,
6333 ["erDot"] = 8787,
6334 ["risingdotseq"] = 8787,
6335 ["Assign"] = 8788,
6336 ["colone"] = 8788,
6337 ["coloneq"] = 8788,
6338 ["ecolon"] = 8789,
6339 ["eqcolon"] = 8789,
6340 ["ecir"] = 8790,
6341 ["eqcirc"] = 8790,
6342 ["circeq"] = 8791,
6343 ["cire"] = 8791,
6344 ["wedgeq"] = 8793,
6345 ["veeeq"] = 8794,
6346 ["triangleq"] = 8796,
6347 ["trie"] = 8796,
6348 ["equest"] = 8799,
6349 ["questeq"] = 8799,
6350 ["NotEqual"] = 8800,
6351 ["ne"] = 8800,
6352 ["Congruent"] = 8801,
6353 ["bnequiv"] = {8801, 8421},
6354 ["equiv"] = 8801,
6355 ["NotCongruent"] = 8802,
6356 ["nequiv"] = 8802,
6357 ["le"] = 8804,
6358 ["leq"] = 8804,
6359 ["nvle"] = {8804, 8402},
6360 ["GreaterEqual"] = 8805,
6361 ["ge"] = 8805,
6362 ["geq"] = 8805,
6363 ["nvge"] = {8805, 8402},
6364 ["LessFullEqual"] = 8806,
6365 ["lE"] = 8806,
6366 ["leqq"] = 8806,
6367 ["nlE"] = {8806, 824},
6368 ["nleqq"] = {8806, 824},
6369 ["GreaterFullEqual"] = 8807,
6370 ["NotGreaterFullEqual"] = {8807, 824},
6371 ["gE"] = 8807,

```



```

6372 ["geqq"] = 8807,
6373 ["ngE"] = {8807, 824},
6374 ["ngeqq"] = {8807, 824},
6375 ["lnE"] = 8808,
6376 ["lneqq"] = 8808,
6377 ["lvertneqq"] = {8808, 65024},
6378 ["lvnE"] = {8808, 65024},
6379 ["gnE"] = 8809,
6380 ["gneqq"] = 8809,
6381 ["gvertneqq"] = {8809, 65024},
6382 ["gvnE"] = {8809, 65024},
6383 ["Lt"] = 8810,
6384 ["NestedLessLess"] = 8810,
6385 ["NotLessLess"] = {8810, 824},
6386 ["ll"] = 8810,
6387 ["nLt"] = {8810, 8402},
6388 ["nLtv"] = {8810, 824},
6389 ["Gt"] = 8811,
6390 ["NestedGreaterGreater"] = 8811,
6391 ["NotGreaterGreater"] = {8811, 824},
6392 ["gg"] = 8811,
6393 ["nGt"] = {8811, 8402},
6394 ["nGtv"] = {8811, 824},
6395 ["between"] = 8812,
6396 ["twixt"] = 8812,
6397 ["NotCupCap"] = 8813,
6398 ["NotLess"] = 8814,
6399 ["nless"] = 8814,
6400 ["nlt"] = 8814,
6401 ["NotGreater"] = 8815,
6402 ["ngt"] = 8815,
6403 ["ngtr"] = 8815,
6404 ["NotLessEqual"] = 8816,
6405 ["nle"] = 8816,
6406 ["nleq"] = 8816,
6407 ["NotGreaterEqual"] = 8817,
6408 ["nge"] = 8817,
6409 ["ngeq"] = 8817,
6410 ["LessTilde"] = 8818,
6411 ["lesssim"] = 8818,
6412 ["lsim"] = 8818,
6413 ["GreaterTilde"] = 8819,
6414 ["gsim"] = 8819,
6415 ["gtrsim"] = 8819,
6416 ["NotLessTilde"] = 8820,
6417 ["nlsim"] = 8820,
6418 ["NotGreaterTilde"] = 8821,

```

```

6419 ["ngsim"] = 8821,
6420 ["LessGreater"] = 8822,
6421 ["lessgtr"] = 8822,
6422 ["lg"] = 8822,
6423 ["GreaterLess"] = 8823,
6424 ["gl"] = 8823,
6425 ["gtrless"] = 8823,
6426 ["NotLessGreater"] = 8824,
6427 ["ntl原因"] = 8824,
6428 ["NotGreaterLess"] = 8825,
6429 ["ntgl"] = 8825,
6430 ["Precedes"] = 8826,
6431 ["pr"] = 8826,
6432 ["prec"] = 8826,
6433 ["Succeeds"] = 8827,
6434 ["sc"] = 8827,
6435 ["succ"] = 8827,
6436 ["PrecedesSlantEqual"] = 8828,
6437 ["prcue"] = 8828,
6438 ["preccurlyeq"] = 8828,
6439 ["SucceedsSlantEqual"] = 8829,
6440 ["sccue"] = 8829,
6441 ["succcurlyeq"] = 8829,
6442 ["PrecedesTilde"] = 8830,
6443 ["precsim"] = 8830,
6444 ["prsim"] = 8830,
6445 ["NotSucceedsTilde"] = {8831, 824},
6446 ["SucceedsTilde"] = 8831,
6447 ["scsim"] = 8831,
6448 ["succsim"] = 8831,
6449 ["NotPrecedes"] = 8832,
6450 ["npr"] = 8832,
6451 ["nprec"] = 8832,
6452 ["NotSucceeds"] = 8833,
6453 ["nsc"] = 8833,
6454 ["nsucc"] = 8833,
6455 ["NotSubset"] = {8834, 8402},
6456 ["nsubset"] = {8834, 8402},
6457 ["sub"] = 8834,
6458 ["subset"] = 8834,
6459 ["vnsup"] = {8834, 8402},
6460 ["NotSuperset"] = {8835, 8402},
6461 ["Superset"] = 8835,
6462 ["nsupset"] = {8835, 8402},
6463 ["sup"] = 8835,
6464 ["supset"] = 8835,
6465 ["vnsup"] = {8835, 8402},

```

```

6466 ["nsub"] = 8836,
6467 ["nsup"] = 8837,
6468 ["SubsetEqual"] = 8838,
6469 ["sube"] = 8838,
6470 ["subseteq"] = 8838,
6471 ["SupersetEqual"] = 8839,
6472 ["supe"] = 8839,
6473 ["supseteq"] = 8839,
6474 ["NotSubsetEqual"] = 8840,
6475 ["nsube"] = 8840,
6476 ["nsubseteq"] = 8840,
6477 ["NotSupersetEqual"] = 8841,
6478 ["nsupe"] = 8841,
6479 ["nsupseteq"] = 8841,
6480 ["subne"] = 8842,
6481 ["subsetneq"] = 8842,
6482 ["varsubsetneq"] = {8842, 65024},
6483 ["vsubne"] = {8842, 65024},
6484 ["supne"] = 8843,
6485 ["supsetneq"] = 8843,
6486 ["varsupsetneq"] = {8843, 65024},
6487 ["vsupne"] = {8843, 65024},
6488 ["cupdot"] = 8845,
6489 ["UnionPlus"] = 8846,
6490 ["uplus"] = 8846,
6491 ["NotSquareSubset"] = {8847, 824},
6492 ["SquareSubset"] = 8847,
6493 ["sqsub"] = 8847,
6494 ["sqsubset"] = 8847,
6495 ["NotSquareSuperset"] = {8848, 824},
6496 ["SquareSuperset"] = 8848,
6497 ["sqsup"] = 8848,
6498 ["sqsupset"] = 8848,
6499 ["SquareSubsetEqual"] = 8849,
6500 ["sqsube"] = 8849,
6501 ["sqsubseteq"] = 8849,
6502 ["SquareSupersetEqual"] = 8850,
6503 ["sqsupe"] = 8850,
6504 ["sqsupseteq"] = 8850,
6505 ["SquareIntersection"] = 8851,
6506 ["sqcap"] = 8851,
6507 ["sqcaps"] = {8851, 65024},
6508 ["SquareUnion"] = 8852,
6509 ["sqcup"] = 8852,
6510 ["sqcups"] = {8852, 65024},
6511 ["CirclePlus"] = 8853,
6512 ["oplus"] = 8853,

```

```

6513 ["CircleMinus"] = 8854,
6514 ["ominus"] = 8854,
6515 ["CircleTimes"] = 8855,
6516 ["otimes"] = 8855,
6517 ["osol"] = 8856,
6518 ["CircleDot"] = 8857,
6519 ["odot"] = 8857,
6520 ["circledcirc"] = 8858,
6521 ["ocir"] = 8858,
6522 ["circledast"] = 8859,
6523 ["oast"] = 8859,
6524 ["circleddash"] = 8861,
6525 ["odash"] = 8861,
6526 ["boxplus"] = 8862,
6527 ["plusb"] = 8862,
6528 ["boxminus"] = 8863,
6529 ["minusb"] = 8863,
6530 ["boxtimes"] = 8864,
6531 ["timesb"] = 8864,
6532 ["dotsquare"] = 8865,
6533 ["sdotb"] = 8865,
6534 ["RightTee"] = 8866,
6535 ["vdash"] = 8866,
6536 ["LeftTee"] = 8867,
6537 ["dashv"] = 8867,
6538 ["DownTee"] = 8868,
6539 ["top"] = 8868,
6540 ["UpTee"] = 8869,
6541 ["bot"] = 8869,
6542 ["bottom"] = 8869,
6543 ["perp"] = 8869,
6544 ["models"] = 8871,
6545 ["DoubleRightTee"] = 8872,
6546 ["vDash"] = 8872,
6547 ["Vdash"] = 8873,
6548 ["Vvdash"] = 8874,
6549 ["VDash"] = 8875,
6550 ["nvDash"] = 8876,
6551 ["nvDash"] = 8877,
6552 ["nVdash"] = 8878,
6553 ["nVDash"] = 8879,
6554 ["prurel"] = 8880,
6555 ["LeftTriangle"] = 8882,
6556 ["vartriangleleft"] = 8882,
6557 ["vltri"] = 8882,
6558 ["RightTriangle"] = 8883,
6559 ["vartriangleright"] = 8883,

```

```

6560 ["vrtri"] = 8883,
6561 ["LeftTriangleEqual"] = 8884,
6562 ["ltrie"] = 8884,
6563 ["nvltrie"] = {8884, 8402},
6564 ["trianglelefteq"] = 8884,
6565 ["RightTriangleEqual"] = 8885,
6566 ["nvrtrie"] = {8885, 8402},
6567 ["rtrie"] = 8885,
6568 ["trianglerighteq"] = 8885,
6569 ["origof"] = 8886,
6570 ["imof"] = 8887,
6571 ["multimap"] = 8888,
6572 ["mumap"] = 8888,
6573 ["hercon"] = 8889,
6574 ["intcal"] = 8890,
6575 ["intercal"] = 8890,
6576 ["veebar"] = 8891,
6577 ["barvee"] = 8893,
6578 ["angrtvb"] = 8894,
6579 ["ltri"] = 8895,
6580 ["Wedge"] = 8896,
6581 ["bigwedge"] = 8896,
6582 ["xwedge"] = 8896,
6583 ["Vee"] = 8897,
6584 ["bigvee"] = 8897,
6585 ["xvee"] = 8897,
6586 ["Intersection"] = 8898,
6587 ["bigcap"] = 8898,
6588 ["xcap"] = 8898,
6589 ["Union"] = 8899,
6590 ["bigcup"] = 8899,
6591 ["xcup"] = 8899,
6592 ["Diamond"] = 8900,
6593 ["diam"] = 8900,
6594 ["diamond"] = 8900,
6595 ["sdot"] = 8901,
6596 ["Star"] = 8902,
6597 ["sstarf"] = 8902,
6598 ["divideontimes"] = 8903,
6599 ["divonx"] = 8903,
6600 ["bowtie"] = 8904,
6601 ["ltimes"] = 8905,
6602 ["rtimes"] = 8906,
6603 ["leftthreetimes"] = 8907,
6604 ["lthree"] = 8907,
6605 ["rightthreetimes"] = 8908,
6606 ["rthree"] = 8908,

```

```

6607 ["backsimeq"] = 8909,
6608 ["bsime"] = 8909,
6609 ["curlyvee"] = 8910,
6610 ["cuvee"] = 8910,
6611 ["curlywedge"] = 8911,
6612 ["cuwed"] = 8911,
6613 ["Sub"] = 8912,
6614 ["Subset"] = 8912,
6615 ["Sup"] = 8913,
6616 ["Supset"] = 8913,
6617 ["Cap"] = 8914,
6618 ["Cup"] = 8915,
6619 ["fork"] = 8916,
6620 ["pitchfork"] = 8916,
6621 ["epar"] = 8917,
6622 ["lessdot"] = 8918,
6623 ["ltdot"] = 8918,
6624 ["gtdot"] = 8919,
6625 ["gtrdot"] = 8919,
6626 ["Ll"] = 8920,
6627 ["nLl"] = {8920, 824},
6628 ["Gg"] = 8921,
6629 ["ggg"] = 8921,
6630 ["nGg"] = {8921, 824},
6631 ["LessEqualGreater"] = 8922,
6632 ["leg"] = 8922,
6633 ["lesg"] = {8922, 65024},
6634 ["lesseqgtr"] = 8922,
6635 ["GreaterEqualLess"] = 8923,
6636 ["gel"] = 8923,
6637 ["gesl"] = {8923, 65024},
6638 ["gtreqless"] = 8923,
6639 ["cuepr"] = 8926,
6640 ["curlyeqprec"] = 8926,
6641 ["cuesc"] = 8927,
6642 ["curlyeqsucc"] = 8927,
6643 ["NotPrecedesSlantEqual"] = 8928,
6644 ["nprcue"] = 8928,
6645 ["NotSucceedsSlantEqual"] = 8929,
6646 ["nsccue"] = 8929,
6647 ["NotSquareSubsetEqual"] = 8930,
6648 ["nsqsube"] = 8930,
6649 ["NotSquareSupersetEqual"] = 8931,
6650 ["nsqsupe"] = 8931,
6651 ["lnsim"] = 8934,
6652 ["gnsim"] = 8935,
6653 ["precnsim"] = 8936,

```

```

6654 ["prnsim"] = 8936,
6655 ["scnsim"] = 8937,
6656 ["succnsim"] = 8937,
6657 ["NotLeftTriangle"] = 8938,
6658 ["nltri"] = 8938,
6659 ["ntriangleleft"] = 8938,
6660 ["NotRightTriangle"] = 8939,
6661 ["nrtri"] = 8939,
6662 ["ntriangleright"] = 8939,
6663 ["NotLeftTriangleEqual"] = 8940,
6664 ["nltrie"] = 8940,
6665 ["ntrianglelefteq"] = 8940,
6666 ["NotRightTriangleEqual"] = 8941,
6667 ["nrtrie"] = 8941,
6668 ["ntrianglerighteq"] = 8941,
6669 ["vellip"] = 8942,
6670 ["ctdot"] = 8943,
6671 ["utdot"] = 8944,
6672 ["dtdot"] = 8945,
6673 ["disin"] = 8946,
6674 ["isinsv"] = 8947,
6675 ["isins"] = 8948,
6676 ["isindot"] = 8949,
6677 ["notindot"] = {8949, 824},
6678 ["notinvc"] = 8950,
6679 ["notinvb"] = 8951,
6680 ["isinE"] = 8953,
6681 ["notinE"] = {8953, 824},
6682 ["nisd"] = 8954,
6683 ["xnis"] = 8955,
6684 ["nis"] = 8956,
6685 ["notnivc"] = 8957,
6686 ["notnivb"] = 8958,
6687 ["barwed"] = 8965,
6688 ["barwedge"] = 8965,
6689 ["Barwed"] = 8966,
6690 ["doublebarwedge"] = 8966,
6691 ["LeftCeiling"] = 8968,
6692 ["lceil"] = 8968,
6693 ["RightCeiling"] = 8969,
6694 ["rceil"] = 8969,
6695 ["LeftFloor"] = 8970,
6696 ["lfloor"] = 8970,
6697 ["RightFloor"] = 8971,
6698 ["rfloor"] = 8971,
6699 ["drcrop"] = 8972,
6700 ["dlcrop"] = 8973,

```

6701 ["urcrop"] = 8974,  
 6702 ["ulcrop"] = 8975,  
 6703 ["bnot"] = 8976,  
 6704 ["proflin"] = 8978,  
 6705 ["profsurf"] = 8979,  
 6706 ["telrec"] = 8981,  
 6707 ["target"] = 8982,  
 6708 ["ulcorn"] = 8988,  
 6709 ["ulcorner"] = 8988,  
 6710 ["urcorn"] = 8989,  
 6711 ["urcorner"] = 8989,  
 6712 ["dlcorn"] = 8990,  
 6713 ["llcorner"] = 8990,  
 6714 ["drcorn"] = 8991,  
 6715 ["lrcorner"] = 8991,  
 6716 ["frown"] = 8994,  
 6717 ["sfrown"] = 8994,  
 6718 ["smile"] = 8995,  
 6719 ["ssmile"] = 8995,  
 6720 ["cylcty"] = 9005,  
 6721 ["profalar"] = 9006,  
 6722 ["topbot"] = 9014,  
 6723 ["ovbar"] = 9021,  
 6724 ["solbar"] = 9023,  
 6725 ["angzarr"] = 9084,  
 6726 ["lmoust"] = 9136,  
 6727 ["lmoustache"] = 9136,  
 6728 ["rmoust"] = 9137,  
 6729 ["rmoustache"] = 9137,  
 6730 ["OverBracket"] = 9140,  
 6731 ["tbrk"] = 9140,  
 6732 ["UnderBracket"] = 9141,  
 6733 ["bbrk"] = 9141,  
 6734 ["bbrktbrk"] = 9142,  
 6735 ["OverParenthesis"] = 9180,  
 6736 ["UnderParenthesis"] = 9181,  
 6737 ["OverBrace"] = 9182,  
 6738 ["UnderBrace"] = 9183,  
 6739 ["trpezium"] = 9186,  
 6740 ["elinters"] = 9191,  
 6741 ["blank"] = 9251,  
 6742 ["circledS"] = 9416,  
 6743 ["oS"] = 9416,  
 6744 ["HorizontalLine"] = 9472,  
 6745 ["boxh"] = 9472,  
 6746 ["boxv"] = 9474,  
 6747 ["boxdr"] = 9484,



```

6748 ["boxdl"] = 9488,
6749 ["boxur"] = 9492,
6750 ["boxul"] = 9496,
6751 ["boxvr"] = 9500,
6752 ["boxvl"] = 9508,
6753 ["boxhd"] = 9516,
6754 ["boxhu"] = 9524,
6755 ["boxvh"] = 9532,
6756 ["boxH"] = 9552,
6757 ["boxV"] = 9553,
6758 ["boxdR"] = 9554,
6759 ["boxDr"] = 9555,
6760 ["boxDR"] = 9556,
6761 ["boxdL"] = 9557,
6762 ["boxDl"] = 9558,
6763 ["boxDL"] = 9559,
6764 ["boxuR"] = 9560,
6765 ["boxUr"] = 9561,
6766 ["boxUR"] = 9562,
6767 ["boxuL"] = 9563,
6768 ["boxUl"] = 9564,
6769 ["boxUL"] = 9565,
6770 ["boxvR"] = 9566,
6771 ["boxVr"] = 9567,
6772 ["boxVR"] = 9568,
6773 ["boxvL"] = 9569,
6774 ["boxVl"] = 9570,
6775 ["boxVL"] = 9571,
6776 ["boxHd"] = 9572,
6777 ["boxhD"] = 9573,
6778 ["boxHD"] = 9574,
6779 ["boxHu"] = 9575,
6780 ["boxhU"] = 9576,
6781 ["boxHU"] = 9577,
6782 ["boxvH"] = 9578,
6783 ["boxVh"] = 9579,
6784 ["boxVH"] = 9580,
6785 ["uhblk"] = 9600,
6786 ["lhblk"] = 9604,
6787 ["block"] = 9608,
6788 ["blk14"] = 9617,
6789 ["blk12"] = 9618,
6790 ["blk34"] = 9619,
6791 ["Square"] = 9633,
6792 ["squ"] = 9633,
6793 ["square"] = 9633,
6794 ["FilledVerySmallSquare"] = 9642,

```

```

6795 ["blacksquare"] = 9642,
6796 ["suarf"] = 9642,
6797 ["squf"] = 9642,
6798 ["EmptyVerySmallSquare"] = 9643,
6799 ["rect"] = 9645,
6800 ["marker"] = 9646,
6801 ["fltns"] = 9649,
6802 ["bigtriangleup"] = 9651,
6803 ["xutri"] = 9651,
6804 ["blacktriangle"] = 9652,
6805 ["utri"] = 9652,
6806 ["triangle"] = 9653,
6807 ["utri"] = 9653,
6808 ["blacktriangleright"] = 9656,
6809 ["rtri"] = 9656,
6810 ["rtri"] = 9657,
6811 ["triangleright"] = 9657,
6812 ["bigtriangledown"] = 9661,
6813 ["xdtri"] = 9661,
6814 ["blacktriangledown"] = 9662,
6815 ["dtri"] = 9662,
6816 ["dtri"] = 9663,
6817 ["triangledown"] = 9663,
6818 ["blacktriangleleft"] = 9666,
6819 ["ltrif"] = 9666,
6820 ["ltrif"] = 9667,
6821 ["triangleleft"] = 9667,
6822 ["loz"] = 9674,
6823 ["lozenge"] = 9674,
6824 ["cir"] = 9675,
6825 ["tridot"] = 9708,
6826 ["bigcirc"] = 9711,
6827 ["xcirc"] = 9711,
6828 ["ultri"] = 9720,
6829 ["urtri"] = 9721,
6830 ["lltri"] = 9722,
6831 ["EmptySmallSquare"] = 9723,
6832 ["FilledSmallSquare"] = 9724,
6833 ["bigstar"] = 9733,
6834 ["starf"] = 9733,
6835 ["star"] = 9734,
6836 ["phone"] = 9742,
6837 ["female"] = 9792,
6838 ["male"] = 9794,
6839 ["spades"] = 9824,
6840 ["spadesuit"] = 9824,
6841 ["clubs"] = 9827,

```

```

6842 ["clubsuit"] = 9827,
6843 ["hearts"] = 9829,
6844 ["heartsuit"] = 9829,
6845 ["diamondsuit"] = 9830,
6846 ["diams"] = 9830,
6847 ["sung"] = 9834,
6848 ["flat"] = 9837,
6849 ["natur"] = 9838,
6850 ["natural"] = 9838,
6851 ["sharp"] = 9839,
6852 ["check"] = 10003,
6853 ["checkmark"] = 10003,
6854 ["cross"] = 10007,
6855 ["malt"] = 10016,
6856 ["maltese"] = 10016,
6857 ["sext"] = 10038,
6858 ["VerticalSeparator"] = 10072,
6859 ["lbbbrk"] = 10098,
6860 ["rbbrk"] = 10099,
6861 ["bsolhsub"] = 10184,
6862 ["suphsol"] = 10185,
6863 ["LeftDoubleBracket"] = 10214,
6864 ["lobrk"] = 10214,
6865 ["RightDoubleBracket"] = 10215,
6866 ["robrk"] = 10215,
6867 ["LeftAngleBracket"] = 10216,
6868 ["lang"] = 10216,
6869 ["langle"] = 10216,
6870 ["RightAngleBracket"] = 10217,
6871 ["rang"] = 10217,
6872 ["rangle"] = 10217,
6873 ["Lang"] = 10218,
6874 ["Rang"] = 10219,
6875 ["loang"] = 10220,
6876 ["roang"] = 10221,
6877 ["LongLeftArrow"] = 10229,
6878 ["longleftarrow"] = 10229,
6879 ["xlarr"] = 10229,
6880 ["LongRightArrow"] = 10230,
6881 ["longrightarrow"] = 10230,
6882 ["xrarr"] = 10230,
6883 ["LongLeftRightArrow"] = 10231,
6884 ["longlefttrightarrow"] = 10231,
6885 ["xharr"] = 10231,
6886 ["DoubleLongLeftArrow"] = 10232,
6887 ["Longleftarrow"] = 10232,
6888 ["xlArr"] = 10232,

```

```

6889 ["DoubleLongRightArrow"] = 10233,
6890 ["Longrightarrow"] = 10233,
6891 ["xrArr"] = 10233,
6892 ["DoubleLongLeftRightArrow"] = 10234,
6893 ["Longlefttrightrightarrow"] = 10234,
6894 ["xhArr"] = 10234,
6895 ["longmapsto"] = 10236,
6896 ["xmap"] = 10236,
6897 ["dzigrarr"] = 10239,
6898 ["nvlArr"] = 10498,
6899 ["nvrArr"] = 10499,
6900 ["nvHarr"] = 10500,
6901 ["Map"] = 10501,
6902 ["lbarr"] = 10508,
6903 ["bkarow"] = 10509,
6904 ["rbarr"] = 10509,
6905 ["lBarr"] = 10510,
6906 ["dbkarow"] = 10511,
6907 ["rBarr"] = 10511,
6908 ["RBarr"] = 10512,
6909 ["drbkarow"] = 10512,
6910 ["DDottrahd"] = 10513,
6911 ["UpArrowBar"] = 10514,
6912 ["DownArrowBar"] = 10515,
6913 ["Rarrtl"] = 10518,
6914 ["latail"] = 10521,
6915 ["ratail"] = 10522,
6916 ["lAtail"] = 10523,
6917 ["rAtail"] = 10524,
6918 ["larrfs"] = 10525,
6919 ["rarrfs"] = 10526,
6920 ["larrbfs"] = 10527,
6921 ["rarrbfs"] = 10528,
6922 ["nwarhk"] = 10531,
6923 ["nearhk"] = 10532,
6924 ["hksearow"] = 10533,
6925 ["searhk"] = 10533,
6926 ["hkswarow"] = 10534,
6927 ["swarhk"] = 10534,
6928 ["nwnear"] = 10535,
6929 ["nesear"] = 10536,
6930 ["toea"] = 10536,
6931 ["seswar"] = 10537,
6932 ["tosa"] = 10537,
6933 ["swnwar"] = 10538,
6934 ["nrarrc"] = {10547, 824},
6935 ["rarrc"] = 10547,

```

```

6936 ["cudarr"] = 10549,
6937 ["ldca"] = 10550,
6938 ["rdca"] = 10551,
6939 ["cudarrr"] = 10552,
6940 ["larrpl"] = 10553,
6941 ["curarrm"] = 10556,
6942 ["cularrp"] = 10557,
6943 ["rarrpl"] = 10565,
6944 ["harrcir"] = 10568,
6945 ["Uarroccir"] = 10569,
6946 ["lurdshar"] = 10570,
6947 ["ldrushar"] = 10571,
6948 ["LeftRightVector"] = 10574,
6949 ["RightUpDownVector"] = 10575,
6950 ["DownLeftRightVector"] = 10576,
6951 ["LeftUpDownVector"] = 10577,
6952 ["LeftVectorBar"] = 10578,
6953 ["RightVectorBar"] = 10579,
6954 ["RightUpVectorBar"] = 10580,
6955 ["RightDownVectorBar"] = 10581,
6956 ["DownLeftVectorBar"] = 10582,
6957 ["DownRightVectorBar"] = 10583,
6958 ["LeftUpVectorBar"] = 10584,
6959 ["LeftDownVectorBar"] = 10585,
6960 ["LeftTeeVector"] = 10586,
6961 ["RightTeeVector"] = 10587,
6962 ["RightUpTeeVector"] = 10588,
6963 ["RightDownTeeVector"] = 10589,
6964 ["DownLeftTeeVector"] = 10590,
6965 ["DownRightTeeVector"] = 10591,
6966 ["LeftUpTeeVector"] = 10592,
6967 ["LeftDownTeeVector"] = 10593,
6968 ["lHar"] = 10594,
6969 ["uHar"] = 10595,
6970 ["rHar"] = 10596,
6971 ["dHar"] = 10597,
6972 ["luruhar"] = 10598,
6973 ["ldrdhar"] = 10599,
6974 ["ruluhar"] = 10600,
6975 ["rdldhar"] = 10601,
6976 ["lharul"] = 10602,
6977 ["llhard"] = 10603,
6978 ["rharul"] = 10604,
6979 ["lrhard"] = 10605,
6980 ["UpEquilibrium"] = 10606,
6981 ["udhar"] = 10606,
6982 ["ReverseUpEquilibrium"] = 10607,

```

```

6983 ["duhar"] = 10607,
6984 ["RoundImplies"] = 10608,
6985 ["erarr"] = 10609,
6986 ["simrarr"] = 10610,
6987 ["larrsim"] = 10611,
6988 ["rarrsim"] = 10612,
6989 ["rarrap"] = 10613,
6990 ["ltlarr"] = 10614,
6991 ["gtrarr"] = 10616,
6992 ["subrarr"] = 10617,
6993 ["suplarr"] = 10619,
6994 ["lfisht"] = 10620,
6995 ["rfisht"] = 10621,
6996 ["ufisht"] = 10622,
6997 ["dfisht"] = 10623,
6998 ["lopar"] = 10629,
6999 ["ropar"] = 10630,
7000 ["lbrke"] = 10635,
7001 ["rbrke"] = 10636,
7002 ["lbrkslu"] = 10637,
7003 ["rbrksld"] = 10638,
7004 ["lbrksld"] = 10639,
7005 ["rbrkslu"] = 10640,
7006 ["langd"] = 10641,
7007 ["rangd"] = 10642,
7008 ["lparlt"] = 10643,
7009 ["rpargt"] = 10644,
7010 ["gtlPar"] = 10645,
7011 ["ltrPar"] = 10646,
7012 ["vzigzag"] = 10650,
7013 ["vangrt"] = 10652,
7014 ["angrtvbd"] = 10653,
7015 ["ange"] = 10660,
7016 ["range"] = 10661,
7017 ["dwangle"] = 10662,
7018 ["uwangle"] = 10663,
7019 ["angmsdaa"] = 10664,
7020 ["angmsdab"] = 10665,
7021 ["angmsdac"] = 10666,
7022 ["angmsdad"] = 10667,
7023 ["angmsdae"] = 10668,
7024 ["angmsdaf"] = 10669,
7025 ["angmsdag"] = 10670,
7026 ["angmsdah"] = 10671,
7027 ["bemptyv"] = 10672,
7028 ["demptyv"] = 10673,
7029 ["cemptyv"] = 10674,

```

```

7030 ["raemptyv"] = 10675,
7031 ["laemptyv"] = 10676,
7032 ["ohbar"] = 10677,
7033 ["omid"] = 10678,
7034 ["opar"] = 10679,
7035 ["operp"] = 10681,
7036 ["olcross"] = 10683,
7037 ["odsold"] = 10684,
7038 ["olcir"] = 10686,
7039 ["ofcir"] = 10687,
7040 ["olt"] = 10688,
7041 ["ogt"] = 10689,
7042 ["cirscir"] = 10690,
7043 ["cirE"] = 10691,
7044 ["solb"] = 10692,
7045 ["bsolb"] = 10693,
7046 ["boxbox"] = 10697,
7047 ["trisb"] = 10701,
7048 ["rtriltri"] = 10702,
7049 ["LeftTriangleBar"] = 10703,
7050 ["NotLeftTriangleBar"] = {10703, 824},
7051 ["NotRightTriangleBar"] = {10704, 824},
7052 ["RightTriangleBar"] = 10704,
7053 ["i infin"] = 10716,
7054 ["infintie"] = 10717,
7055 ["nvinfin"] = 10718,
7056 ["eparsl"] = 10723,
7057 ["smeparsl"] = 10724,
7058 ["eqvparsl"] = 10725,
7059 ["blacklozenge"] = 10731,
7060 ["lozf"] = 10731,
7061 ["RuleDelayed"] = 10740,
7062 ["dsol"] = 10742,
7063 ["bigodot"] = 10752,
7064 ["xodot"] = 10752,
7065 ["bigoplus"] = 10753,
7066 ["xoplus"] = 10753,
7067 ["bigotimes"] = 10754,
7068 ["xotime"] = 10754,
7069 ["biguplus"] = 10756,
7070 ["xuplus"] = 10756,
7071 ["bigscup"] = 10758,
7072 ["xscup"] = 10758,
7073 ["iiint"] = 10764,
7074 ["qint"] = 10764,
7075 ["fpartint"] = 10765,
7076 ["cirfnint"] = 10768,

```

```

7077 ["awint"] = 10769,
7078 ["rppolint"] = 10770,
7079 ["scpolint"] = 10771,
7080 ["npolint"] = 10772,
7081 ["pointint"] = 10773,
7082 ["quatint"] = 10774,
7083 ["intlarhk"] = 10775,
7084 ["pluscir"] = 10786,
7085 ["plusacir"] = 10787,
7086 ["simplus"] = 10788,
7087 ["plusdu"] = 10789,
7088 ["plussim"] = 10790,
7089 ["plustwo"] = 10791,
7090 ["mcomma"] = 10793,
7091 ["minusdu"] = 10794,
7092 ["loplus"] = 10797,
7093 ["roplus"] = 10798,
7094 ["Cross"] = 10799,
7095 ["timesd"] = 10800,
7096 ["timesbar"] = 10801,
7097 ["smashp"] = 10803,
7098 ["lotimes"] = 10804,
7099 ["rotimes"] = 10805,
7100 ["otimesas"] = 10806,
7101 ["Otimes"] = 10807,
7102 ["odiv"] = 10808,
7103 ["triplus"] = 10809,
7104 ["triminus"] = 10810,
7105 ["tritime"] = 10811,
7106 ["intprod"] = 10812,
7107 ["iproduct"] = 10812,
7108 ["amalg"] = 10815,
7109 ["capdot"] = 10816,
7110 ["ncup"] = 10818,
7111 ["ncap"] = 10819,
7112 ["capand"] = 10820,
7113 ["cupor"] = 10821,
7114 ["cupcap"] = 10822,
7115 ["capcup"] = 10823,
7116 ["cupbrcap"] = 10824,
7117 ["capbrcup"] = 10825,
7118 ["cupcup"] = 10826,
7119 ["capcap"] = 10827,
7120 ["ccups"] = 10828,
7121 ["ccaps"] = 10829,
7122 ["ccupssm"] = 10832,
7123 ["And"] = 10835,

```



7124 ["Or"] = 10836,  
7125 ["andand"] = 10837,  
7126 ["oror"] = 10838,  
7127 ["orslope"] = 10839,  
7128 ["andslope"] = 10840,  
7129 ["andv"] = 10842,  
7130 ["orv"] = 10843,  
7131 ["andd"] = 10844,  
7132 ["ord"] = 10845,  
7133 ["wedbar"] = 10847,  
7134 ["sdote"] = 10854,  
7135 ["simdot"] = 10858,  
7136 ["congdote"] = 10861,  
7137 ["ncongdote"] = {10861, 824},  
7138 ["easter"] = 10862,  
7139 ["apacir"] = 10863,  
7140 ["apE"] = 10864,  
7141 ["napE"] = {10864, 824},  
7142 ["eplus"] = 10865,  
7143 ["pluse"] = 10866,  
7144 ["Esim"] = 10867,  
7145 ["Colone"] = 10868,  
7146 ["Equal"] = 10869,  
7147 ["ddotseq"] = 10871,  
7148 ["eDDot"] = 10871,  
7149 ["equivDD"] = 10872,  
7150 ["ltcir"] = 10873,  
7151 ["gtcir"] = 10874,  
7152 ["ltquest"] = 10875,  
7153 ["gtquest"] = 10876,  
7154 ["LessSlantEqual"] = 10877,  
7155 ["NotLessSlantEqual"] = {10877, 824},  
7156 ["leqslant"] = 10877,  
7157 ["les"] = 10877,  
7158 ["nleqslant"] = {10877, 824},  
7159 ["nles"] = {10877, 824},  
7160 ["GreaterSlantEqual"] = 10878,  
7161 ["NotGreaterSlantEqual"] = {10878, 824},  
7162 ["geqslant"] = 10878,  
7163 ["ges"] = 10878,  
7164 ["ngeqslant"] = {10878, 824},  
7165 ["nges"] = {10878, 824},  
7166 ["lesdot"] = 10879,  
7167 ["gesdot"] = 10880,  
7168 ["lesdoto"] = 10881,  
7169 ["gesdoto"] = 10882,  
7170 ["lesdotor"] = 10883,

```

7171 ["gesdotol"] = 10884,
7172 ["lap"] = 10885,
7173 ["lessapprox"] = 10885,
7174 ["gap"] = 10886,
7175 ["gtrapprox"] = 10886,
7176 ["lne"] = 10887,
7177 ["lneq"] = 10887,
7178 ["gne"] = 10888,
7179 ["gneq"] = 10888,
7180 ["lnap"] = 10889,
7181 ["lnapprox"] = 10889,
7182 ["gnap"] = 10890,
7183 ["gnapprox"] = 10890,
7184 ["lEg"] = 10891,
7185 ["lesseqqgtr"] = 10891,
7186 ["gEl"] = 10892,
7187 ["gtreqqless"] = 10892,
7188 ["lsime"] = 10893,
7189 ["gsime"] = 10894,
7190 ["lsimg"] = 10895,
7191 ["gsiml"] = 10896,
7192 ["lgE"] = 10897,
7193 ["glE"] = 10898,
7194 ["lesges"] = 10899,
7195 ["gesles"] = 10900,
7196 ["els"] = 10901,
7197 ["eqslantless"] = 10901,
7198 ["egs"] = 10902,
7199 ["eqslantgtr"] = 10902,
7200 ["elsdot"] = 10903,
7201 ["egsdot"] = 10904,
7202 ["el"] = 10905,
7203 ["eg"] = 10906,
7204 ["siml"] = 10909,
7205 ["simg"] = 10910,
7206 ["simlE"] = 10911,
7207 ["simgE"] = 10912,
7208 ["LessLess"] = 10913,
7209 ["NotNestedLessLess"] = {10913, 824},
7210 ["GreaterGreater"] = 10914,
7211 ["NotNestedGreaterGreater"] = {10914, 824},
7212 ["glj"] = 10916,
7213 ["gla"] = 10917,
7214 ["ltcc"] = 10918,
7215 ["gtcc"] = 10919,
7216 ["lescc"] = 10920,
7217 ["gescc"] = 10921,

```

```

7218 ["smt"] = 10922,
7219 ["lat"] = 10923,
7220 ["smte"] = 10924,
7221 ["smtes"] = {10924, 65024},
7222 ["late"] = 10925,
7223 ["lates"] = {10925, 65024},
7224 ["bumpE"] = 10926,
7225 ["NotPrecedesEqual"] = {10927, 824},
7226 ["PrecedesEqual"] = 10927,
7227 ["npre"] = {10927, 824},
7228 ["npreceq"] = {10927, 824},
7229 ["pre"] = 10927,
7230 ["preceq"] = 10927,
7231 ["NotSucceedsEqual"] = {10928, 824},
7232 ["SucceedsEqual"] = 10928,
7233 ["nsce"] = {10928, 824},
7234 ["nsucceq"] = {10928, 824},
7235 ["sce"] = 10928,
7236 ["succeq"] = 10928,
7237 ["prE"] = 10931,
7238 ["scE"] = 10932,
7239 ["precneqq"] = 10933,
7240 ["prnE"] = 10933,
7241 ["scnE"] = 10934,
7242 ["succneqq"] = 10934,
7243 ["prap"] = 10935,
7244 ["precapprox"] = 10935,
7245 ["scap"] = 10936,
7246 ["succapprox"] = 10936,
7247 ["precnapprox"] = 10937,
7248 ["prnap"] = 10937,
7249 ["scnap"] = 10938,
7250 ["succnapprox"] = 10938,
7251 ["Pr"] = 10939,
7252 ["Sc"] = 10940,
7253 ["subdot"] = 10941,
7254 ["supdot"] = 10942,
7255 ["subplus"] = 10943,
7256 ["supplus"] = 10944,
7257 ["submult"] = 10945,
7258 ["supmult"] = 10946,
7259 ["subedot"] = 10947,
7260 ["supedot"] = 10948,
7261 ["nsubE"] = {10949, 824},
7262 ["nsubseteqq"] = {10949, 824},
7263 ["subE"] = 10949,
7264 ["subseteqq"] = 10949,

```

```

7265 ["nsupE"] = {10950, 824},
7266 ["nsupseteqq"] = {10950, 824},
7267 ["supE"] = 10950,
7268 ["supseteqq"] = 10950,
7269 ["subsim"] = 10951,
7270 ["supsim"] = 10952,
7271 ["subnE"] = 10955,
7272 ["subsetneqq"] = 10955,
7273 ["varsubsetneqq"] = {10955, 65024},
7274 ["vsubnE"] = {10955, 65024},
7275 ["supnE"] = 10956,
7276 ["supsetneqq"] = 10956,
7277 ["varsupsetneqq"] = {10956, 65024},
7278 ["vsupnE"] = {10956, 65024},
7279 ["csub"] = 10959,
7280 ["csup"] = 10960,
7281 ["csube"] = 10961,
7282 ["csupe"] = 10962,
7283 ["subsup"] = 10963,
7284 ["supsub"] = 10964,
7285 ["subsub"] = 10965,
7286 ["supsup"] = 10966,
7287 ["suphsub"] = 10967,
7288 ["supdsub"] = 10968,
7289 ["forkv"] = 10969,
7290 ["topfork"] = 10970,
7291 ["mlcp"] = 10971,
7292 ["Dashv"] = 10980,
7293 ["DoubleLeftTee"] = 10980,
7294 ["Vdashl"] = 10982,
7295 ["Barv"] = 10983,
7296 ["vBar"] = 10984,
7297 ["vBarv"] = 10985,
7298 ["Vbar"] = 10987,
7299 ["Not"] = 10988,
7300 ["bNot"] = 10989,
7301 ["rnmid"] = 10990,
7302 ["cirmid"] = 10991,
7303 ["midcir"] = 10992,
7304 ["topcir"] = 10993,
7305 ["nhpar"] = 10994,
7306 ["parsim"] = 10995,
7307 ["nparsl"] = {11005, 8421},
7308 ["parsl"] = 11005,
7309 ["fflig"] = 64256,
7310 ["filig"] = 64257,
7311 ["fllig"] = 64258,

```

```

7312 ["ffilig"] = 64259,
7313 ["ffllig"] = 64260,
7314 ["Ascr"] = 119964,
7315 ["Cscr"] = 119966,
7316 ["Dscr"] = 119967,
7317 ["Gscr"] = 119970,
7318 ["Jscr"] = 119973,
7319 ["Kscr"] = 119974,
7320 ["Nscr"] = 119977,
7321 ["Oscr"] = 119978,
7322 ["Pscr"] = 119979,
7323 ["Qscr"] = 119980,
7324 ["Sscr"] = 119982,
7325 ["Tscr"] = 119983,
7326 ["Uscr"] = 119984,
7327 ["Vscr"] = 119985,
7328 ["Wscr"] = 119986,
7329 ["Xscr"] = 119987,
7330 ["Yscr"] = 119988,
7331 ["Zscr"] = 119989,
7332 ["ascr"] = 119990,
7333 ["bscr"] = 119991,
7334 ["cscr"] = 119992,
7335 ["dscr"] = 119993,
7336 ["fscr"] = 119995,
7337 ["hscr"] = 119997,
7338 ["iscr"] = 119998,
7339 ["jscr"] = 119999,
7340 ["kscr"] = 120000,
7341 ["lscr"] = 120001,
7342 ["mscr"] = 120002,
7343 ["nscr"] = 120003,
7344 ["pscr"] = 120005,
7345 ["qscr"] = 120006,
7346 ["rscr"] = 120007,
7347 ["sscr"] = 120008,
7348 ["tscr"] = 120009,
7349 ["uscr"] = 120010,
7350 ["vscr"] = 120011,
7351 ["wscr"] = 120012,
7352 ["xscr"] = 120013,
7353 ["yscr"] = 120014,
7354 ["zscr"] = 120015,
7355 ["Afr"] = 120068,
7356 ["Bfr"] = 120069,
7357 ["Dfr"] = 120071,
7358 ["Efr"] = 120072,

```

```

7359 ["Ffr"] = 120073,
7360 ["Gfr"] = 120074,
7361 ["Jfr"] = 120077,
7362 ["Kfr"] = 120078,
7363 ["Lfr"] = 120079,
7364 ["Mfr"] = 120080,
7365 ["Nfr"] = 120081,
7366 ["Ofr"] = 120082,
7367 ["Pfr"] = 120083,
7368 ["Qfr"] = 120084,
7369 ["Sfr"] = 120086,
7370 ["Tfr"] = 120087,
7371 ["Ufr"] = 120088,
7372 ["Vfr"] = 120089,
7373 ["Wfr"] = 120090,
7374 ["Xfr"] = 120091,
7375 ["Yfr"] = 120092,
7376 ["afr"] = 120094,
7377 ["bfr"] = 120095,
7378 ["cfr"] = 120096,
7379 ["dfr"] = 120097,
7380 ["efr"] = 120098,
7381 ["ffr"] = 120099,
7382 ["gfr"] = 120100,
7383 ["hfr"] = 120101,
7384 ["ifr"] = 120102,
7385 ["jfr"] = 120103,
7386 ["kfr"] = 120104,
7387 ["lfr"] = 120105,
7388 ["mfr"] = 120106,
7389 ["nfr"] = 120107,
7390 ["ofr"] = 120108,
7391 ["pfr"] = 120109,
7392 ["qfr"] = 120110,
7393 ["rfr"] = 120111,
7394 ["sfr"] = 120112,
7395 ["tfr"] = 120113,
7396 ["ufr"] = 120114,
7397 ["vfr"] = 120115,
7398 ["wfr"] = 120116,
7399 ["xfr"] = 120117,
7400 ["yfr"] = 120118,
7401 ["zfr"] = 120119,
7402 ["Aopf"] = 120120,
7403 ["Bopf"] = 120121,
7404 ["Dopf"] = 120123,
7405 ["Eopf"] = 120124,

```

```

7406 ["Fopf"] = 120125,
7407 ["Gopf"] = 120126,
7408 ["Iopf"] = 120128,
7409 ["Jopf"] = 120129,
7410 ["Kopf"] = 120130,
7411 ["Lopf"] = 120131,
7412 ["Mopf"] = 120132,
7413 ["Oopf"] = 120134,
7414 ["Sopf"] = 120138,
7415 ["Topf"] = 120139,
7416 ["Uopf"] = 120140,
7417 ["Vopf"] = 120141,
7418 ["Wopf"] = 120142,
7419 ["Xopf"] = 120143,
7420 ["Yopf"] = 120144,
7421 ["aopf"] = 120146,
7422 ["bopf"] = 120147,
7423 ["copf"] = 120148,
7424 ["dopf"] = 120149,
7425 ["eopf"] = 120150,
7426 ["fopf"] = 120151,
7427 ["gopf"] = 120152,
7428 ["hopf"] = 120153,
7429 ["iopf"] = 120154,
7430 ["jopf"] = 120155,
7431 ["kopf"] = 120156,
7432 ["lopf"] = 120157,
7433 ["mopf"] = 120158,
7434 ["nopf"] = 120159,
7435 ["oopf"] = 120160,
7436 ["popf"] = 120161,
7437 ["qopf"] = 120162,
7438 ["ropf"] = 120163,
7439 ["sopf"] = 120164,
7440 ["topf"] = 120165,
7441 ["uopf"] = 120166,
7442 ["vopf"] = 120167,
7443 ["wopf"] = 120168,
7444 ["xopf"] = 120169,
7445 ["yopf"] = 120170,
7446 ["zopf"] = 120171,
7447 }

```

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

```

7448 function entities.dec_entity(s)
7449 local n = tonumber(s)

```

```

7450 if n == nil then
7451 return "&#" .. s .. ";" -- fallback for unknown entities
7452 end
7453 return utf8.char(n)
7454 end

```

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

```

7455 function entities.hex_entity(s)
7456 local n = tonumber("0x"..s)
7457 if n == nil then
7458 return "&#x" .. s .. ";" -- fallback for unknown entities
7459 end
7460 return utf8.char(n)
7461 end

```

Given a captured character `x` and a string `s` of hexadecimal digits, the `entities.hex_entity_with_x_char` returns the corresponding UTF8-encoded Unicode codepoint or fallback with the `x` character.

```

7462 function entities.hex_entity_with_x_char(x, s)
7463 local n = tonumber("0x"..s)
7464 if n == nil then
7465 return "&#" .. x .. s .. ";" -- fallback for unknown entities
7466 end
7467 return utf8.char(n)
7468 end

```

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

```

7469 function entities.char_entity(s)
7470 local code_points = character_entities[s]
7471 if code_points == nil then
7472 return "&" .. s .. ";"
7473 end
7474 if type(code_points) ~= 'table' then
7475 code_points = {code_points}
7476 end
7477 local char_table = {}
7478 for _, code_point in ipairs(code_points) do
7479 table.insert(char_table, utf8.char(code_point))
7480 end
7481 return table.concat(char_table)
7482 end

```

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

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X,



L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Luna-mark Lua module.

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

```
7483 M.writer = {}
```

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

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

```
7484 local parsers
7485 function M.writer.new(options)
7486 local self = {}
```

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

```
7487 self.options = options
```

Define `writer->flatten_inlines`, which indicates whether or not the writer should produce raw text rather than text in the output format for inline elements. The `writer->flatten_inlines` member variable is mutable.

```
7488 self.flatten_inlines = false
```

### 3.1.4.1 Slicing

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

```
7489 local slice_specifiers = {}
7490 for specifier in options.slice:gmatch("[^%s]+") do
7491 table.insert(slice_specifiers, specifier)
7492 end
7493
7494 if #slice_specifiers == 2 then
7495 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
7496 local slice_begin_type = self.slice_begin:sub(1, 1)
7497 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
7498 self.slice_begin = "^" .. self.slice_begin
7499 end
7500 local slice_end_type = self.slice_end:sub(1, 1)
7501 if slice_end_type ~= "^" and slice_end_type ~= "$" then
7502 self.slice_end = "$" .. self.slice_end
```

```

7503 end
7504 elseif #slice_specifiers == 1 then
7505 self.slice_begin = "^" .. slice_specifiers[1]
7506 self.slice_end = "$" .. slice_specifiers[1]
7507 end
7508
7509 self.slice_begin_type = self.slice_begin:sub(1, 1)
7510 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
7511 self.slice_end_type = self.slice_end:sub(1, 1)
7512 self.slice_end_identifier = self.slice_end:sub(2) or ""
7513
7514 if self.slice_begin == "^" and self.slice_end ~= "^" then
7515 self.is_writing = true
7516 else
7517 self.is_writing = false
7518 end

```

### 3.1.4.2 Basic Formatter Variables and Functions

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

```

7519 self.space = " "

```

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

```

7520 self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

7521 function self.plain(s)
7522 return s
7523 end

```

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

```

7524 function self.paragraph(s)
7525 if not self.is_writing then return "" end
7526 return s
7527 end

```

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

```

7528 self.interblocksep_text = "\\markdownRendererInterblockSeparator\\n{}"
7529 function self.interblocksep()
7530 if not self.is_writing then return "" end
7531 return self.interblocksep_text
7532 end

```

Define `writer->paragraphsep` as the output format of a paragraph separator. Users can use more than one blank line to delimit two blocks to indicate the end of a series of blocks that make up a paragraph. This produces a paragraph separator instead of an interblock separator.

```

7533 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
7534 function self.paragraphsep()
7535 if not self.is_writing then return "" end
7536 return self.paragraphsep_text
7537 end

```

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

```

7538 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
7539 function self.undosep()
7540 if not self.is_writing then return "" end
7541 return self.undosep_text
7542 end

```

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

```

7543 self.soft_line_break = function()
7544 if self.flatten_inlines then return "\n" end
7545 return "\\markdownRendererSoftLineBreak\n{"
7546 end

```

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

```

7547 self.hard_line_break = function()
7548 if self.flatten_inlines then return "\n" end
7549 return "\\markdownRendererHardLineBreak\n{"
7550 end

```

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

```

7551 self.ellipsis = "\\markdownRendererEllipsis{"

```

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

```

7552 function self.thematic_break()
7553 if not self.is_writing then return "" end
7554 return "\\markdownRendererThematicBreak{"
7555 end

```

### 3.1.4.3 Escaping Special Characters

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.

```

7556 self.escaped_uri_chars = {
7557 [{""] = "\\markdownRendererLeftBrace{"},
7558 ["}"] = "\\markdownRendererRightBrace{"},
7559 ["\\"] = "\\markdownRendererBackslash{"},
7560 ["\r"] = " ",
7561 ["\n"] = " ",
7562 }
7563 self.escaped_minimal_strings = {
7564 ["^"] = "\\markdownRendererCircumflex"

```

```

7565 .. "\\markdownRendererCircumflex ",
7566 ["☒"] = "\\markdownRendererTickedBox{}",
7567 ["☐"] = "\\markdownRendererHalfTickedBox{}",
7568 ["□"] = "\\markdownRendererUntickedBox{}",
7569 [entities.hex_entity('FFFD')]
7570 = "\\markdownRendererReplacementCharacter{}",
7571 }

```

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

```

7572 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
7573 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

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

```

7574 self.escaped_chars = {
7575 ["{"] = "\\markdownRendererLeftBrace{}",
7576 ["}"] = "\\markdownRendererRightBrace{}",
7577 ["%"] = "\\markdownRendererPercentSign{}",
7578 ["\\"] = "\\markdownRendererBackslash{}",
7579 ["#"] = "\\markdownRendererHash{}",
7580 ["$"] = "\\markdownRendererDollarSign{}",
7581 ["&"] = "\\markdownRendererAmpersand{}",
7582 ["_"] = "\\markdownRendererUnderscore{}",
7583 ["^"] = "\\markdownRendererCircumflex{}",
7584 ["~"] = "\\markdownRendererTilde{}",
7585 ["|"] = "\\markdownRendererPipe{}",
7586 [entities.hex_entity('0000')]
7587 = "\\markdownRendererReplacementCharacter{}",
7588 }

```

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

```

7589 local function create_escaper(char_escapes, string_escapes)
7590 local escape = util.escaper(char_escapes, string_escapes)
7591 return function(s)
7592 if self.flatten_inlines then return s end
7593 return escape(s)
7594 end
7595 end
7596 local escape_typographic_text = create_escaper(
7597 self.escaped_chars, self.escaped_strings)
7598 local escape_programmatic_text = create_escaper(
7599 self.escaped_uri_chars, self.escaped_minimal_strings)
7600 local escape_minimal = create_escaper(
7601 {}, self.escaped_minimal_strings)

```

Define the following semantic aliases for the escaper functions:

- `writer->escape` transforms a text string that should always be made printable.
- `writer->string` transforms a text string that should be made printable only when the `hybrid` Lua option is disabled. When `hybrid` is enabled, the text string should be kept as-is.
- `writer->math` transforms a math span.
- `writer->identifier` transforms an input programmatic identifier.
- `writer->uri` transforms an input URI.
- `writer->infostring` transforms a fence code infostring.

```
7602 self.escape = escape_typographic_text
7603 self.math = escape_minimal
7604 if options.hybrid then
7605 self.identifier = escape_minimal
7606 self.string = escape_minimal
7607 self.uri = escape_minimal
7608 self.infostring = escape_minimal
7609 else
7610 self.identifier = escape_programmatic_text
7611 self.string = escape_typographic_text
7612 self.uri = escape_programmatic_text
7613 self.infostring = escape_programmatic_text
7614 end
```

#### 3.1.4.4 Formatters of Warnings and Errors

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

```
7615 function self.warning(t, m)
7616 return {"\\markdownRendererWarning{", self.escape(t), "}{" ,
7617 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7618 escape_minimal(m or ""), "}"}
7619 end
```

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

```
7620 function self.error(t, m)
7621 return {"\\markdownRendererError{", self.escape(t), "}{" ,
7622 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7623 escape_minimal(m or ""), "}"}
7624 end
```

#### 3.1.4.5 Formatter of Code Spans

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

```

7625 function self.code(s, attributes)
7626 if self.flatten_inlines then return s end
7627 local buf = {}
7628 if attributes ~= nil then
7629 table.insert(buf,
7630 "\\markdownRendererCodeSpanAttributeContextBegin\n")
7631 table.insert(buf, self.attributes(attributes))
7632 end
7633 table.insert(buf,
7634 {"\\markdownRendererCodeSpan{", self.escape(s), "}"}))
7635 if attributes ~= nil then
7636 table.insert(buf,
7637 "\\markdownRendererCodeSpanAttributeContextEnd{")
7638 end
7639 return buf
7640 end

```

#### 3.1.4.6 Formatter of Hyperlinks

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.

```

7641 function self.link(lab, src, tit, attributes)
7642 if self.flatten_inlines then return lab end
7643 local buf = {}
7644 if attributes ~= nil then
7645 table.insert(buf,
7646 "\\markdownRendererLinkAttributeContextBegin\n")
7647 table.insert(buf, self.attributes(attributes))
7648 end
7649 table.insert(buf, {"\\markdownRendererLink{",lab,"}",
7650 {"",self.escape(src),"}",
7651 {"",self.uri(src),"}",
7652 {"",self.string(tit or ""),"}"}))
7653 if attributes ~= nil then
7654 table.insert(buf,
7655 "\\markdownRendererLinkAttributeContextEnd{")
7656 end
7657 return buf
7658 end

```

#### 3.1.4.7 Formatter of Images

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.

```

7659 function self.image(lab, src, tit, attributes)

```

```

7660 if self.flatten_inlines then return lab end
7661 local buf = {}
7662 if attributes ~= nil then
7663 table.insert(buf,
7664 "\\markdownRendererImageAttributeContextBegin\n")
7665 table.insert(buf, self.attributes(attributes))
7666 end
7667 table.insert(buf, {"\\markdownRendererImage{",lab,"}",
7668 "{",self.string(src),"}",
7669 "{",self.uri(src),"}",
7670 "{",self.string(tit or ""),"}}"})
7671 if attributes ~= nil then
7672 table.insert(buf,
7673 "\\markdownRendererImageAttributeContextEnd{ }")
7674 end
7675 return buf
7676 end

```

#### 3.1.4.8 Formatters of Lists

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.

```

7677 function self.bulletlist(items,tight)
7678 if not self.is_writing then return "" end
7679 local buffer = {}
7680 for _,item in ipairs(items) do
7681 if item ~= "" then
7682 buffer[#buffer + 1] = self.bulletitem(item)
7683 end
7684 end
7685 local contents = util.intersperse(buffer,"\n")
7686 if tight and options.tightLists then
7687 return {"\\markdownRendererULBeginTight\n",contents,
7688 "\n\\markdownRendererULEndTight "}
7689 else
7690 return {"\\markdownRendererULBegin\n",contents,
7691 "\n\\markdownRendererULEnd "}
7692 end
7693 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.

```

7694 function self.bulletitem(s)
7695 return {"\\markdownRendererULItem ",s,
7696 "\\markdownRendererULItemEnd "}
7697 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.

```

7698 function self.orderedlist(items,tight,startnum)
7699 if not self.is_writing then return "" end
7700 local buffer = {}
7701 local num = startnum
7702 for _,item in ipairs(items) do
7703 if item ~= "" then
7704 buffer[#buffer + 1] = self.ordereditem(item,num)
7705 end
7706 if num ~= nil and item ~= "" then
7707 num = num + 1
7708 end
7709 end
7710 local contents = util.intersperse(buffer,"\n")
7711 if tight and options.tightLists then
7712 return {"\\markdownRendererOlBeginTight\n",contents,
7713 "\\n\\markdownRendererOlEndTight "}
7714 else
7715 return {"\\markdownRendererOlBegin\n",contents,
7716 "\\n\\markdownRendererOlEnd "}
7717 end
7718 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.

```

7719 function self.ordereditem(s,num)
7720 if num ~= nil then
7721 return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
7722 "\\markdownRendererOlItemEnd "}
7723 else
7724 return {"\\markdownRendererOlItem ",s,
7725 "\\markdownRendererOlItemEnd "}
7726 end
7727 end

```

### 3.1.4.9 Formatters of HTML Tags, Elements, and Comments

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.

```

7728 function self.inline_html_comment(contents)
7729 if self.flatten_inlines then return contents end
7730 return {"\\markdownRendererInlineHtmlComment{",contents,""}

```



```
7731 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.

```
7732 function self.inline_html_tag(contents)
7733 if self.flatten_inlines then return contents end
7734 return {"\\markdownRendererInlineHtmlTag{",
7735 self.string(contents),"}"}
7736 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.

```
7737 function self.block_html_element(s)
7738 if not self.is_writing then return "" end
7739 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
7740 return {"\\markdownRendererInputBlockHtmlElement{",name,""}
7741 end
```

#### 3.1.4.10 Formatter of Emphasis

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

```
7742 function self.emphasis(s)
7743 if self.flatten_inlines then return s end
7744 return {"\\markdownRendererEmphasis{",s,""}
7745 end
```

#### 3.1.4.11 Formatter of Strong Emphasis

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

```
7746 function self.strong(s)
7747 if self.flatten_inlines then return s end
7748 return {"\\markdownRendererStrongEmphasis{",s,""}
7749 end
```

#### 3.1.4.12 Formatter of Tickboxes

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

```
7750 function self.tickbox(f)
7751 if f == 1.0 then
7752 return "☒ "
7753 elseif f == 0.0 then
7754 return "☐ "
7755 else
```

```

7756 return "□ "
7757 end
7758 end

```

### 3.1.4.13 Formatter of Blockquotes

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

```

7759 function self.blockquote(s)
7760 if not self.is_writing then return "" end
7761 return {"\\markdownRendererBlockQuoteBegin\n",s,
7762 "\\markdownRendererBlockQuoteEnd "}
7763 end

```

### 3.1.4.14 Formatter of Code Blocks

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

```

7764 function self.verbatim(s)
7765 if not self.is_writing then return "" end
7766 s = s:gsub("\n$", "")
7767 local name = util.cache_verbatim(options.cacheDir, s)
7768 return {"\\markdownRendererInputVerbatim{" ,name,"}"}
7769 end

```

### 3.1.4.15 Formatter of Documents

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

```

7770 function self.document(d)
7771 local buf = {"\\markdownRendererDocumentBegin\n"}
7772
7773 -- warn against the `hybrid` option
7774 if options.hybrid then
7775 local text = "The `hybrid` option has been soft-deprecated."
7776 local more = "Consider using one of the following better options "
7777 .. "for mixing TeX and markdown: `contentBlocks`, "
7778 .. "`rawAttribute`, `texComments`, `texMathDollars`, "
7779 .. "`texMathSingleBackslash`, and "
7780 .. "`texMathDoubleBackslash`. "
7781 .. "For more information, see the user manual at "
7782 .. "<https://witiko.github.io/markdown/>."
7783 table.insert(buf, self.warning(text, more))
7784 end
7785
7786 -- insert the text of the document
7787 table.insert(buf, d)

```

```

7788
7789 -- pop all attributes
7790 table.insert(buf, self.pop_attributes())
7791
7792 table.insert(buf, "\\markdownRendererDocumentEnd")
7793
7794 return buf
7795 end

```

### 3.1.4.16 Formatter of Attributes

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

```

7796 local seen_identifiers = {}
7797 local key_value_regex = "([~=]+)%s*=%s*(.*)"
7798 local function normalize_attributes(attributes, auto_identifiers)
7799 -- normalize attributes
7800 local normalized_attributes = {}
7801 local has_explicit_identifiers = false
7802 local key, value
7803 for _, attribute in ipairs(attributes or {}) do
7804 if attribute:sub(1, 1) == "#" then
7805 table.insert(normalized_attributes, attribute)
7806 has_explicit_identifiers = true
7807 seen_identifiers[attribute:sub(2)] = true
7808 elseif attribute:sub(1, 1) == "." then
7809 table.insert(normalized_attributes, attribute)
7810 else
7811 key, value = attribute:match(key_value_regex)
7812 if key:lower() == "id" then
7813 table.insert(normalized_attributes, "#" .. value)
7814 elseif key:lower() == "class" then
7815 local classes = {}
7816 for class in value:gmatch("%S+") do
7817 table.insert(classes, class)
7818 end
7819 table.sort(classes)
7820 for _, class in ipairs(classes) do
7821 table.insert(normalized_attributes, "." .. class)
7822 end
7823 else
7824 table.insert(normalized_attributes, attribute)
7825 end
7826 end
7827 end
7828
7829 -- if no explicit identifiers exist, add auto identifiers

```

```

7830 if not has_explicit_identifiers and auto_identifiers ~= nil then
7831 local seen_auto_identifiers = {}
7832 for _, auto_identifier in ipairs(auto_identifiers) do
7833 if seen_auto_identifiers[auto_identifier] == nil then
7834 seen_auto_identifiers[auto_identifier] = true
7835 if seen_identifiers[auto_identifier] == nil then
7836 seen_identifiers[auto_identifier] = true
7837 table.insert(normalized_attributes,
7838 "#" .. auto_identifier)
7839 else
7840 local auto_identifier_number = 1
7841 while true do
7842 local numbered_auto_identifier = auto_identifier .. "-"
7843 .. auto_identifier_number
7844 if seen_identifiers[numbered_auto_identifier] == nil then
7845 seen_identifiers[numbered_auto_identifier] = true
7846 table.insert(normalized_attributes,
7847 "#" .. numbered_auto_identifier)
7848 break
7849 end
7850 auto_identifier_number = auto_identifier_number + 1
7851 end
7852 end
7853 end
7854 end
7855 end
7856
7857 -- sort and deduplicate normalized attributes
7858 table.sort(normalized_attributes)
7859 local seen_normalized_attributes = {}
7860 local deduplicated_normalized_attributes = {}
7861 for _, attribute in ipairs(normalized_attributes) do
7862 if seen_normalized_attributes[attribute] == nil then
7863 seen_normalized_attributes[attribute] = true
7864 table.insert(deduplicated_normalized_attributes, attribute)
7865 end
7866 end
7867
7868 return deduplicated_normalized_attributes
7869 end
7870
7871 function self.attributes(attributes, should_normalize_attributes)
7872 local normalized_attributes
7873 if should_normalize_attributes == false then
7874 normalized_attributes = attributes
7875 else
7876 normalized_attributes = normalize_attributes(attributes)

```

```

7877 end
7878
7879 local buf = {}
7880 local key, value
7881 for _, attribute in ipairs(normalized_attributes) do
7882 if attribute:sub(1, 1) == "#" then
7883 table.insert(buf, {"\\markdownRenderAttributeIdentifier{",
7884 attribute:sub(2), "}"}))
7885 elseif attribute:sub(1, 1) == "." then
7886 table.insert(buf, {"\\markdownRendererAttributeName{",
7887 attribute:sub(2), "}"}))
7888 else
7889 key, value = attribute:match(key_value_regex)
7890 table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
7891 key, "{", value, "}"}))
7892 end
7893 end
7894
7895 return buf
7896 end

```

#### 3.1.4.17 Tracking Active Attributes

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

```

7897 self.active_attributes = {}

```

Define `writer->attribute_type_levels` as a hash table that maps attribute types to the number of attributes of said type in `writer->active_attributes`.

```

7898 self.attribute_type_levels = {}
7899 setmetatable(self.attribute_type_levels,
7900 { __index = function() return 0 end })

```

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```

7901 local function apply_attributes()
7902 local buf = {}
7903 for i = 1, #self.active_attributes do
7904 local start_output = self.active_attributes[i][3]
7905 if start_output ~= nil then
7906 table.insert(buf, start_output)
7907 end
7908 end
7909 return buf
7910 end
7911
7912 local function tear_down_attributes()

```

```

7913 local buf = {}
7914 for i = #self.active_attributes, 1, -1 do
7915 local end_output = self.active_attributes[i][4]
7916 if end_output ~= nil then
7917 table.insert(buf, end_output)
7918 end
7919 end
7920 return buf
7921 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.

```

7922 function self.push_attributes(attribute_type, attributes,
7923 start_output, end_output)
7924 local attribute_type_level
7925 = self.attribute_type_levels[attribute_type]
7926 self.attribute_type_levels[attribute_type]
7927 = attribute_type_level + 1
7928
7929 -- index attributes in a hash table for easy lookup
7930 attributes = attributes or {}
7931 for i = 1, #attributes do
7932 attributes[attributes[i]] = true
7933 end
7934
7935 local buf = {}
7936 -- handle slicing
7937 if attributes["#" .. self.slice_end_identifier] ~= nil and
7938 self.slice_end_type == "^" then
7939 if self.is_writing then
7940 table.insert(buf, self.undosep())
7941 table.insert(buf, tear_down_attributes())
7942 end
7943 self.is_writing = false
7944 end
7945 if attributes["#" .. self.slice_begin_identifier] ~= nil and
7946 self.slice_begin_type == "^" then
7947 table.insert(buf, apply_attributes())
7948 self.is_writing = true
7949 end
7950 if self.is_writing and start_output ~= nil then
7951 table.insert(buf, start_output)
7952 end
7953 table.insert(self.active_attributes,

```

```

7954 {attribute_type, attributes,
7955 start_output, end_output})
7956 return buf
7957 end
7958

```

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`).

```

7959 function self.pop_attributes(attribute_type)
7960 local buf = {}
7961 -- pop attributes until we find attributes of correct type
7962 -- or until no attributes remain
7963 local current_attribute_type = false
7964 while current_attribute_type ~= attribute_type and
7965 #self.active_attributes > 0 do
7966 local attributes, _, end_output
7967 current_attribute_type, attributes, _, end_output = table.unpack(
7968 self.active_attributes[#self.active_attributes])
7969 local attribute_type_level
7970 = self.attribute_type_levels[current_attribute_type]
7971 self.attribute_type_levels[current_attribute_type]
7972 = attribute_type_level - 1
7973 if self.is_writing and end_output ~= nil then
7974 table.insert(buf, end_output)
7975 end
7976 table.remove(self.active_attributes, #self.active_attributes)
7977 -- handle slicing
7978 if attributes["#" .. self.slice_end_identifier] ~= nil
7979 and self.slice_end_type == "$" then
7980 if self.is_writing then
7981 table.insert(buf, self.undosep())
7982 table.insert(buf, tear_down_attributes())
7983 end
7984 self.is_writing = false
7985 end
7986 if attributes["#" .. self.slice_begin_identifier] ~= nil and
7987 self.slice_begin_type == "$" then
7988 self.is_writing = true
7989 table.insert(buf, apply_attributes())
7990 end
7991 end
7992 return buf
7993 end

```

### 3.1.4.18 Automatically Generated Identifiers for Headings

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

```
7994 local function create_auto_identifier(s)
7995 local buffer = {}
7996 local prev_space = false
7997 local letter_found = false
7998 local normalized_s = s
7999 if not options.unicodeNormalization
8000 or options.unicodeNormalizationForm ~= "nfc" then
8001 normalized_s = util.normalize(normalized_s, "nfc")
8002 end
8003
8004 for _, code in utf8.codes(normalized_s) do
8005 local char = utf8.char(code)
8006
8007 -- Remove everything up to the first letter.
8008 if not letter_found then
8009 local is_letter = lpeg.match(
8010 parsers.unicode.following_alpha,
8011 char
8012)
8013 if is_letter then
8014 letter_found = true
8015 else
8016 goto continue
8017 end
8018 end
8019
8020 -- Remove all non-alphanumeric characters, except underscores,
8021 -- hyphens, and periods.
8022 if not lpeg.match(
8023 (parsers.underscore
8024 + parsers.dash
8025 + parsers.period
8026 + parsers.unicode.following_word
8027 + parsers.unicode.following_whitespace),
8028 char
8029) then
8030 goto continue
8031 end
8032
8033 -- Replace all spaces and newlines with hyphens.
8034 if lpeg.match(
8035 (parsers.newline
8036 + parsers.unicode.following_whitespace),
8037 char
```



```

8038) then
8039 char = "-"
8040 if prev_space then
8041 goto continue
8042 else
8043 prev_space = true
8044 end
8045 else
8046 -- Case-fold all alphabetic characters.
8047 local form = nil
8048 if options.unicodeNormalization then
8049 form = options.unicodeNormalizationForm
8050 end
8051 char = util.casefold(char, form)
8052 prev_space = false
8053 end
8054
8055 table.insert(buffer, char)
8056
8057 ::continue::
8058 end
8059
8060 if prev_space then
8061 table.remove(buffer)
8062 end
8063
8064 local identifier = #buffer == 0 and "section"
8065 or table.concat(buffer, "")
8066 return identifier
8067 end

```

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

```

8068 local function create_gfm_auto_identifier(s)
8069 local buffer = {}
8070 local prev_space = false
8071 local letter_found = false
8072 local normalized_s = s
8073 if not options.unicodeNormalization
8074 or options.unicodeNormalizationForm ~= "nfc" then
8075 normalized_s = util.normalize(normalized_s, "nfc")
8076 end
8077
8078 for _, code in utf8.codes(normalized_s) do
8079 local char = utf8.char(code)
8080
8081 -- Remove everything up to the first non-space.

```

```

8082 if not letter_found then
8083 local is_letter = not lpeg.match(
8084 parsers.unicode.following_whitespace,
8085 char
8086)
8087 if is_letter then
8088 letter_found = true
8089 else
8090 goto continue
8091 end
8092 end
8093
8094 -- Remove all non-alphanumeric characters, except underscores
8095 -- and hyphens.
8096 if not lpeg.match(
8097 (parsers.underscore
8098 + parsers.dash
8099 + parsers.unicode.following_word
8100 + parsers.unicode.following_whitespace),
8101 char
8102) then
8103 prev_space = false
8104 goto continue
8105 end
8106
8107 -- Replace all spaces and newlines with hyphens.
8108 if lpeg.match(
8109 (parsers.newline
8110 + parsers.unicode.following_whitespace),
8111 char
8112) then
8113 char = "-"
8114 if prev_space then
8115 goto continue
8116 else
8117 prev_space = true
8118 end
8119 else
8120 -- Case-fold all alphabetic characters.
8121 local form = nil
8122 if options.unicodeNormalization then
8123 form = options.unicodeNormalizationForm
8124 end
8125 char = util.casefold(char, form)
8126 prev_space = false
8127 end
8128

```

```

8129 table.insert(buffer, char)
8130
8131 ::continue::
8132 end
8133
8134 if prev_space then
8135 table.remove(buffer)
8136 end
8137
8138 local identifier = #buffer == 0 and "section"
8139 or table.concat(buffer, "")
8140 return identifier
8141 end

```

### 3.1.4.19 Formatter of Headings

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

```

8142 self.secbegin_text = "\\markdownRendererSectionBegin\n"
8143 self.secend_text = "\n\\markdownRendererSectionEnd "
8144 function self.heading(s, level, attributes)
8145 local buf = {}
8146 local flat_text, inlines = table.unpack(s)
8147
8148 -- push empty attributes for implied sections
8149 while self.attribute_type_levels["heading"] < level - 1 do
8150 table.insert(buf,
8151 self.push_attributes("heading",
8152 nil,
8153 self.secbegin_text,
8154 self.secend_text))
8155 end
8156
8157 -- pop attributes for sections that have ended
8158 while self.attribute_type_levels["heading"] >= level do
8159 table.insert(buf, self.pop_attributes("heading"))
8160 end
8161
8162 -- construct attributes for the new section
8163 local auto_identifiers = {}
8164 if self.options.autoIdentifiers then
8165 table.insert(auto_identifiers, create_auto_identifier(flat_text))
8166 end
8167 if self.options.gfmAutoIdentifiers then
8168 table.insert(auto_identifiers,
8169 create_gfm_auto_identifier(flat_text))
8170 end
8171 end

```

```

8171 local normalized_attributes = normalize_attributes(attributes,
8172 auto_identifiers)
8173
8174 -- push attributes for the new section
8175 local start_output = {}
8176 local end_output = {}
8177 table.insert(start_output, self.secbegin_text)
8178 table.insert(end_output, self.secend_text)
8179
8180 table.insert(buf, self.push_attributes("heading",
8181 normalized_attributes,
8182 start_output,
8183 end_output))
8184 assert(self.attribute_type_levels["heading"] == level)
8185
8186 -- render the heading and its attributes
8187 if self.is_writing and #normalized_attributes > 0 then
8188 table.insert(buf,
8189 "\\markdownRendererHeaderAttributeContextBegin\n")
8190 table.insert(buf, self.attributes(normalized_attributes, false))
8191 end
8192
8193 local cmd
8194 level = level + options.shiftHeadings
8195 if level <= 1 then
8196 cmd = "\\markdownRendererHeadingOne"
8197 elseif level == 2 then
8198 cmd = "\\markdownRendererHeadingTwo"
8199 elseif level == 3 then
8200 cmd = "\\markdownRendererHeadingThree"
8201 elseif level == 4 then
8202 cmd = "\\markdownRendererHeadingFour"
8203 elseif level == 5 then
8204 cmd = "\\markdownRendererHeadingFive"
8205 elseif level >= 6 then
8206 cmd = "\\markdownRendererHeadingSix"
8207 else
8208 cmd = ""
8209 end
8210 if self.is_writing then
8211 table.insert(buf, {cmd, "{", inlines, "}"})
8212 end
8213
8214 if self.is_writing and #normalized_attributes > 0 then
8215 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
8216 end
8217

```

```

8218 return buf
8219 end

```

### 3.1.4.20 Managing State and Deferred Writer Calls

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.

```

8220 function self.get_state()
8221 return {
8222 is_writing=self.is_writing,
8223 flatten_inlines=self.flatten_inlines,
8224 active_attributes={table.unpack(self.active_attributes)},
8225 }
8226 end

```

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

```

8227 function self.set_state(s)
8228 local previous_state = self.get_state()
8229 for key, value in pairs(s) do
8230 self[key] = value
8231 end
8232 return previous_state
8233 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`.

```

8234 function self.defer_call(f)
8235 local previous_state = self.get_state()
8236 return function(...)
8237 local state = self.set_state(previous_state)
8238 local return_value = f(...)
8239 self.set_state(state)
8240 return return_value
8241 end
8242 end
8243
8244 return self
8245 end

```

### 3.1.5 Parsers

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

```

8246 parsers = {}

```

### 3.1.5.1 Basic Parsers

```
8247 parsers.percent = P("%")
8248 parsers.at = P("@")
8249 parsers.comma = P(",")
8250 parsers.asterisk = P("*")
8251 parsers.dash = P("-")
8252 parsers.plus = P("+")
8253 parsers.underscore = P("_")
8254 parsers.period = P(".")
8255 parsers.hash = P("#")
8256 parsers.dollar = P("$")
8257 parsers.ampersand = P("&")
8258 parsers.backtick = P("`")
8259 parsers.less = P("<")
8260 parsers.more = P(">")
8261 parsers.space = P(" ")
8262 parsers.squote = P("'")
8263 parsers.dquote = P('"')
8264 parsers.lparent = P("(")
8265 parsers.rparent = P(")")
8266 parsers.lbracket = P("[")
8267 parsers.rbracket = P("]")
8268 parsers.lbrace = P("{")
8269 parsers.rbrace = P("}")
8270 parsers.circumflex = P("^")
8271 parsers.slash = P("/")
8272 parsers.equal = P("=")
8273 parsers.colon = P(":")
8274 parsers.semicolon = P(";")
8275 parsers.exclamation = P("!")
8276 parsers.pipe = P("|")
8277 parsers.tilde = P("~")
8278 parsers.backslash = P("\\")
8279 parsers.tab = P("\t")
8280 parsers.newline = P("\n")
8281
8282 parsers.digit = R("09")
8283 parsers.hexdigit = R("09", "af", "AF")
8284 parsers.letter = R("AZ", "az")
8285 parsers.alphanumeric = R("AZ", "az", "09")
8286 parsers.keyword = parsers.letter
8287 * (parsers.alphanumeric + parsers.dash)^0
8288
8289 parsers.doubleasterisks = P("**")
8290 parsers.doubleunderscores = P("__")
8291 parsers.doubletildes = P("~~")
8292 parsers.fourspace = P(" ")
```

```

8293
8294 parsers.any = P(1)
8295 parsers.succeed = P(true)
8296 parsers.fail = P(false)
8297
8298 parsers.internal_punctuation = S(":,;.?"")
8299 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
8300
8301 parsers.escapable = parsers.ascii_punctuation
8302 parsers.anyescaped = parsers.backslash / ""
8303 * parsers.escapable
8304 + parsers.any
8305
8306 parsers.spacechar = S("\t ")
8307 parsers.spacing = S(" \n\r\t")
8308 parsers.nonspacechar = parsers.any - parsers.spacing
8309 parsers.optionalspace = parsers.spacechar^0
8310
8311 parsers.normalchar = parsers.any - (V("SpecialChar")
8312 + parsers.spacing)
8313 parsers.eof = -parsers.any
8314 parsers.nonindentspace = parsers.space^-3 * - parsers.spacechar
8315 parsers.indent = parsers.space^-3 * parsers.tab
8316 + parsers.fourspace / ""
8317 parsers.linechar = P(1 - parsers.newline)
8318
8319 parsers.blankline = parsers.optionalspace
8320 * parsers.newline / "\n"
8321 parsers.blanklines = parsers.blankline^0
8322 parsers.skipblanklines = (parsers.optionalspace
8323 * parsers.newline)^0
8324 parsers.indentedline = parsers.indent / ""
8325 * C(parsers.linechar^1
8326 * parsers.newline^1)
8327 parsers.optionallyindentedline = parsers.indent^-1 / ""
8328 * C(parsers.linechar^1
8329 * parsers.newline^1)
8330 parsers.sp = parsers.spacing^0
8331 parsers.spnl = parsers.optionalspace
8332 * (parsers.newline
8333 * parsers.optionalspace)^-1
8334 parsers.line = parsers.linechar^0 * parsers.newline
8335 parsers.nonemptyline = parsers.line - parsers.blankline
8336

```

### 3.1.5.2 Parsers for Unicode Character Classes and Categories

We define high-level parsers in table `parsers.unicode` based on the low-level parsers in `unicode_data.categories`, defined in Section 3.1.1.5. Unlike the low-level parsers, the high-level parsers are invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
8337 parsers.unicode = {}
8338 parsers.unicode.preceding_punctuation = parsers.fail
8339 parsers.unicode.following_punctuation = parsers.fail
8340 parsers.unicode.following_alpha = parsers.fail
8341 parsers.unicode.following_numeric = parsers.fail
8342 parsers.unicode.following_word = parsers.fail
8343 parsers.unicode.preceding_whitespace = parsers.fail
8344 parsers.unicode.following_whitespace = parsers.fail
8345 for n = 1, 4 do
```

For punctuation, accept any characters from Unicode categories P (punctuation) and S (symbol), as mandated by the CommonMark standard<sup>36</sup>.

```
8346 local punctuation_of_length_n
8347 = unicode_data.categories.P[n]
8348 + unicode_data.categories.S[n]
8349 parsers.unicode.preceding_punctuation
8350 = parsers.unicode.preceding_punctuation
8351 + B(punctuation_of_length_n)
8352 parsers.unicode.following_punctuation
8353 = parsers.unicode.following_punctuation
8354 + #punctuation_of_length_n
```

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

```
8355 local alpha_of_length_n = unicode_data.categories.L[n]
8356 parsers.unicode.following_alpha
8357 = parsers.unicode.following_alpha
8358 + alpha_of_length_n
```

For numeric characters, accept any characters from Unicode category N (number), similar to the character class ‘Unicode.

```
8359 local numeric_of_length_n = unicode_data.categories.N[n]
8360 parsers.unicode.following_numeric
8361 = parsers.unicode.following_numeric
8362 + numeric_of_length_n
```

For word characters, accept any characters from Unicode categories L (letter), N (number), and Pc (connector punctuation), similar to the character class ‘

```
8363 local word_of_length_n
8364 = alpha_of_length_n
8365 + numeric_of_length_n
8366 + unicode_data.categories.Pc[n]
```

<sup>36</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.



```

8367 parsers.unicode.following_word
8368 = parsers.unicode.following_word
8369 + word_of_length_n

```

For space characters, accept any characters from Unicode category Z (separator), as well as the ASCII control characters 9 (horizontal tab) through 13 (carriage return), similar to the character class ‘Lua library Selene Unicode.

```

8370 local whitespace_of_length_n = unicode_data.categories.Z[n]
8371 if n == 1 then
8372 whitespace_of_length_n
8373 = whitespace_of_length_n
8374 + R("\t\r")
8375 end
8376 parsers.unicode.preceding_whitespace
8377 = parsers.unicode.preceding_whitespace
8378 + B(whitespace_of_length_n)
8379 parsers.unicode.following_whitespace
8380 = parsers.unicode.following_whitespace
8381 + #whitespace_of_length_n
8382 end

```

### 3.1.5.3 Parsers Used for Indentation

```

8383
8384 parsers.leader = parsers.space~-3
8385

```

Check if a trail exists and is non-empty in the indent table [indent\\_table](#).

```

8386 local function has_trail(indent_table)
8387 return indent_table ~= nil and
8388 indent_table.trail ~= nil and
8389 next(indent_table.trail) ~= nil
8390 end
8391

```

Check if indent table [indent\\_table](#) has any indents.

```

8392 local function has_indents(indent_table)
8393 return indent_table ~= nil and
8394 indent_table.indents ~= nil and
8395 next(indent_table.indents) ~= nil
8396 end
8397

```

Add a trail [trail\\_info](#) to the indent table [indent\\_table](#).

```

8398 local function add_trail(indent_table, trail_info)
8399 indent_table.trail = trail_info
8400 return indent_table
8401 end
8402

```

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

```
8403 local function remove_trail(indent_table)
8404 indent_table.trail = nil
8405 return indent_table
8406 end
8407
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
8408 local function update_indent_table(indent_table, new_indent, add)
8409 indent_table = remove_trail(indent_table)
8410
8411 if not has_indents(indent_table) then
8412 indent_table.indents = {}
8413 end
8414
8415
8416 if add then
8417 indent_table.indents[#indent_table.indents + 1] = new_indent
8418 else
8419 if indent_table.indents[#indent_table.indents].name
8420 == new_indent.name then
8421 indent_table.indents[#indent_table.indents] = nil
8422 end
8423 end
8424
8425 return indent_table
8426 end
8427
```

Remove an indent by its name `name`.

```
8428 local function remove_indent(name)
8429 local remove_indent_level =
8430 function(s, i, indent_table) -- luacheck: ignore s i
8431 indent_table = update_indent_table(indent_table, {name=name},
8432 false)
8433 return true, indent_table
8434 end
8435
8436 return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
8437 end
8438
```

Process the spacing of a string of spaces and tabs `spacing` with preceding indent width from the start of the line `indent` and strip up to `left_strip_length` spaces. Return the remainder `remainder` and whether there is enough spaces to produce a code `is_code`. Return how many spaces were stripped, as well as if the minimum was met `is_minimum` and what remainder it left `minimum_remainder`.

```
8439 local function process_starter_spacing(indent, spacing,
```

```

8440 minimum, left_strip_length)
8441 left_strip_length = left_strip_length or 0
8442
8443 local count = 0
8444 local tab_value = 4 - (indent) % 4
8445
8446 local code_started, minimum_found = false, false
8447 local code_start, minimum_remainder = "", ""
8448
8449 local left_total_stripped = 0
8450 local full_remainder = ""
8451
8452 if spacing ~= nil then
8453 for i = 1, #spacing do
8454 local character = spacing:sub(i, i)
8455
8456 if character == "\t" then
8457 count = count + tab_value
8458 tab_value = 4
8459 elseif character == " " then
8460 count = count + 1
8461 tab_value = 4 - (1 - tab_value) % 4
8462 end
8463
8464 if (left_strip_length ~= 0) then
8465 local possible_to_strip = math.min(count, left_strip_length)
8466 count = count - possible_to_strip
8467 left_strip_length = left_strip_length - possible_to_strip
8468 left_total_stripped = left_total_stripped + possible_to_strip
8469 else
8470 full_remainder = full_remainder .. character
8471 end
8472
8473 if (minimum_found) then
8474 minimum_remainder = minimum_remainder .. character
8475 elseif (count >= minimum) then
8476 minimum_found = true
8477 minimum_remainder = minimum_remainder
8478 .. string.rep(" ", count - minimum)
8479 end
8480
8481 if (code_started) then
8482 code_start = code_start .. character
8483 elseif (count >= minimum + 4) then
8484 code_started = true
8485 code_start = code_start
8486 .. string.rep(" ", count - (minimum + 4))

```

```

8487 end
8488 end
8489 end
8490
8491 local remainder
8492 if (code_started) then
8493 remainder = code_start
8494 else
8495 remainder = string.rep(" ", count - minimum)
8496 end
8497
8498 local is_minimum = count >= minimum
8499 return {
8500 is_code = code_started,
8501 remainder = remainder,
8502 left_total_stripped = left_total_stripped,
8503 is_minimum = is_minimum,
8504 minimum_remainder = minimum_remainder,
8505 total_length = count,
8506 full_remainder = full_remainder
8507 }
8508 end
8509

```

Count the total width of all indents in the indent table `indent_table`.

```

8510 local function count_indent_tab_level(indent_table)
8511 local count = 0
8512 if not has_indents(indent_table) then
8513 return count
8514 end
8515
8516 for i=1, #indent_table.indents do
8517 count = count + indent_table.indents[i].length
8518 end
8519 return count
8520 end
8521

```

Count the total width of a delimiter `delimiter`.

```

8522 local function total_delimiter_length(delimiter)
8523 local count = 0
8524 if type(delimiter) == "string" then return #delimiter end
8525 for _, value in pairs(delimiter) do
8526 count = count + total_delimiter_length(value)
8527 end
8528 return count
8529 end
8530

```

Process the container starter `starter` of a type `indent_type`. Adjust the width of the indent if the delimiter is followed only by whitespaces `is_blank`.

```
8531 local function process_starter_indent(_, _, indent_table, starter,
8532 is_blank, indent_type, breakable)
8533 local last_trail = starter[1]
8534 local delimiter = starter[2]
8535 local raw_new_trail = starter[3]
8536
8537 if indent_type == "bq" and not breakable then
8538 indent_table.ignore_blockquote_blank = true
8539 end
8540
8541 if has_trail(indent_table) then
8542 local trail = indent_table.trail
8543 if trail.is_code then
8544 return false
8545 end
8546 last_trail = trail.remainder
8547 else
8548 local sp = process_starter_spacing(0, last_trail, 0, 0)
8549
8550 if sp.is_code then
8551 return false
8552 end
8553 last_trail = sp.remainder
8554 end
8555
8556 local preceding_indentation = count_indent_tab_level(indent_table) % 4
8557 local last_trail_length = #last_trail
8558 local delimiter_length = total_delimiter_length(delimiter)
8559
8560 local total_indent_level = preceding_indentation + last_trail_length
8561 + delimiter_length
8562
8563 local sp = {}
8564 if not is_blank then
8565 sp = process_starter_spacing(total_indent_level, raw_new_trail,
8566 0, 1)
8567 end
8568
8569 local del_trail_length = sp.left_total_stripped
8570 if is_blank then
8571 del_trail_length = 1
8572 elseif not sp.is_code then
8573 del_trail_length = del_trail_length + #sp.remainder
8574 end
8575
```

```

8576 local indent_length = last_trail_length + delimiter_length
8577 + del_trail_length
8578 local new_indent_info = {name=indent_type, length=indent_length}
8579
8580 indent_table = update_indent_table(indent_table, new_indent_info,
8581 true)
8582 indent_table = add_trail(indent_table,
8583 {is_code=sp.is_code,
8584 remainder=sp.remainder,
8585 total_length=sp.total_length,
8586 full_remainder=sp.full_remainder})
8587
8588 return true, indent_table
8589 end
8590

```

Return the pattern corresponding with the indent name `name`.

```

8591 local function decode_pattern(name)
8592 local delimiter = parsers.succeed
8593 if name == "bq" then
8594 delimiter = parsers.more
8595 end
8596
8597 return C(parsers.optionalspace) * C(delimiter)
8598 * C(parsers.optionalspace) * Cp()
8599 end
8600

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

8601 local function left_blank_starter(indent_table)
8602 local blank_starter_index
8603
8604 if not has_indents(indent_table) then
8605 return
8606 end
8607
8608 for i = #indent_table.indents,1,-1 do
8609 local value = indent_table.indents[i]
8610 if value.name == "li" then
8611 blank_starter_index = i
8612 else
8613 break
8614 end
8615 end
8616
8617 return blank_starter_index
8618 end

```

8619

Apply the patterns decoded from the indents of the indent table `indent_table` iteratively starting at position `index` of the string `s`. If the `is_optional` mode is selected, match as many patterns as possible, else match all or fail. With the option `is_blank`, the parsing behaves as optional after the position of a blank-only indent has been surpassed.

```
8620 local function traverse_indent(s, i, indent_table, is_optional,
8621 is_blank, current_line_indents)
8622 local new_index = i
8623
8624 local preceding_indentation = 0
8625 local current_trail = {}
8626
8627 local blank_starter = left_blank_starter(indent_table)
8628
8629 if current_line_indents == nil then
8630 current_line_indents = {}
8631 end
8632
8633 for index = 1, #indent_table.indents do
8634 local value = indent_table.indents[index]
8635 local pattern = decode_pattern(value.name)
8636
8637 -- match decoded pattern
8638 local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
8639 if new_indent_info == nil then
8640 local blankline_end = lpeg.match(
8641 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
8642 if is_optional or not indent_table.ignore_blockquote_blank
8643 or not blankline_end then
8644 return is_optional, new_index, current_trail,
8645 current_line_indents
8646 end
8647
8648 return traverse_indent(s, tonumber(blankline_end.pos),
8649 indent_table, is_optional, is_blank,
8650 current_line_indents)
8651 end
8652
8653 local raw_last_trail = new_indent_info[1]
8654 local delimiter = new_indent_info[2]
8655 local raw_new_trail = new_indent_info[3]
8656 local next_index = new_indent_info[4]
8657
8658 local space_only = delimiter == ""
8659
```

```

8660 -- check previous trail
8661 if not space_only and next(current_trail) == nil then
8662 local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
8663 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8664 total_length=sp.total_length,
8665 full_remainder=sp.full_remainder}
8666 end
8667
8668 if next(current_trail) ~= nil then
8669 if not space_only and current_trail.is_code then
8670 return is_optional, new_index, current_trail,
8671 current_line_indents
8672 end
8673 if current_trail.internal_remainder ~= nil then
8674 raw_last_trail = current_trail.internal_remainder
8675 end
8676 end
8677
8678 local raw_last_trail_length = 0
8679 local delimiter_length = 0
8680
8681 if not space_only then
8682 delimiter_length = #delimiter
8683 raw_last_trail_length = #raw_last_trail
8684 end
8685
8686 local total_indent_level = preceding_indentation
8687 + raw_last_trail_length + delimiter_length
8688
8689 local spacing_to_process
8690 local minimum = 0
8691 local left_strip_length = 0
8692
8693 if not space_only then
8694 spacing_to_process = raw_new_trail
8695 left_strip_length = 1
8696 else
8697 spacing_to_process = raw_last_trail
8698 minimum = value.length
8699 end
8700
8701 local sp = process_starter_spacing(total_indent_level,
8702 spacing_to_process, minimum,
8703 left_strip_length)
8704
8705 if space_only and not sp.is_minimum then
8706 return is_optional or (is_blank and blank_starter <= index),

```



```

8707 new_index, current_trail, current_line_indents
8708 end
8709
8710 local indent_length = raw_last_trail_length + delimiter_length
8711 + sp.left_total_stripped
8712
8713 -- update info for the next pattern
8714 if not space_only then
8715 preceding_indentation = preceding_indentation + indent_length
8716 else
8717 preceding_indentation = preceding_indentation + value.length
8718 end
8719
8720 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8721 internal_remainder=sp.minimum_remainder,
8722 total_length=sp.total_length,
8723 full_remainder=sp.full_remainder}
8724
8725 current_line_indents[#current_line_indents + 1] = new_indent_info
8726 new_index = next_index
8727 end
8728
8729 return true, new_index, current_trail, current_line_indents
8730 end
8731

```

Check if a code trail is expected.

```

8732 local function check_trail(expect_code, is_code)
8733 return (expect_code and is_code) or (not expect_code and not is_code)
8734 end
8735

```

Check if the current trail of the [indent\\_table](#) would produce code if it is expected [expect\\_code](#) or it would not if it is not. If there is no trail, process and check the current spacing [spacing](#).

```

8736 local check_trail_joined =
8737 function(s, i, indent_table, -- luacheck: ignore s i
8738 spacing, expect_code, omit_remainder)
8739 local is_code
8740 local remainder
8741
8742 if has_trail(indent_table) then
8743 local trail = indent_table.trail
8744 is_code = trail.is_code
8745 if is_code then
8746 remainder = trail.remainder
8747 else
8748 remainder = trail.full_remainder

```

```

8749 end
8750 else
8751 local sp = process_starter_spacing(0, spacing, 0, 0)
8752 is_code = sp.is_code
8753 if is_code then
8754 remainder = sp.remainder
8755 else
8756 remainder = sp.full_remainder
8757 end
8758 end
8759
8760 local result = check_trail(expect_code, is_code)
8761 if omit_remainder then
8762 return result
8763 end
8764 return result, remainder
8765 end
8766

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

8767 local check_trail_length =
8768 function(s, i, indent_table, -- luacheck: ignore s i
8769 spacing, min, max)
8770 local trail
8771
8772 if has_trail(indent_table) then
8773 trail = indent_table.trail
8774 else
8775 trail = process_starter_spacing(0, spacing, 0, 0)
8776 end
8777
8778 local total_length = trail.total_length
8779 if total_length == nil then
8780 return false
8781 end
8782
8783 return min <= total_length and total_length <= max
8784 end
8785

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

8786 local function check_continuation_indentation(s, i, indent_table,
8787 is_optional, is_blank)
8788 if not has_indents(indent_table) then
8789 return true
8790 end
8791

```

```

8792 local passes, new_index, current_trail, current_line_indents =
8793 traverse_indent(s, i, indent_table, is_optional, is_blank)
8794
8795 if passes then
8796 indent_table.current_line_indents = current_line_indents
8797 indent_table = add_trail(indent_table, current_trail)
8798 return new_index, indent_table
8799 end
8800 return false
8801 end
8802

```

Get name of the last indent from the `indent_table`.

```

8803 local function get_last_indent_name(indent_table)
8804 if has_indents(indent_table) then
8805 return indent_table.indents[#indent_table.indents].name
8806 end
8807 end
8808

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

8809 local function remove_remainder_if_blank(indent_table, remainder)
8810 if get_last_indent_name(indent_table) == "li" then
8811 return ""
8812 end
8813 return remainder
8814 end
8815

```

Take the trail `trail` or create a new one from `spacing` and compare it with the expected `trail_type`. On success return the index `i` and the remainder of the trail.

```

8816 local check_trail_type =
8817 function(s, i, -- luacheck: ignore s i
8818 trail, spacing, trail_type)
8819 if trail == nil then
8820 trail = process_starter_spacing(0, spacing, 0, 0)
8821 end
8822
8823 if trail_type == "non-code" then
8824 return check_trail(false, trail.is_code)
8825 end
8826 if trail_type == "code" then
8827 return check_trail(true, trail.is_code)
8828 end
8829 if trail_type == "full-code" then
8830 if (trail.is_code) then
8831 return i, trail.remainder

```

```

8832 end
8833 return i, ""
8834 end
8835 if trail_type == "full-any" then
8836 return i, trail.internal_remainder
8837 end
8838 end
8839

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

8840 local trail_freezing =
8841 function(s, i, -- luacheck: ignore s i
8842 indent_table, is_freezing)
8843 if is_freezing then
8844 if indent_table.is_trail_frozen then
8845 indent_table.trail = indent_table.frozen_trail
8846 else
8847 indent_table.frozen_trail = indent_table.trail
8848 indent_table.is_trail_frozen = true
8849 end
8850 else
8851 indent_table.frozen_trail = nil
8852 indent_table.is_trail_frozen = false
8853 end
8854 return true, indent_table
8855 end
8856

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line specifically with `is_blank`. Additionally, also directly check the new trail with a type `trail_type`.

```

8857 local check_continuation_indentation_and_trail =
8858 function (s, i, indent_table, is_optional, is_blank, trail_type,
8859 reset_rem, omit_remainder)
8860 if not has_indents(indent_table) then
8861 local spacing, new_index = lpeg.match(C(parsers.spacechar^0)
8862 * Cp(), s, i)
8863 local result, remainder = check_trail_type(s, i,
8864 indent_table.trail, spacing, trail_type)
8865 if remainder == nil then
8866 if result then
8867 return new_index
8868 end
8869 return false
8870 end
8871 if result then
8872 return new_index, remainder
8873 end
8874 end
8875 end
8876

```

```

8874 return false
8875 end
8876
8877 local passes, new_index, current_trail = traverse_indent(s, i,
8878 indent_table, is_optional, is_blank)
8879
8880 if passes then
8881 local spacing
8882 if current_trail == nil then
8883 local newer_spacing, newer_index = lpeg.match(
8884 C(parsers.spacechar^0) * Cp(), s, i)
8885 current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
8886 new_index = newer_index
8887 spacing = newer_spacing
8888 else
8889 spacing = current_trail.remainder
8890 end
8891 local result, remainder = check_trail_type(s, new_index,
8892 current_trail, spacing, trail_type)
8893 if remainder == nil or omit_remainder then
8894 if result then
8895 return new_index
8896 end
8897 return false
8898 end
8899
8900 if is_blank and reset_rem then
8901 remainder = remove_remainder_if_blank(indent_table, remainder)
8902 end
8903 if result then
8904 return new_index, remainder
8905 end
8906 return false
8907 end
8908 return false
8909 end
8910

```

The following patterns check whitespace indentation at the start of a block.

```

8911 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0)
8912 * Cc(false), check_trail_joined)
8913
8914 parsers.check_trail_no_rem = Cmt(Cb("indent_info")
8915 * C(parsers.spacechar^0) * Cc(false)
8916 * Cc(true), check_trail_joined)
8917
8918 parsers.check_code_trail = Cmt(Cb("indent_info")
8919 * C(parsers.spacechar^0)

```

```

8920 * Cc(true), check_trail_joined)
8921
8922 parsers.check_trail_length_range = function(min, max)
8923 return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
8924 * Cc(max), check_trail_length)
8925 end
8926
8927 parsers.check_trail_length = function(n)
8928 return parsers.check_trail_length_range(n, n)
8929 end
8930

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

8931 parsers.freeze_trail = Cg(Cmt(Cb("indent_info")
8932 * Cc(true), trail_freezing), "indent_info")
8933
8934 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
8935 trail_freezing), "indent_info")
8936

```

The following patterns check indentation in continuation lines as defined by the container start.

```

8937 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
8938 check_continuation_indentation)
8939
8940 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
8941 check_continuation_indentation)
8942
8943 parsers.check_minimal_blank_indent
8944 = Cmt(Cb("indent_info") * Cc(false)
8945 * Cc(true)
8946 , check_continuation_indentation)
8947

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

8948
8949 parsers.check_minimal_indent_and_trail =
8950 Cmt(Cb("indent_info")
8951 * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
8952 , check_continuation_indentation_and_trail)
8953
8954 parsers.check_minimal_indent_and_code_trail =
8955 Cmt(Cb("indent_info")
8956 * Cc(false) * Cc(false) * Cc("code") * Cc(false)
8957 , check_continuation_indentation_and_trail)
8958

```

```

8959 parsers.check_minimal_blank_indent_and_full_code_trail =
8960 Cmt(Cb("indent_info")
8961 * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
8962 , check_continuation_indentation_and_trail)
8963
8964 parsers.check_minimal_indent_and_any_trail =
8965 Cmt(Cb("indent_info")
8966 * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
8967 , check_continuation_indentation_and_trail)
8968
8969 parsers.check_minimal_blank_indent_and_any_trail =
8970 Cmt(Cb("indent_info")
8971 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
8972 , check_continuation_indentation_and_trail)
8973
8974 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
8975 Cmt(Cb("indent_info")
8976 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
8977 , check_continuation_indentation_and_trail)
8978
8979 parsers.check_optional_indent_and_any_trail =
8980 Cmt(Cb("indent_info")
8981 * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
8982 , check_continuation_indentation_and_trail)
8983
8984 parsers.check_optional_blank_indent_and_any_trail =
8985 Cmt(Cb("indent_info")
8986 * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
8987 , check_continuation_indentation_and_trail)
8988

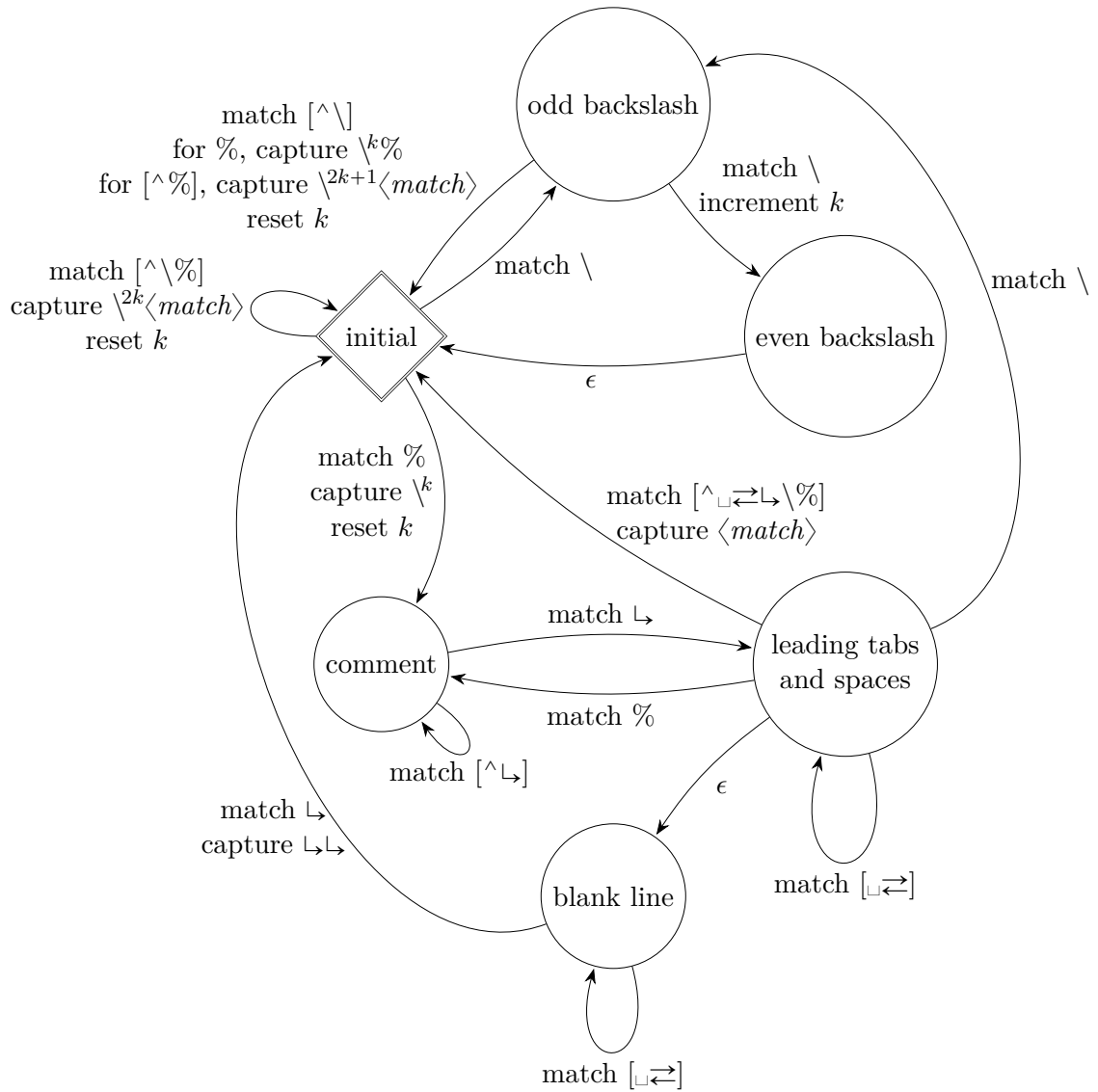
```

The following patterns specify behaviour around newlines.

```

8989
8990 parsers.spnlc_noexc = parsers.optionalspace
8991 * (parsers.newline
8992 * parsers.check_minimal_indent_and_any_trail)^-1
8993
8994 parsers.spnlc = parsers.optionalspace
8995 * (V("EndlineNoSub"))^-1
8996
8997 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
8998 + parsers.spacechar^1
8999
9000 parsers.only_blank = parsers.spacechar^0
9001 * (parsers.newline + parsers.eof)
9002

```



**Figure 8: A pushdown automaton that recognizes TeX comments**



The `parsers.commented_line1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 8.

```

9003 parsers.commented_line_letter = parsers.linechar
9004 + parsers.newline
9005 - parsers.backslash
9006 - parsers.percent
9007 parsers.commented_line = Cg(Cc(""), "backslashes")
9008 * ((#(parsers.commented_line_letter
9009 - parsers.newline)
9010 * Cb("backslashes")
9011 * Cs(parsers.commented_line_letter
9012 - parsers.newline)1 -- initial
9013 * Cg(Cc(""), "backslashes"))
9014 + #(parsers.backslash
9015 * (parsers.backslash + parsers.newline))
9016 * Cg((parsers.backslash -- even backslash
9017 * (parsers.backslash
9018 + #parsers.newline))1, "backslashes")
9019 + (parsers.backslash
9020 * (#parsers.percent
9021 * Cb("backslashes")
9022 / function(backslashes)
9023 return string.rep("\\", #backslashes / 2)
9024 end
9025 * C(parsers.percent)
9026 + #parsers.commented_line_letter
9027 * Cb("backslashes")
9028 * Cc("\\")
9029 * C(parsers.commented_line_letter))
9030 * Cg(Cc(""), "backslashes")))0
9031 * (#parsers.percent
9032 * Cb("backslashes")
9033 / function(backslashes)
9034 return string.rep("\\", #backslashes / 2)
9035 end
9036 * ((parsers.percent -- comment
9037 * parsers.line
9038 * #parsers.blankline) -- blank line
9039 / "\n"
9040 + parsers.percent -- comment
9041 * parsers.line
9042 * parsers.optionalspace) -- leading spaces
9043 + #(parsers.newline)
9044 * Cb("backslashes")
9045 * C(parsers.newline))
9046
9047 parsers.chunk = parsers.line * (parsers.optionallyindentedline

```

```

9048 - parsers.blankline)^0
9049
9050 parsers.attribute_key_char = parsers.unicode.following_alpha
9051 + parsers.unicode.following_numeric
9052 + S("-_:.")
9053 parsers.attribute_raw_char = parsers.unicode.following_alpha
9054 + parsers.unicode.following_numeric
9055 + S("-_")
9056 parsers.attribute_key = (parsers.attribute_key_char
9057 - parsers.dash - parsers.digit)
9058 * parsers.attribute_key_char^0
9059 parsers.attribute_value = ((parsers.dquote / "\"")
9060 * (parsers.anyescaped - parsers.dquote)^0
9061 * (parsers.dquote / "\""))
9062 + ((parsers.squote / "\"")
9063 * (parsers.anyescaped - parsers.squote)^0
9064 * (parsers.squote / "\""))
9065 + (parsers.anyescaped
9066 - parsers.dquote
9067 - parsers.rbrace
9068 - parsers.space)^0
9069 parsers.attribute_identfier = parsers.attribute_key_char^1
9070 parsers.attribute_classname = parsers.unicode.following_alpha
9071 * parsers.attribute_key_char^0
9072 parsers.attribute_raw = parsers.attribute_raw_char^1
9073
9074 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
9075 + C(parsers.hash
9076 * parsers.attribute_identfier)
9077 + C(parsers.period
9078 * parsers.attribute_classname)
9079 + Cs(parsers.attribute_key
9080 * parsers.optionalspace
9081 * parsers.equal
9082 * parsers.optionalspace
9083 * parsers.attribute_value)
9084 parsers.attributes = parsers.lbrace
9085 * parsers.optionalspace
9086 * parsers.attribute
9087 * (parsers.spacechar^1
9088 * parsers.attribute)^0
9089 * parsers.optionalspace
9090 * parsers.rbrace
9091
9092 parsers.raw_attribute = parsers.lbrace
9093 * parsers.optionalspace
9094 * parsers.equal

```

```

9095 * C(parsers.attribute_raw)
9096 * parsers.optionalspace
9097 * parsers.rbrace
9098
9099 -- block followed by 0 or more optionally
9100 -- indented blocks with first line indented.
9101 parsers.indented_blocks = function(bl)
9102 return Cs(bl
9103 * (parsers.blankline~1
9104 * parsers.indent
9105 * -parsers.blankline
9106 * bl)^0
9107 * (parsers.blankline~1 + parsers.eof))
9108 end

```

### 3.1.5.4 Parsers Used for HTML Entities

```

9109 local function repeat_between(pattern, min, max)
9110 return -pattern^(max + 1) * pattern^min
9111 end
9112
9113 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
9114 * C(repeat_between(parsers.hexdigit, 1, 6))
9115 * parsers.semicolon
9116 parsers.decentity = parsers.ampersand * parsers.hash
9117 * C(repeat_between(parsers.digit, 1, 7))
9118 * parsers.semicolon
9119 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric~1)
9120 * parsers.semicolon
9121
9122 parsers.html_entities
9123 = parsers.hexentity / entities.hex_entity_with_x_char
9124 + parsers.decentity / entities.dec_entity
9125 + parsers.tagentity / entities.char_entity

```

### 3.1.5.5 Parsers Used for Markdown Lists

```

9126 parsers.bullet = function(bullet_char, interrupting)
9127 local allowed_end
9128 if interrupting then
9129 allowed_end = C(parsers.spacechar~1) * #parsers.linechar
9130 else
9131 allowed_end = C(parsers.spacechar~1)
9132 + #(parsers.newline + parsers.eof)
9133 end
9134 return parsers.check_trail
9135 * Ct(C(bullet_char) * Cc(""))
9136 * allowed_end

```

```

9137 end
9138
9139 local function tickbox(interior)
9140 return parsers.optionalspace * parsers.lbracket
9141 * interior * parsers.rbracket * parsers.spacechar^1
9142 end
9143
9144 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
9145 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
9146 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
9147

```

### 3.1.5.6 Parsers Used for Markdown Code Spans

```

9148 parsers.openticks = Cg(parsers.backtick^1, "ticks")
9149
9150 local function captures_equal_length(_,i,a,b)
9151 return #a == #b and i
9152 end
9153
9154 parsers.closeticks = Cmt(C(parsers.backtick^1)
9155 * Cb("ticks"), captures_equal_length)
9156
9157 parsers.intickschar = (parsers.any - S("\n\r`"))
9158 + V("NoSoftLineBreakEndline")
9159 + (parsers.backtick^1 - parsers.closeticks)
9160
9161 local function process_inticks(s)
9162 s = s:gsub("\n", " ")
9163 s = s:gsub("^ (.*) $", "%1")
9164 return s
9165 end
9166
9167 parsers.inticks = parsers.openticks
9168 * C(parsers.space^0)
9169 * parsers.closeticks
9170 + parsers.openticks
9171 * Cs(Cs(parsers.intickschar^0) / process_inticks)
9172 * parsers.closeticks
9173

```

### 3.1.5.7 Parsers Used for HTML

```

9174 -- case-insensitive match (we assume s is lowercase)
9175 -- must be single byte encoding
9176 parsers.keyword_exact = function(s)
9177 local parser = P(0)
9178 for i=1,#s do

```

```

9179 local c = s:sub(i,i)
9180 local m = c .. upper(c)
9181 parser = parser * S(m)
9182 end
9183 return parser
9184 end
9185
9186 parsers.special_block_keyword =
9187 parsers.keyword_exact("pre") +
9188 parsers.keyword_exact("script") +
9189 parsers.keyword_exact("style") +
9190 parsers.keyword_exact("textarea")
9191
9192 parsers.block_keyword =
9193 parsers.keyword_exact("address") +
9194 parsers.keyword_exact("article") +
9195 parsers.keyword_exact("aside") +
9196 parsers.keyword_exact("base") +
9197 parsers.keyword_exact("basefont") +
9198 parsers.keyword_exact("blockquote") +
9199 parsers.keyword_exact("body") +
9200 parsers.keyword_exact("caption") +
9201 parsers.keyword_exact("center") +
9202 parsers.keyword_exact("col") +
9203 parsers.keyword_exact("colgroup") +
9204 parsers.keyword_exact("dd") +
9205 parsers.keyword_exact("details") +
9206 parsers.keyword_exact("dialog") +
9207 parsers.keyword_exact("dir") +
9208 parsers.keyword_exact("div") +
9209 parsers.keyword_exact("dl") +
9210 parsers.keyword_exact("dt") +
9211 parsers.keyword_exact("fieldset") +
9212 parsers.keyword_exact("figcaption") +
9213 parsers.keyword_exact("figure") +
9214 parsers.keyword_exact("footer") +
9215 parsers.keyword_exact("form") +
9216 parsers.keyword_exact("frame") +
9217 parsers.keyword_exact("frameset") +
9218 parsers.keyword_exact("h1") +
9219 parsers.keyword_exact("h2") +
9220 parsers.keyword_exact("h3") +
9221 parsers.keyword_exact("h4") +
9222 parsers.keyword_exact("h5") +
9223 parsers.keyword_exact("h6") +
9224 parsers.keyword_exact("head") +
9225 parsers.keyword_exact("header") +

```

```

9226 parsers.keyword_exact("hr") +
9227 parsers.keyword_exact("html") +
9228 parsers.keyword_exact("iframe") +
9229 parsers.keyword_exact("legend") +
9230 parsers.keyword_exact("li") +
9231 parsers.keyword_exact("link") +
9232 parsers.keyword_exact("main") +
9233 parsers.keyword_exact("menu") +
9234 parsers.keyword_exact("menuitem") +
9235 parsers.keyword_exact("nav") +
9236 parsers.keyword_exact("noframes") +
9237 parsers.keyword_exact("ol") +
9238 parsers.keyword_exact("optgroup") +
9239 parsers.keyword_exact("option") +
9240 parsers.keyword_exact("p") +
9241 parsers.keyword_exact("param") +
9242 parsers.keyword_exact("section") +
9243 parsers.keyword_exact("source") +
9244 parsers.keyword_exact("summary") +
9245 parsers.keyword_exact("table") +
9246 parsers.keyword_exact("tbody") +
9247 parsers.keyword_exact("td") +
9248 parsers.keyword_exact("tfoot") +
9249 parsers.keyword_exact("th") +
9250 parsers.keyword_exact("thead") +
9251 parsers.keyword_exact("title") +
9252 parsers.keyword_exact("tr") +
9253 parsers.keyword_exact("track") +
9254 parsers.keyword_exact("ul")
9255
9256 -- end conditions
9257 parsers.html_blankline_end_condition
9258 = parsers.linechar^0
9259 * (parsers.newline
9260 * (parsers.check_minimal_blank_indent_and_any_trail
9261 * #parsers.blankline
9262 + parsers.check_minimal_indent_and_any_trail)
9263 * parsers.linechar^1)^0
9264 * (parsers.newline^-1 / "")
9265
9266 local function remove_trailing_blank_lines(s)
9267 return s:gsub("[\n\r]+%s*$", "")
9268 end
9269
9270 parsers.html_until_end = function(end_marker)
9271 return Cs(Cs((parsers.newline
9272 * (parsers.check_minimal_blank_indent_and_any_trail

```

```

9273 * #parsers.blankline
9274 + parsers.check_minimal_indent_and_any_trail)
9275 + parsers.linechar - end_marker)^0
9276 * parsers.linechar^0 * parsers.newline~1)
9277 / remove_trailing_blank_lines)
9278 end
9279
9280 -- attributes
9281 parsers.html_attribute_spacing = parsers.optionalspace
9282 * V("NoSoftLineBreakEndline")
9283 * parsers.optionalspace
9284 + parsers.spacechar^1
9285
9286 parsers.html_attribute_name = (parsers.letter
9287 + parsers.colon
9288 + parsers.underscore)
9289 * (parsers.alphanumeric
9290 + parsers.colon
9291 + parsers.underscore
9292 + parsers.period
9293 + parsers.dash)^0
9294
9295 parsers.html_attribute_value = parsers.squote
9296 * (parsers.linechar - parsers.squote)^0
9297 * parsers.squote
9298 + parsers.dquote
9299 * (parsers.linechar - parsers.dquote)^0
9300 * parsers.dquote
9301 + (parsers.any
9302 - parsers.spacechar
9303 - parsers.newline
9304 - parsers.dquote
9305 - parsers.squote
9306 - parsers.backtick
9307 - parsers.equal
9308 - parsers.less
9309 - parsers.more)^1
9310
9311 parsers.html_inline_attribute_value = parsers.squote
9312 * (V("NoSoftLineBreakEndline")
9313 + parsers.any
9314 - parsers.blankline^2
9315 - parsers.squote)^0
9316 * parsers.squote
9317 + parsers.dquote
9318 * (V("NoSoftLineBreakEndline")
9319 + parsers.any

```

```

9320 - parsers.blankline^2
9321 - parsers.dquote)^0
9322 * parsers.dquote
9323 + (parsers.any
9324 - parsers.spacechar
9325 - parsers.newline
9326 - parsers.dquote
9327 - parsers.squote
9328 - parsers.backtick
9329 - parsers.equal
9330 - parsers.less
9331 - parsers.more)^1
9332
9333 parsers.html_attribute_value_specification
9334 = parsers.optionalspace
9335 * parsers.equal
9336 * parsers.optionalspace
9337 * parsers.html_attribute_value
9338
9339 parsers.html_spnl = parsers.optionalspace
9340 * (V("NoSoftLineBreakEndline")
9341 * parsers.optionalspace)^-1
9342
9343 parsers.html_inline_attribute_value_specification
9344 = parsers.html_spnl
9345 * parsers.equal
9346 * parsers.html_spnl
9347 * parsers.html_inline_attribute_value
9348
9349 parsers.html_attribute
9350 = parsers.html_attribute_spacing
9351 * parsers.html_attribute_name
9352 * parsers.html_inline_attribute_value_specification^-1
9353
9354 parsers.html_non_newline_attribute
9355 = parsers.spacechar^1
9356 * parsers.html_attribute_name
9357 * parsers.html_attribute_value_specification^-1
9358
9359 parsers.nested_breaking_blank = parsers.newline
9360 * parsers.check_minimal_blank_indent
9361 * parsers.blankline
9362
9363 parsers.html_comment_start = P("<!--")
9364
9365 parsers.html_comment_end = P("-->")
9366

```



```

9367 parsers.html_comment
9368 = Cs(parsers.html_comment_start
9369 * parsers.html_until_end(parsers.html_comment_end))
9370
9371 parsers.html_inline_comment = (parsers.html_comment_start / "")
9372 * -P(">") * -P("->")
9373 * Cs((V("NoSoftLineBreakEndline")
9374 + parsers.any
9375 - parsers.nested_breaking_blank
9376 - parsers.html_comment_end)^0)
9377 * (parsers.html_comment_end / "")
9378
9379 parsers.html_cdatasection_start = P("<![CDATA[")
9380
9381 parsers.html_cdatasection_end = P("]]>")
9382
9383 parsers.html_cdatasection
9384 = Cs(parsers.html_cdatasection_start
9385 * parsers.html_until_end(parsers.html_cdatasection_end))
9386
9387 parsers.html_inline_cdatasection
9388 = parsers.html_cdatasection_start
9389 * Cs(V("NoSoftLineBreakEndline") + parsers.any
9390 - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
9391 * parsers.html_cdatasection_end
9392
9393 parsers.html_declaration_start = P("<!") * parsers.letter
9394
9395 parsers.html_declaration_end = P(">")
9396
9397 parsers.html_declaration
9398 = Cs(parsers.html_declaration_start
9399 * parsers.html_until_end(parsers.html_declaration_end))
9400
9401 parsers.html_inline_declaration
9402 = parsers.html_declaration_start
9403 * Cs(V("NoSoftLineBreakEndline") + parsers.any
9404 - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
9405 * parsers.html_declaration_end
9406
9407 parsers.html_instruction_start = P("<?")
9408
9409 parsers.html_instruction_end = P(">")
9410
9411 parsers.html_instruction
9412 = Cs(parsers.html_instruction_start
9413 * parsers.html_until_end(parsers.html_instruction_end))

```

```

9414
9415 parsers.html_inline_instruction = parsers.html_instruction_start
9416 * Cs(V("NoSoftLineBreakEndline")
9417 + parsers.any
9418 - parsers.nested_breaking_blank
9419 - parsers.html_instruction_end)^0
9420 * parsers.html_instruction_end
9421
9422 parsers.html_blankline = parsers.newline
9423 * parsers.optionalspace
9424 * parsers.newline
9425
9426 parsers.html_tag_start = parsers.less
9427
9428 parsers.html_tag_closing_start = parsers.less
9429 * parsers.slash
9430
9431 parsers.html_tag_end = parsers.html_spnl
9432 * parsers.more
9433
9434 parsers.html_empty_tag_end = parsers.html_spnl
9435 * parsers.slash
9436 * parsers.more
9437
9438 -- opening tags
9439 parsers.html_any_open_inline_tag = parsers.html_tag_start
9440 * parsers.keyword
9441 * parsers.html_attribute^0
9442 * parsers.html_tag_end
9443
9444 parsers.html_any_open_tag = parsers.html_tag_start
9445 * parsers.keyword
9446 * parsers.html_non_newline_attribute^0
9447 * parsers.html_tag_end
9448
9449 parsers.html_open_tag = parsers.html_tag_start
9450 * parsers.block_keyword
9451 * parsers.html_attribute^0
9452 * parsers.html_tag_end
9453
9454 parsers.html_open_special_tag = parsers.html_tag_start
9455 * parsers.special_block_keyword
9456 * parsers.html_attribute^0
9457 * parsers.html_tag_end
9458
9459 -- incomplete tags
9460 parsers.incomplete_tag_following = parsers.spacechar

```

```

9461 + parsers.more
9462 + parsers.slash * parsers.more
9463 + #(parsers.newline + parsers.eof)
9464
9465 parsers.incomplete_special_tag_following = parsers.spacechar
9466 + parsers.more
9467 + #(parsers.newline
9468 + parsers.eof)
9469
9470 parsers.html_incomplete_open_tag = parsers.html_tag_start
9471 * parsers.block_keyword
9472 * parsers.incomplete_tag_following
9473
9474 parsers.html_incomplete_open_special_tag
9475 = parsers.html_tag_start
9476 * parsers.special_block_keyword
9477 * parsers.incomplete_special_tag_following
9478
9479 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
9480 * parsers.block_keyword
9481 * parsers.incomplete_tag_following
9482
9483 parsers.html_incomplete_close_special_tag
9484 = parsers.html_tag_closing_start
9485 * parsers.special_block_keyword
9486 * parsers.incomplete_tag_following
9487
9488 -- closing tags
9489 parsers.html_close_tag = parsers.html_tag_closing_start
9490 * parsers.block_keyword
9491 * parsers.html_tag_end
9492
9493 parsers.html_any_close_tag = parsers.html_tag_closing_start
9494 * parsers.keyword
9495 * parsers.html_tag_end
9496
9497 parsers.html_close_special_tag = parsers.html_tag_closing_start
9498 * parsers.special_block_keyword
9499 * parsers.html_tag_end
9500
9501 -- empty tags
9502 parsers.html_any_empty_inline_tag = parsers.html_tag_start
9503 * parsers.keyword
9504 * parsers.html_attribute^0
9505 * parsers.html_empty_tag_end
9506
9507 parsers.html_any_empty_tag = parsers.html_tag_start

```

```

9508 * parsers.keyword
9509 * parsers.html_non_newline_attribute^0
9510 * parsers.optionalspace
9511 * parsers.slash
9512 * parsers.more
9513
9514 parsers.html_empty_tag = parsers.html_tag_start
9515 * parsers.block_keyword
9516 * parsers.html_attribute^0
9517 * parsers.html_empty_tag_end
9518
9519 parsers.html_empty_special_tag = parsers.html_tag_start
9520 * parsers.special_block_keyword
9521 * parsers.html_attribute^0
9522 * parsers.html_empty_tag_end
9523
9524 parsers.html_incomplete_blocks
9525 = parsers.html_incomplete_open_tag
9526 + parsers.html_incomplete_open_special_tag
9527 + parsers.html_incomplete_close_tag
9528
9529 -- parse special html blocks
9530 parsers.html_blankline_ending_special_block_opening
9531 = (parsers.html_close_special_tag
9532 + parsers.html_empty_special_tag)
9533 * #(parsers.optionalspace
9534 * (parsers.newline + parsers.eof))
9535
9536 parsers.html_blankline_ending_special_block
9537 = parsers.html_blankline_ending_special_block_opening
9538 * parsers.html_blankline_end_condition
9539
9540 parsers.html_special_block_opening
9541 = parsers.html_incomplete_open_special_tag
9542 - parsers.html_empty_special_tag
9543
9544 parsers.html_closing_special_block
9545 = parsers.html_special_block_opening
9546 * parsers.html_until_end(parsers.html_close_special_tag)
9547
9548 parsers.html_special_block
9549 = parsers.html_blankline_ending_special_block
9550 + parsers.html_closing_special_block
9551
9552 -- parse html blocks
9553 parsers.html_block_opening = parsers.html_incomplete_open_tag
9554 + parsers.html_incomplete_close_tag

```

```

9555
9556 parsers.html_block = parsers.html_block_opening
9557 * parsers.html_blankline_end_condition
9558
9559 -- parse any html blocks
9560 parsers.html_any_block_opening
9561 = (parsers.html_any_open_tag
9562 + parsers.html_any_close_tag
9563 + parsers.html_any_empty_tag)
9564 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
9565
9566 parsers.html_any_block = parsers.html_any_block_opening
9567 * parsers.html_blankline_end_condition
9568
9569 parsers.html_inline_comment_full = parsers.html_comment_start
9570 * -P(">") * -P("->")
9571 * Cs((V("NoSoftLineBreakEndline")
9572 + parsers.any - P("--")
9573 - parsers.nested_breaking_blank
9574 - parsers.html_comment_end)^0)
9575 * parsers.html_comment_end
9576
9577 parsers.html_inline_tags = parsers.html_inline_comment_full
9578 + parsers.html_any_empty_inline_tag
9579 + parsers.html_inline_instruction
9580 + parsers.html_inline_cdatasection
9581 + parsers.html_inline_declaration
9582 + parsers.html_any_open_inline_tag
9583 + parsers.html_any_close_tag
9584

```

### 3.1.5.8 Parsers Used for Markdown Tags and Links

```

9585 parsers.urlchar = parsers.anyescaped
9586 - parsers.newline
9587 - parsers.more
9588
9589 parsers.auto_link_scheme_part = parsers.alphanumeric
9590 + parsers.plus
9591 + parsers.period
9592 + parsers.dash
9593
9594 parsers.auto_link_scheme = parsers.letter
9595 * parsers.auto_link_scheme_part
9596 * parsers.auto_link_scheme_part^-30
9597
9598 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon

```

```

9599 * (parsers.any - parsers.spacing
9600 - parsers.less - parsers.more)^0
9601
9602 parsers.printable_characters = S(" !#$%&'*/+=?^_`{|}~-")
9603
9604 parsers.email_address_local_part_char = parsers.alphanumeric
9605 + parsers.printable_characters
9606
9607 parsers.email_address_local_part
9608 = parsers.email_address_local_part_char^1
9609
9610 parsers.email_address_dns_label = parsers.alphanumeric
9611 * (parsers.alphanumeric
9612 + parsers.dash)^-62
9613 * B(parsers.alphanumeric)
9614
9615 parsers.email_address_domain = parsers.email_address_dns_label
9616 * (parsers.period
9617 * parsers.email_address_dns_label)^0
9618
9619 parsers.email_address = parsers.email_address_local_part
9620 * parsers.at
9621 * parsers.email_address_domain
9622
9623 parsers.auto_link_url = parsers.less
9624 * C(parsers.absolute_uri)
9625 * parsers.more
9626
9627 parsers.auto_link_email = parsers.less
9628 * C(parsers.email_address)
9629 * parsers.more
9630
9631 parsers.auto_link_relative_reference = parsers.less
9632 * C(parsers.urlchar^1)
9633 * parsers.more
9634
9635 parsers.autolink = parsers.auto_link_url
9636 + parsers.auto_link_email
9637
9638 -- content in balanced brackets, parentheses, or quotes:
9639 parsers.bracketed = P{ parsers.lbracket
9640 * ((parsers.backslash / "\"" * parsers.rbracket
9641 + parsers.any - (parsers.lbracket
9642 + parsers.rbracket
9643 + parsers.blankline^2)
9644) + V(1))^0
9645 * parsers.rbracket }

```

```

9646
9647 parsers.inparens = P{ parsers.lparent
9648 * ((parsers.anyescaped - (parsers.lparent
9649 + parsers.rparent
9650 + parsers.blankline^2)
9651) + V(1))^0
9652 * parsers.rparent }
9653
9654 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
9655 * ((parsers.anyescaped - (parsers.squote
9656 + parsers.blankline^2)
9657) + V(1))^0
9658 * parsers.squote }
9659
9660 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
9661 * ((parsers.anyescaped - (parsers.dquote
9662 + parsers.blankline^2)
9663) + V(1))^0
9664 * parsers.dquote }
9665
9666 parsers.link_text = parsers.lbracket
9667 * Cs((parsers.alphanumeric^1
9668 + parsers.bracketed
9669 + parsers.inticks
9670 + parsers.autolink
9671 + V("InlineHtml")
9672 + (parsers.backslash * parsers.backslash)
9673 + (parsers.backslash
9674 * (parsers.lbracket
9675 + parsers.rbracket)
9676 + V("NoSoftLineBreakSpace")
9677 + V("NoSoftLineBreakEndline")
9678 + (parsers.any
9679 - (parsers.newline
9680 + parsers.lbracket
9681 + parsers.rbracket
9682 + parsers.blankline^2))))^0)
9683 * parsers.rbracket
9684
9685 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
9686 * #((parsers.any
9687 - parsers.rbracket)^-999
9688 * parsers.rbracket)
9689 * Cs((parsers.alphanumeric^1
9690 + parsers.inticks
9691 + parsers.autolink
9692 + V("InlineHtml")

```

```

9693 + (parsers.backslash * parsers.backslash)
9694 + (parsers.backslash
9695 * (parsers.lbracket
9696 + parsers.rbracket)
9697 + V("NoSoftLineBreakSpace")
9698 + V("NoSoftLineBreakEndline")
9699 + (parsers.any
9700 - (parsers.newline
9701 + parsers.lbracket
9702 + parsers.rbracket
9703 + parsers.blankline^2))))^1)
9704
9705 parsers.link_label = parsers.lbracket
9706 * parsers.link_label_body
9707 * parsers.rbracket
9708
9709 parsers.inparens_url = P{ parsers.lparent
9710 * ((parsers.anyescaped - (parsers.lparent
9711 + parsers.rparent
9712 + parsers.spacing)
9713) + V(1))^0
9714 * parsers.rparent }
9715
9716 -- url for markdown links, allowing nested brackets:
9717 parsers.url = parsers.less * Cs((parsers.anyescaped
9718 - parsers.newline
9719 - parsers.less
9720 - parsers.more)^0)
9721 * parsers.more
9722 + -parsers.less
9723 * Cs((parsers.inparens_url + (parsers.anyescaped
9724 - parsers.spacing
9725 - parsers.lparent
9726 - parsers.rparent))^1)
9727
9728 -- quoted text:
9729 parsers.title_s = parsers.squote
9730 * Cs((parsers.html_entities
9731 + V("NoSoftLineBreakSpace")
9732 + V("NoSoftLineBreakEndline")
9733 + (parsers.anyescaped
9734 - parsers.newline
9735 - parsers.squote
9736 - parsers.blankline^2))^0)
9737 * parsers.squote
9738
9739 parsers.title_d = parsers.dquote

```



```

9740 * Cs((parsers.html_entities
9741 + V("NoSoftLineBreakSpace")
9742 + V("NoSoftLineBreakEndline")
9743 + (parsers.anyescaped
9744 - parsers.newline
9745 - parsers.dquote
9746 - parsers.blankline^2))^0)
9747 * parsers.dquote
9748
9749 parsers.title_p = parsers.lparent
9750 * Cs((parsers.html_entities
9751 + V("NoSoftLineBreakSpace")
9752 + V("NoSoftLineBreakEndline")
9753 + (parsers.anyescaped
9754 - parsers.newline
9755 - parsers.lparent
9756 - parsers.rparent
9757 - parsers.blankline^2))^0)
9758 * parsers.rparent
9759
9760 parsers.title
9761 = parsers.title_d + parsers.title_s + parsers.title_p
9762
9763 parsers.optionaltitle
9764 = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
9765

```

### 3.1.5.9 Helpers for Links and Link Reference Definitions

```

9766 -- parse a reference definition: [foo]: /bar "title"
9767 parsers.define_reference_parser = (parsers.check_trail / "")
9768 * parsers.link_label * parsers.colon
9769 * parsers.spnlc * parsers.url
9770 * (parsers.spnlc_sep * parsers.title
9771 * parsers.only_blank
9772 + Cc("") * parsers.only_blank)

```

### 3.1.5.10 Inline Elements

```

9773 parsers.Inline = V("Inline")
9774
9775 -- parse many p between starter and ender
9776 parsers.between = function(p, starter, ender)
9777 local ender2 = B(parsers.nonspacechar) * ender
9778 return (starter
9779 * #parsers.nonspacechar
9780 * Ct(p * (p - ender2)^0)
9781 * ender2)

```

```

9782 end
9783

```

### 3.1.5.11 Block Elements

```

9784 parsers.lineof = function(c)
9785 return (parsers.check_trail_no_rem
9786 * (P(c) * parsers.optionalspace)^3
9787 * (parsers.newline + parsers.eof))
9788 end
9789
9790 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
9791 + parsers.lineof(parsers.dash)
9792 + parsers.lineof(parsers.underscore)

```

### 3.1.5.12 Headings

```

9793 -- parse Atx heading start and return level
9794 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
9795 * -parsers.hash / length
9796
9797 -- parse setext header ending and return level
9798 parsers.heading_level
9799 = parsers.nonindentspace * parsers.equal^1
9800 * parsers.optionalspace * #parsers.newline * Cc(1)
9801 + parsers.nonindentspace * parsers.dash^1
9802 * parsers.optionalspace * #parsers.newline * Cc(2)
9803
9804 local function strip_atx_end(s)
9805 return s:gsub("%s+#+%s*\n$", "")
9806 end
9807
9808 parsers.atx_heading = parsers.check_trail_no_rem
9809 * Cg(parsers.heading_start, "level")
9810 * (C(parsers.optionalspace
9811 * parsers.hash^0
9812 * parsers.optionalspace
9813 * parsers.newline)
9814 + parsers.spacechar^1
9815 * C(parsers.line))

```

## 3.1.6 Markdown Reader

This section documents the [reader](#) object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the [lunamark/reader/markdown.lua](#) file in the Lunamark Lua module.

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

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```
9816 M.reader = {}
9817 function M.reader.new(writer, options)
9818 local self = {}
```

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

```
9819 self.writer = writer
9820 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.

```
9821 self.parsers = {}
9822 (function(parsers)
9823 setmetatable(self.parsers, {
9824 __index = function (_, key)
9825 return parsers[key]
9826 end
9827 })
9828 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.

```
9829 local parsers = self.parsers
```

### 3.1.6.1 Top-Level Helper Functions

Define `reader->normalize_tag` as a function that normalizes a markdown reference tag by lowercasing it, and by collapsing any adjacent whitespace characters.

```
9830 function self.normalize_tag(tag)
9831 tag = util.ropetostring(tag)
9832 tag = tag:gsub("[\n\r\t]+", " ")
9833 tag = tag:gsub("^ ", ""):gsub(" $", "")
9834 local form = nil
9835 if options.unicodeNormalization then
9836 form = options.unicodeNormalizationForm
9837 end
9838 tag = util.casefold(tag, form)
9839 return tag
9840 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.

```

9841 local function iterlines(s, f)
9842 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
9843 return util.rope_to_string(rope)
9844 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.

```

9845 if options.preserveTabs then
9846 self.expandtabs = function(s) return s end
9847 else
9848 self.expandtabs = function(s)
9849 if s:find("\t") then
9850 return iterlines(s, util.expand_tabs_in_line)
9851 else
9852 return s
9853 end
9854 end
9855 end

```

### 3.1.6.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```

9856 self.parser_functions = {}
9857 self.create_parser = function(name, grammar, toplevel)
9858 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.

```

9859 if toplevel and options.stripIndent then
9860 local min_prefix_length, min_prefix = nil, ''
9861 str = iterlines(str, function(line)
9862 if lpeg.match(parsers.nonemptyline, line) == nil then
9863 return line
9864 end
9865 line = util.expand_tabs_in_line(line)
9866 local prefix = lpeg.match(C(parsers.optionalspace), line)
9867 local prefix_length = #prefix
9868 local is_shorter = min_prefix_length == nil

```

```

9869 if not is_shorter then
9870 is_shorter = prefix_length < min_prefix_length
9871 end
9872 if is_shorter then
9873 min_prefix_length, min_prefix = prefix_length, prefix
9874 end
9875 return line
9876 end)
9877 str = str:gsub('^' .. min_prefix, '')
9878 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.

```

9879 if toplevel and (options.texComments or options.hybrid) then
9880 str = lpeg.match(Ct(parsers.commented_line^1), str)
9881 str = util.rope_to_string(str)
9882 end
9883 local res = lpeg.match(grammar(), str)
9884 if res == nil then
9885 return writer.error(
9886 format("Parser `%s` failed to process the input text.", name),
9887 format("Here are the first 20 characters of the remaining "
9888 .. "unprocessed text: `%s`.", str:sub(1,20))
9889)
9890 else
9891 return res
9892 end
9893 end
9894 end
9895
9896 self.create_parser("parse_blocks",
9897 function()
9898 return parsers.blocks
9899 end, true)
9900
9901 self.create_parser("parse_blocks_nested",
9902 function()
9903 return parsers.blocks_nested
9904 end, false)
9905
9906 self.create_parser("parse_inlines",
9907 function()
9908 return parsers.inlines
9909 end, false)
9910
9911 self.create_parser("parse_inlines_no_inline_note",

```

```

9912 function()
9913 return parsers.inlines_no_inline_note
9914 end, false)
9915
9916 self.create_parser("parse_inlines_no_html",
9917 function()
9918 return parsers.inlines_no_html
9919 end, false)
9920
9921 self.create_parser("parse_inlines_nbsp",
9922 function()
9923 return parsers.inlines_nbsp
9924 end, false)
9925 self.create_parser("parse_inlines_no_link_or_emphasis",
9926 function()
9927 return parsers.inlines_no_link_or_emphasis
9928 end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

9929 parsers.minimally_indented_blankline
9930 = parsers.check_minimal_indent * (parsers.blankline / "")
9931
9932 parsers.minimally_indented_block
9933 = parsers.check_minimal_indent * V("Block")
9934
9935 parsers.minimally_indented_block_or_paragraph
9936 = parsers.check_minimal_indent * V("BlockOrParagraph")
9937
9938 parsers.minimally_indented_paragraph
9939 = parsers.check_minimal_indent * V("Paragraph")
9940
9941 parsers.minimally_indented_plain
9942 = parsers.check_minimal_indent * V("Plain")
9943
9944 parsers.minimally_indented_par_or_plain
9945 = parsers.minimally_indented_paragraph
9946 + parsers.minimally_indented_plain
9947
9948 parsers.minimally_indented_par_or_plain_no_blank
9949 = parsers.minimally_indented_par_or_plain
9950 - parsers.minimally_indented_blankline
9951
9952 parsers.minimally_indented_ref
9953 = parsers.check_minimal_indent * V("Reference")
9954

```

```

9955 parsers.minimally_indented_blank
9956 = parsers.check_minimal_indent * V("Blank")
9957
9958 parsers.conditionally_indented_blankline
9959 = parsers.check_minimal_blank_indent * (parsers.blankline / "")
9960
9961 parsers.minimally_indented_ref_or_block
9962 = parsers.minimally_indented_ref
9963 + parsers.minimally_indented_block
9964 - parsers.minimally_indented_blankline
9965
9966 parsers.minimally_indented_ref_or_block_or_par
9967 = parsers.minimally_indented_ref
9968 + parsers.minimally_indented_block_or_paragraph
9969 - parsers.minimally_indented_blankline
9970

```

The following pattern parses the properly indented content that follows the initial container start.

```

9971
9972 function parsers.separator_loop(separated_block, paragraph,
9973 block_separator, paragraph_separator)
9974 return separated_block
9975 + block_separator
9976 * paragraph
9977 * separated_block
9978 + paragraph_separator
9979 * paragraph
9980 end
9981
9982 function parsers.create_loop_body_pair(separated_block, paragraph,
9983 block_separator,
9984 paragraph_separator)
9985 return {
9986 block = parsers.separator_loop(separated_block, paragraph,
9987 block_separator, block_separator),
9988 par = parsers.separator_loop(separated_block, paragraph,
9989 block_separator, paragraph_separator)
9990 }
9991 end
9992
9993 parsers.block_sep_group = function(blank)
9994 return blank^0 * parsers.eof
9995 + (blank^2 / writer.paragraphsep
9996 + blank^0 / writer.interblocksep
9997)
9998 end

```

```

9999
10000 parsers.par_sep_group = function(blank)
10001 return blank^0 * parsers.eof
10002 + blank^0 / writer.paragraphsep
10003 end
10004
10005 parsers.sep_group_no_output = function(blank)
10006 return blank^0 * parsers.eof
10007 + blank^0
10008 end
10009
10010 parsers.content_blank = parsers.minimally_indented_blankline
10011
10012 parsers.ref_or_block_separated
10013 = parsers.sep_group_no_output(parsers.content_blank)
10014 * (parsers.minimally_indented_ref
10015 - parsers.content_blank)
10016 + parsers.block_sep_group(parsers.content_blank)
10017 * (parsers.minimally_indented_block
10018 - parsers.content_blank)
10019
10020 parsers.loop_body_pair =
10021 parsers.create_loop_body_pair(
10022 parsers.ref_or_block_separated,
10023 parsers.minimally_indented_par_or_plain_no_blank,
10024 parsers.block_sep_group(parsers.content_blank),
10025 parsers.par_sep_group(parsers.content_blank))
10026
10027 parsers.content_loop = (V("Block")
10028 * parsers.loop_body_pair.block^0
10029 + (V("Paragraph") + V("Plain")))
10030 * parsers.ref_or_block_separated
10031 * parsers.loop_body_pair.block^0
10032 + (V("Paragraph") + V("Plain")))
10033 * parsers.loop_body_pair.par^0)
10034 * parsers.content_blank^0
10035
10036 parsers.indented_content = function()
10037 return Ct((V("Reference") + (parsers.blankline / ""))
10038 * parsers.content_blank^0
10039 * parsers.check_minimal_indent
10040 * parsers.content_loop
10041 + (V("Reference") + (parsers.blankline / ""))
10042 * parsers.content_blank^0
10043 + parsers.content_loop)
10044 end
10045

```



```

10046 parsers.add_indent = function(pattern, name, breakable)
10047 return Cg(Cmt(Cb("indent_info")
10048 * Ct(pattern)
10049 * (#parsers.linechar -- check if starter is blank
10050 * Cc(false) + Cc(true))
10051 * Cc(name)
10052 * Cc(breakable),
10053 process_starter_indent), "indent_info")
10054 end
10055

```

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

```

10056 if options.hashEnumerators then
10057 parsers.dig = parsers.digit + parsers.hash
10058 else
10059 parsers.dig = parsers.digit
10060 end
10061
10062 parsers.enumerator = function(delimiter_type, interrupting)
10063 local delimiter_range
10064 local allowed_end
10065 if interrupting then
10066 delimiter_range = P("1")
10067 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10068 else
10069 delimiter_range = parsers.dig * parsers.dig^-8
10070 allowed_end = C(parsers.spacechar^1)
10071 + #(parsers.newline + parsers.eof)
10072 end
10073
10074 return parsers.check_trail
10075 * Ct(C(delimiter_range) * C(delimiter_type))
10076 * allowed_end
10077 end
10078
10079 parsers.starter = parsers.bullet(parsers.dash)
10080 + parsers.bullet(parsers.asterisk)
10081 + parsers.bullet(parsers.plus)
10082 + parsers.enumerator(parsers.period)
10083 + parsers.enumerator(parsers.rparent)
10084

```

#### 3.1.6.5 Parsers Used for Blockquotes (local)

```

10085 parsers.blockquote_start
10086 = parsers.check_trail
10087 * C(parsers.more)

```

```

10088 * C(parsers.spacechar^0)
10089
10090 parsers.blockquote_body
10091 = parsers.add_indent(parsers.blockquote_start, "bq", true)
10092 * parsers.indented_content()
10093 * remove_indent("bq")
10094
10095 if not options.breakableBlockquotes then
10096 parsers.blockquote_body
10097 = parsers.add_indent(parsers.blockquote_start, "bq", false)
10098 * parsers.indented_content()
10099 * remove_indent("bq")
10100 end

```

### 3.1.6.6 Helpers for Emphasis and Strong Emphasis (local)

Parse the content of a table `content_part` with links, images and emphasis disabled.

```

10101 local function parse_content_part(content_part)
10102 local rope = util.rope_to_string(content_part)
10103 local parsed
10104 = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
10105 parsed.indent_info = nil
10106 return parsed
10107 end
10108

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10109 local collect_emphasis_content =
10110 function(t, opening_index, closing_index)
10111 local content = {}
10112
10113 local content_part = {}
10114 for i = opening_index, closing_index do
10115 local value = t[i]
10116
10117 if value.rendered ~= nil then
10118 content[#content + 1] = parse_content_part(content_part)
10119 content_part = {}
10120 content[#content + 1] = value.rendered
10121 value.rendered = nil
10122 else
10123 if value.warning ~= nil then
10124 if next(content_part) ~= nil then
10125 content[#content + 1] = parse_content_part(content_part)
10126 end
10127 content_part = {}

```

```

10128
10129 content[#content + 1] = value.warning
10130 value.warning = nil
10131 end
10132 if value.type == "delimiter"
10133 and value.element == "emphasis" then
10134 if value.is_active then
10135 content_part[#content_part + 1]
10136 = string.rep(value.character, value.current_count)
10137 end
10138 else
10139 content_part[#content_part + 1] = value.content
10140 end
10141 value.content = ''
10142 value.is_active = false
10143 end
10144 end
10145
10146 if next(content_part) ~= nil then
10147 content[#content + 1] = parse_content_part(content_part)
10148 end
10149
10150 return content
10151 end
10152

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

10153 local function fill_emph(t, opening_index, closing_index)
10154 local content
10155 = collect_emphasis_content(t, opening_index + 1,
10156 closing_index - 1)
10157 t[opening_index + 1].is_active = true
10158 t[opening_index + 1].rendered = writer.emphasis(content)
10159 end
10160

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

10161 local function fill_strong(t, opening_index, closing_index)
10162 local content
10163 = collect_emphasis_content(t, opening_index + 1,
10164 closing_index - 1)
10165 t[opening_index + 1].is_active = true
10166 t[opening_index + 1].rendered = writer.strong(content)
10167 end
10168

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

10169 local function breaks_three_rule(opening_delimiter, closing_delimiter)
10170 return (opening_delimiter.is_closing
10171 or closing_delimiter.is_opening)
10172 and ((opening_delimiter.original_count
10173 + closing_delimiter.original_count) % 3 == 0)
10174 and (opening_delimiter.original_count % 3 ~= 0
10175 or closing_delimiter.original_count % 3 ~= 0)
10176 end
10177

```

Look for the first potential emphasis opener in the delimiter table `t` in the range from `bottom_index` to `latest_index` that has the same character `character` as the closing delimiter `closing_delimiter`.

```

10178 local find_emphasis_opener = function(t, bottom_index, latest_index,
10179 character, closing_delimiter)
10180 for i = latest_index, bottom_index, -1 do
10181 local value = t[i]
10182 if value.is_active and
10183 value.is_opening and
10184 value.type == "delimiter" and
10185 value.element == "emphasis" and
10186 (value.character == character) and
10187 (value.current_count > 0) then
10188 if not breaks_three_rule(value, closing_delimiter) then
10189 return i
10190 end
10191 end
10192 end
10193 end
10194

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

10195 local function process_emphasis(t, opening_index, closing_index)
10196 for i = opening_index, closing_index do
10197 local value = t[i]
10198 if value.type == "delimiter" and value.element == "emphasis" then
10199 local delimiter_length = string.len(value.content)
10200 value.character = string.sub(value.content, 1, 1)
10201 value.current_count = delimiter_length
10202 value.original_count = delimiter_length
10203 end
10204 end
10205
10206 local openers_bottom = {

```

```

10207 ['*'] = {
10208 [true] = {opening_index, opening_index, opening_index},
10209 [false] = {opening_index, opening_index, opening_index}
10210 },
10211 ['_'] = {
10212 [true] = {opening_index, opening_index, opening_index},
10213 [false] = {opening_index, opening_index, opening_index}
10214 }
10215 }
10216
10217 local current_position = opening_index
10218 local max_position = closing_index
10219
10220 while current_position <= max_position do
10221 local value = t[current_position]
10222
10223 if value.type ~= "delimiter" or
10224 value.element ~= "emphasis" or
10225 not value.is_active or
10226 not value.is_closing or
10227 (value.current_count <= 0) then
10228 current_position = current_position + 1
10229 goto continue
10230 end
10231
10232 local character = value.character
10233 local is_opening = value.is_opening
10234 local closing_length_modulo_three = value.original_count % 3
10235
10236 local current_openers_bottom
10237 = openers_bottom[character][is_opening]
10238 [closing_length_modulo_three + 1]
10239
10240 local opener_position
10241 = find_emphasis_opener(t, current_openers_bottom,
10242 current_position - 1, character, value)
10243
10244 if (opener_position == nil) then
10245 openers_bottom[character][is_opening]
10246 [closing_length_modulo_three + 1]
10247 = current_position
10248 current_position = current_position + 1
10249 goto continue
10250 end
10251
10252 local opening_delimiter = t[opener_position]
10253

```

```

10254 local current_opening_count = opening_delimiter.current_count
10255 local current_closing_count = t[current_position].current_count
10256
10257 if (current_opening_count >= 2)
10258 and (current_closing_count >= 2) then
10259 opening_delimiter.current_count = current_opening_count - 2
10260 t[current_position].current_count = current_closing_count - 2
10261 fill_strong(t, opener_position, current_position)
10262 else
10263 opening_delimiter.current_count = current_opening_count - 1
10264 t[current_position].current_count = current_closing_count - 1
10265 fill_emph(t, opener_position, current_position)
10266 end
10267
10268 ::continue::
10269 end
10270 end
10271
10272 parsers.delimiter_run = function(character)
10273 return (B(parsers.backslash * character) + -B(character))
10274 * character~1
10275 * -#character
10276 end
10277
10278 parsers.left_flanking_delimiter_run = function(character)
10279 return (B(parsers.any)
10280 * (parsers.unicode.preceding_punctuation
10281 + parsers.unicode.preceding_whitespace)
10282 + -B(parsers.any))
10283 * parsers.delimiter_run(character)
10284 * parsers.unicode.following_punctuation
10285 + parsers.delimiter_run(character)
10286 * -(parsers.unicode.following_punctuation
10287 + parsers.unicode.following_whitespace
10288 + parsers.eof)
10289 end
10290
10291 parsers.right_flanking_delimiter_run = function(character)
10292 return parsers.unicode.preceding_punctuation
10293 * parsers.delimiter_run(character)
10294 * (parsers.unicode.following_punctuation
10295 + parsers.unicode.following_whitespace
10296 + parsers.eof)
10297 + (B(parsers.any)
10298 * -(parsers.unicode.preceding_punctuation
10299 + parsers.unicode.preceding_whitespace))
10300 * parsers.delimiter_run(character)

```

```

10301 end
10302
10303 if options.underscores then
10304 parsers.emph_start
10305 = parsers.left_flanking_delimiter_run(parsers.asterisk)
10306 + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
10307 + (parsers.unicode.preceding_punctuation
10308 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
10309 * parsers.left_flanking_delimiter_run(parsers.underscore)
10310
10311 parsers.emph_end
10312 = parsers.right_flanking_delimiter_run(parsers.asterisk)
10313 + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
10314 + #(parsers.left_flanking_delimiter_run(parsers.underscore)
10315 * parsers.unicode.following_punctuation))
10316 * parsers.right_flanking_delimiter_run(parsers.underscore)
10317 else
10318 parsers.emph_start
10319 = parsers.left_flanking_delimiter_run(parsers.asterisk)
10320
10321 parsers.emph_end
10322 = parsers.right_flanking_delimiter_run(parsers.asterisk)
10323 end
10324
10325 parsers.emph_capturing_open_and_close
10326 = #parsers.emph_start * #parsers.emph_end
10327 * Ct(Cg(Cc("delimiter"), "type")
10328 * Cg(Cc("emphasis"), "element")
10329 * Cg(C(parsers.emph_start), "content")
10330 * Cg(Cc(true), "is_opening")
10331 * Cg(Cc(true), "is_closing"))
10332
10333 parsers.emph_capturing_open = Ct(Cg(Cc("delimiter"), "type")
10334 * Cg(Cc("emphasis"), "element")
10335 * Cg(C(parsers.emph_start), "content")
10336 * Cg(Cc(true), "is_opening")
10337 * Cg(Cc(false), "is_closing"))
10338
10339 parsers.emph_capturing_close = Ct(Cg(Cc("delimiter"), "type")
10340 * Cg(Cc("emphasis"), "element")
10341 * Cg(C(parsers.emph_end), "content")
10342 * Cg(Cc(false), "is_opening")
10343 * Cg(Cc(true), "is_closing"))
10344
10345 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
10346 + parsers.emph_capturing_open
10347 + parsers.emph_capturing_close

```

```

10348
10349 parsers.emph_open = parsers.emph_capturing_open_and_close
10350 + parsers.emph_capturing_open
10351
10352 parsers.emph_close = parsers.emph_capturing_open_and_close
10353 + parsers.emph_capturing_close
10354

```

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

```

10355 -- List of references defined in the document
10356 local references
10357
10358 -- List of note references defined in the document
10359 parsers.rawnotes = {}
10360

```

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.

```

10361 function self.register_link(_, tag, url, title,
10362 attributes)
10363 local normalized_tag = self.normalize_tag(tag)
10364 if references[normalized_tag] == nil then
10365 references[normalized_tag] = {
10366 url = url,
10367 title = title,
10368 attributes = attributes
10369 }
10370 return ""
10371 else
10372 local text
10373 = string.format('Multiply defined link reference "%s"', tag)
10374 local more = string.format("Look for the text `[%s]: ...`.", tag)
10375 return writer.warning(text, more)
10376 end
10377 end
10378

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

10379 function self.lookup_reference(tag)
10380 return references[self.normalize_tag(tag)]
10381 end
10382

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

10383 function self.lookup_note_reference(tag)

```



```

10384 return parsers.rawnotes[self.normalize_tag(tag)]
10385 end
10386
10387 parsers.title_s_direct_ref = parsers.squote
10388 * Cs((parsers.html_entities
10389 + (parsers.anyescaped
10390 - parsers.squote
10391 - parsers.blankline^2))^0)
10392 * parsers.squote
10393
10394 parsers.title_d_direct_ref = parsers.dquote
10395 * Cs((parsers.html_entities
10396 + (parsers.anyescaped
10397 - parsers.dquote
10398 - parsers.blankline^2))^0)
10399 * parsers.dquote
10400
10401 parsers.title_p_direct_ref = parsers.lparent
10402 * Cs((parsers.html_entities
10403 + (parsers.anyescaped
10404 - parsers.lparent
10405 - parsers.rparent
10406 - parsers.blankline^2))^0)
10407 * parsers.rparent
10408
10409 parsers.title_direct_ref = parsers.title_s_direct_ref
10410 + parsers.title_d_direct_ref
10411 + parsers.title_p_direct_ref
10412
10413 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
10414 * Cg(parsers.url + Cc(""), "url")
10415 * parsers.spnl
10416 * Cg(parsers.title_direct_ref
10417 + Cc(""), "title")
10418 * parsers.spnl * parsers.rparent
10419
10420 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
10421 * Cg(parsers.url + Cc(""), "url")
10422 * parsers.spnlc
10423 * Cg(parsers.title + Cc(""), "title")
10424 * parsers.spnlc * parsers.rparent
10425
10426 parsers.empty_link = parsers.lbracket
10427 * parsers.rbracket
10428
10429 parsers.inline_link = parsers.link_text
10430 * parsers.inline_direct_ref

```

```

10431
10432 parsers.full_link = parsers.link_text
10433 * parsers.link_label
10434
10435 parsers.shortcut_link = parsers.link_label
10436 * -(parsers.empty_link + parsers.link_label)
10437
10438 parsers.collapsed_link = parsers.link_label
10439 * parsers.empty_link
10440
10441 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
10442 * Cg(Cc("inline"), "link_type")
10443 + #(parsers.exclamation * parsers.full_link)
10444 * Cg(Cc("full"), "link_type")
10445 + #(parsers.exclamation
10446 * parsers.collapsed_link)
10447 * Cg(Cc("collapsed"), "link_type")
10448 + #(parsers.exclamation * parsers.shortcut_link)
10449 * Cg(Cc("shortcut"), "link_type")
10450 + #(parsers.exclamation * parsers.empty_link)
10451 * Cg(Cc("empty"), "link_type")
10452
10453 parsers.link_opening = #parsers.inline_link
10454 * Cg(Cc("inline"), "link_type")
10455 + #parsers.full_link
10456 * Cg(Cc("full"), "link_type")
10457 + #parsers.collapsed_link
10458 * Cg(Cc("collapsed"), "link_type")
10459 + #parsers.shortcut_link
10460 * Cg(Cc("shortcut"), "link_type")
10461 + #parsers.empty_link
10462 * Cg(Cc("empty_link"), "link_type")
10463 + #parsers.link_text
10464 * Cg(Cc("link_text"), "link_type")
10465
10466 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
10467 * Cg(Cc("note_inline"), "link_type")
10468
10469 parsers.raw_note_opening = #(parsers.lbracket
10470 * parsers.circumflex
10471 * parsers.link_label_body
10472 * parsers.rbracket)
10473 * Cg(Cc("raw_note"), "link_type")
10474
10475 local inline_note_element = Cg(Cc("note"), "element")
10476 * parsers.note_opening
10477 * Cg(parsers.circumflex

```

```

10478 * parsers.lbracket, "content")
10479
10480 local image_element = Cg(Cc("image"), "element")
10481 * parsers.image_opening
10482 * Cg(parsers.exclamation
10483 * parsers.lbracket, "content")
10484
10485 local note_element = Cg(Cc("note"), "element")
10486 * parsers.raw_note_opening
10487 * Cg(parsers.lbracket
10488 * parsers.circumflex, "content")
10489
10490 local link_element = Cg(Cc("link"), "element")
10491 * parsers.link_opening
10492 * Cg(parsers.lbracket, "content")
10493
10494 local opening_elements = parsers.fail
10495
10496 if options.inlineNotes then
10497 opening_elements = opening_elements + inline_note_element
10498 end
10499
10500 opening_elements = opening_elements + image_element
10501
10502 if options.notes then
10503 opening_elements = opening_elements + note_element
10504 end
10505
10506 opening_elements = opening_elements + link_element
10507
10508 parsers.link_image_opening = Ct(Cg(Cc("delimiter"), "type")
10509 * Cg(Cc(true), "is_opening")
10510 * Cg(Cc(false), "is_closing")
10511 * opening_elements)
10512
10513 parsers.link_image_closing = Ct(Cg(Cc("delimiter"), "type")
10514 * Cg(Cc("link"), "element")
10515 * Cg(Cc(false), "is_opening")
10516 * Cg(Cc(true), "is_closing")
10517 * (Cg(Cc(true), "is_direct")
10518 * Cg(parsers.rbracket
10519 * #parsers.inline_direct_ref,
10520 "content")
10521 + Cg(Cc(false), "is_direct")
10522 * Cg(parsers.rbracket, "content")))
10523
10524 parsers.link_image_open_or_close = parsers.link_image_opening

```

```

10525 + parsers.link_image_closing
10526
10527 if options.html then
10528 parsers.link_emph_precedence = parsers.inticks
10529 + parsers.autolink
10530 + parsers.html_inline_tags
10531 else
10532 parsers.link_emph_precedence = parsers.inticks
10533 + parsers.autolink
10534 end
10535
10536 parsers.link_and_emph_endline = parsers.newline
10537 * ((parsers.check_minimal_indent
10538 * -V("EndlineExceptions")
10539 + parsers.check_optional_indent
10540 * -V("EndlineExceptions")
10541 * -V("ListStarter")) / "")
10542 * parsers.spacechar^0 / "\n"
10543
10544 parsers.link_and_emph_content
10545 = Ct(Cg(Cc("content"), "type")
10546 * Cg(Cs((parsers.link_emph_precedence
10547 + parsers.backslash * parsers.linechar
10548 + parsers.link_and_emph_endline
10549 + (parsers.linechar
10550 - parsers.blankline^2
10551 - parsers.link_image_open_or_close
10552 - parsers.emph_open_or_close))^0), "content"))
10553
10554 parsers.link_and_emph_table
10555 = (parsers.link_image_opening + parsers.emph_open)
10556 * parsers.link_and_emph_content
10557 * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
10558 * parsers.link_and_emph_content)^1
10559
10560 Collect the content between the opening_index and closing_index in the delimiter
10561 table t.
10562
10563 local function collect_link_content(t, opening_index, closing_index)
10564 local content = {}
10565 for i = opening_index, closing_index do
10566 content[#content + 1] = t[i].content
10567 end
10568 return util.rope_to_string(content)
10569 end
10570

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
10568 local function find_link_opener(t, bottom_index, latest_index)
10569 for i = latest_index, bottom_index, -1 do
10570 local value = t[i]
10571 if value.type == "delimiter" and
10572 value.is_opening and
10573 (value.element == "link"
10574 or value.element == "image"
10575 or value.element == "note")
10576 and not value.removed then
10577 if value.is_active then
10578 return i
10579 end
10580 value.removed = true
10581 return nil
10582 end
10583 end
10584 end
10585
```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
10586 local function find_next_link_closing_index(t, latest_index)
10587 for i = latest_index, #t do
10588 local value = t[i]
10589 if value.is_closing and
10590 value.element == "link" and
10591 not value.removed then
10592 return i
10593 end
10594 end
10595 end
10596
```

Disable all preceding opening link delimiters by marking them inactive with the `is_active` property to prevent links within links. Images within links are allowed.

```
10597 local function disable_previous_link_openers(t, opening_index)
10598 if t[opening_index].element == "image" then
10599 return
10600 end
10601
10602 for i = opening_index, 1, -1 do
10603 local value = t[i]
10604 if value.is_active and
10605 value.type == "delimiter" and
10606 value.is_opening and
```

```

10607 value.element == "link" then
10608 value.is_active = false
10609 end
10610 end
10611 end
10612

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

10613 local function disable_range(t, opening_index, closing_index)
10614 for i = opening_index, closing_index do
10615 local value = t[i]
10616 if value.is_active then
10617 value.is_active = false
10618 if value.type == "delimiter" then
10619 value.removed = true
10620 end
10621 end
10622 end
10623 end
10624

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10625 local delete_parsed_content_in_range =
10626 function(t, opening_index, closing_index)
10627 for i = opening_index, closing_index do
10628 t[i].rendered = nil
10629 end
10630 end
10631

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10632 local function empty_content_in_range(t, opening_index, closing_index)
10633 for i = opening_index, closing_index do
10634 t[i].content = ''
10635 end
10636 end
10637

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

10638 local function join_attributes(reference_attributes, own_attributes)
10639 local merged_attributes = {}
10640 for _, attribute in ipairs(reference_attributes or {}) do
10641 table.insert(merged_attributes, attribute)
10642 end

```

```

10643 for _, attribute in ipairs(own_attributes or {}) do
10644 table.insert(merged_attributes, attribute)
10645 end
10646 if next(merged_attributes) == nil then
10647 merged_attributes = nil
10648 end
10649 return merged_attributes
10650 end
10651

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

10652 local render_link_or_image =
10653 function(t, opening_index, closing_index, content_end_index,
10654 reference)
10655 process_emphasis(t, opening_index, content_end_index)
10656 local mapped = collect_emphasis_content(t, opening_index + 1,
10657 content_end_index - 1)
10658
10659 local rendered = {}
10660 if (t[opening_index].element == "link") then
10661 rendered = writer.link(mapped, reference.url,
10662 reference.title, reference.attributes)
10663 end
10664
10665 if (t[opening_index].element == "image") then
10666 rendered = writer.image(mapped, reference.url, reference.title,
10667 reference.attributes)
10668 end
10669
10670 if (t[opening_index].element == "note") then
10671 if (t[opening_index].link_type == "note_inline") then
10672 rendered = writer.note(mapped)
10673 end
10674 if (t[opening_index].link_type == "raw_note") then
10675 rendered = writer.note(reference)
10676 end
10677 end
10678
10679 t[opening_index].rendered = rendered
10680 delete_parsed_content_in_range(t, opening_index + 1,
10681 closing_index)
10682 empty_content_in_range(t, opening_index, closing_index)
10683 disable_previous_link_openers(t, opening_index)
10684 disable_range(t, opening_index, closing_index)
10685 end
10686

```

Match the link destination of an inline link at index `closing_index` in table `t` when `match_reference` is true. Additionally, match attributes when the option `linkAttributes` is enabled.

```
10687 local resolve_inline_following_content =
10688 function(t, closing_index, match_reference, match_link_attributes)
10689 local content = ""
10690 for i = closing_index + 1, #t do
10691 content = content .. t[i].content
10692 end
10693
10694 local matching_content = parsers.succeed
10695
10696 if match_reference then
10697 matching_content = matching_content
10698 * parsers.inline_direct_ref_inside
10699 end
10700
10701 if match_link_attributes then
10702 matching_content = matching_content
10703 * Cg(Ct(parsers.attributes^-1), "attributes")
10704 end
10705
10706 local matched = lpeg.match(Ct(matching_content
10707 * Cg(Cp(), "end_position")), content)
10708
10709 local matched_count = matched.end_position - 1
10710 for i = closing_index + 1, #t do
10711 local value = t[i]
10712
10713 local chars_left = matched_count
10714 matched_count = matched_count - #value.content
10715
10716 if matched_count <= 0 then
10717 value.content = value.content:sub(chars_left + 1)
10718 break
10719 end
10720
10721 value.content = ''
10722 value.is_active = false
10723 end
10724
10725 local attributes = matched.attributes
10726 if attributes == nil or next(attributes) == nil then
10727 attributes = nil
10728 end
10729
10730 return {
```



```

10731 url = matched.url or "",
10732 title = matched.title or "",
10733 attributes = attributes
10734 }
10735 end
10736

```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```

10737 local function resolve_inline_link(t, opening_index, closing_index)
10738 local inline_content
10739 = resolve_inline_following_content(t, closing_index, true,
10740 t.match_link_attributes)
10741 render_link_or_image(t, opening_index, closing_index,
10742 closing_index, inline_content)
10743 end
10744

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

10745 local resolve_note_inline_link =
10746 function(t, opening_index, closing_index)
10747 local inline_content
10748 = resolve_inline_following_content(t, closing_index,
10749 false, false)
10750 render_link_or_image(t, opening_index, closing_index,
10751 closing_index, inline_content)
10752 end
10753

```

Resolve a shortcut link `[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

10754 local function resolve_shortcut_link(t, opening_index, closing_index)
10755 local content
10756 = collect_link_content(t, opening_index + 1, closing_index - 1)
10757 local r = self.lookup_reference(content)
10758
10759 if r then
10760 local inline_content
10761 = resolve_inline_following_content(t, closing_index, false,
10762 t.match_link_attributes)
10763 r.attributes
10764 = join_attributes(r.attributes, inline_content.attributes)
10765 render_link_or_image(t, opening_index, closing_index,
10766 closing_index, r)
10767 end

```

```

10768 end
10769

```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```

10770 local function resolve_raw_note_link(t, opening_index, closing_index)
10771 local content
10772 = collect_link_content(t, opening_index + 1, closing_index - 1)
10773 local r = self.lookup_note_reference(content)
10774
10775 if r then
10776 local parsed_ref = self.parser_functions.parse_blocks_nested(r)
10777 render_link_or_image(t, opening_index, closing_index,
10778 closing_index, parsed_ref)
10779 end
10780 end
10781

```

Resolve a full link `[a][b]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `b` is not found in the references.

```

10782 local function resolve_full_link(t, opening_index, closing_index)
10783 local next_link_closing_index
10784 = find_next_link_closing_index(t, closing_index + 4)
10785 local next_link_content
10786 = collect_link_content(t, closing_index + 3,
10787 next_link_closing_index - 1)
10788 local r = self.lookup_reference(next_link_content)
10789
10790 if r then
10791 local inline_content
10792 = resolve_inline_following_content(t, next_link_closing_index,
10793 false,
10794 t.match_link_attributes)
10795 r.attributes
10796 = join_attributes(r.attributes, inline_content.attributes)
10797 render_link_or_image(t, opening_index, next_link_closing_index,
10798 closing_index, r)
10799 else
10800 local text = string.format('Undefined link reference "%s"',
10801 next_link_content)
10802 local more = string.format("Look for the text `[...] [%s]`.",
10803 next_link_content)
10804 t[opening_index].warning = writer.warning(text, more)
10805 end
10806 end
10807

```

Resolve a collapsed link `[a][]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the references.

```

10808 local function resolve_collapsed_link(t, opening_index, closing_index)
10809 local next_link_closing_index
10810 = find_next_link_closing_index(t, closing_index + 4)
10811 local content
10812 = collect_link_content(t, opening_index + 1, closing_index - 1)
10813 local r = self.lookup_reference(content)
10814
10815 if r then
10816 local inline_content
10817 = resolve_inline_following_content(t, closing_index, false,
10818 t.match_link_attributes)
10819 r.attributes
10820 = join_attributes(r.attributes, inline_content.attributes)
10821 render_link_or_image(t, opening_index, next_link_closing_index,
10822 closing_index, r)
10823 else
10824 local text = string.format('Undefined link reference "%s"',
10825 content)
10826 local more = string.format("Look for the text `[%s][]`.",
10827 content)
10828 t[opening_index].warning = writer.warning(text, more)
10829 end
10830 end
10831
```

Parse a table of link and emphasis delimiters `t`. First, iterate over the link delimiters and produce either link or image macros. Then run `process_emphasis` over the entire delimiter table, resolving emphasis and strong emphasis and parsing any content outside of closed delimiters.

```

10832 local function process_links_and_emphasis(t)
10833 for _,value in ipairs(t) do
10834 value.is_active = true
10835 end
10836
10837 for i,value in ipairs(t) do
10838 if not value.is_closing
10839 or value.type ~= "delimiter"
10840 or not (value.element == "link"
10841 or value.element == "image"
10842 or value.element == "note")
10843 or value.removed then
10844 goto continue
10845 end
10846
```

```

10847 local opener_position = find_link_opener(t, 1, i - 1)
10848 if (opener_position == nil) then
10849 goto continue
10850 end
10851
10852 local opening_delimiter = t[opener_position]
10853 opening_delimiter.removed = true
10854
10855 local link_type = opening_delimiter.link_type
10856
10857 if (link_type == "inline") then
10858 resolve_inline_link(t, opener_position, i)
10859 end
10860 if (link_type == "shortcut") then
10861 resolve_shortcut_link(t, opener_position, i)
10862 end
10863 if (link_type == "full") then
10864 resolve_full_link(t, opener_position, i)
10865 end
10866 if (link_type == "collapsed") then
10867 resolve_collapsed_link(t, opener_position, i)
10868 end
10869 if (link_type == "note_inline") then
10870 resolve_note_inline_link(t, opener_position, i)
10871 end
10872 if (link_type == "raw_note") then
10873 resolve_raw_note_link(t, opener_position, i)
10874 end
10875
10876 ::continue::
10877 end
10878
10879 t[#t].content = t[#t].content:gsub("%s*$","")
10880
10881 process_emphasis(t, 1, #t)
10882 local final_result = collect_emphasis_content(t, 1, #t)
10883 return final_result
10884 end
10885
10886 function self.defer_link_and_emphasis_processing(delimiter_table)
10887 return writer.defer_call(function()
10888 return process_links_and_emphasis(delimiter_table)
10889 end)
10890 end
10891

```

### 3.1.6.8 Inline Elements (local)

```
10892 parsers.Str = (parsers.normalchar
10893 * (parsers.normalchar + parsers.at)^0)
10894 / writer.string
10895
10896 parsers.Symbol = (parsers.backtick^1 + V("SpecialChar"))
10897 / writer.string
10898
10899 parsers.Ellipsis = P("...") / writer.ellipsis
10900
10901 parsers.Smart = parsers.Ellipsis
10902
10903 parsers.Code = parsers.inticks / writer.code
10904
10905 if options.blankBeforeBlockquote then
10906 parsers.bqstart = parsers.fail
10907 else
10908 parsers.bqstart = parsers.blockquote_start
10909 end
10910
10911 if options.blankBeforeHeading then
10912 parsers.headerstart = parsers.fail
10913 else
10914 parsers.headerstart = parsers.atx_heading
10915 end
10916
10917 if options.blankBeforeList then
10918 parsers.interrupting_bullets = parsers.fail
10919 parsers.interrupting_enumerators = parsers.fail
10920 else
10921 parsers.interrupting_bullets
10922 = parsers.bullet(parsers.dash, true)
10923 + parsers.bullet(parsers.asterisk, true)
10924 + parsers.bullet(parsers.plus, true)
10925
10926 parsers.interrupting_enumerators
10927 = parsers.enumerator(parsers.period, true)
10928 + parsers.enumerator(parsers.rparent, true)
10929 end
10930
10931 if options.html then
10932 parsers.html_interrupting
10933 = parsers.check_trail
10934 * (parsers.html_incomplete_open_tag
10935 + parsers.html_incomplete_close_tag
10936 + parsers.html_incomplete_open_special_tag
10937 + parsers.html_comment_start
```

```

10938 + parsers.html_cdatasection_start
10939 + parsers.html_declaration_start
10940 + parsers.html_instruction_start
10941 - parsers.html_close_special_tag
10942 - parsers.html_empty_special_tag)
10943 else
10944 parsers.html_interrupting = parsers.fail
10945 end
10946
10947 if options.blankBeforeHtmlBlock then
10948 parsers.html_interrupting = parsers.fail
10949 end
10950
10951 parsers.ListStarter = parsers.starter
10952
10953 parsers.EndlineExceptions
10954 = parsers.blankline -- paragraph break
10955 + parsers.eof -- end of document
10956 + parsers.bqstart
10957 + parsers.thematic_break_lines
10958 + parsers.interrupting_bullets
10959 + parsers.interrupting_enumerators
10960 + parsers.headerstart
10961 + parsers.html_interrupting
10962
10963 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
10964
10965 parsers.endline = parsers.newline
10966 * (parsers.check_minimal_indent
10967 * -V("EndlineExceptions")
10968 + parsers.check_optional_indent
10969 * -V("EndlineExceptions")
10970 * -V("ListStarter")) / function(_) return end
10971 * parsers.spacechar^0
10972
10973 parsers.Endline = parsers.endline
10974 / writer.soft_line_break
10975
10976 parsers.EndlineNoSub = parsers.endline
10977
10978 parsers.NoSoftLineBreakEndline
10979 = parsers.newline
10980 * (parsers.check_minimal_indent
10981 * -V("NoSoftLineBreakEndlineExceptions")
10982 + parsers.check_optional_indent
10983 * -V("NoSoftLineBreakEndlineExceptions")
10984 * -V("ListStarter"))

```

```

10985 * parsers.spacechar^0
10986 / writer.space
10987
10988 parsers.EndlineBreak = parsers.backslash * parsers.endline
10989 / writer.hard_line_break
10990
10991 parsers.OptionalIndent
10992 = parsers.spacechar^1 / writer.space
10993
10994 parsers.Space = parsers.spacechar^2 * parsers.endline
10995 / writer.hard_line_break
10996 + parsers.spacechar^1
10997 * parsers.endline^-1
10998 * parsers.eof / self.expandtabs
10999 + parsers.spacechar^1 * parsers.endline
11000 / writer.soft_line_break
11001 + parsers.spacechar^1
11002 * -parsers.newline / self.expandtabs
11003 + parsers.spacechar^1
11004
11005 parsers.NoSoftLineBreakSpace
11006 = parsers.spacechar^2 * parsers.endline
11007 / writer.hard_line_break
11008 + parsers.spacechar^1
11009 * parsers.endline^-1
11010 * parsers.eof / self.expandtabs
11011 + parsers.spacechar^1 * parsers.endline
11012 / writer.soft_line_break
11013 + parsers.spacechar^1
11014 * -parsers.newline / self.expandtabs
11015 + parsers.spacechar^1
11016
11017 parsers.NonbreakingEndline
11018 = parsers.endline
11019 / writer.nbsp
11020
11021 parsers.NonbreakingSpace
11022 = parsers.spacechar^2 * parsers.endline
11023 / writer.nbsp
11024 + parsers.spacechar^1
11025 * parsers.endline^-1 * parsers.eof / ""
11026 + parsers.spacechar^1 * parsers.endline
11027 * parsers.optionalspace
11028 / writer.nbsp
11029 + parsers.spacechar^1 * parsers.optionalspace
11030 / writer.nbsp
11031

```

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.

```
11032 function self.auto_link_url(url, attributes)
11033 return writer.link(writer.escape(url),
11034 url, nil, attributes)
11035 end
```

The `reader->auto_link_email` method produces an autolink to an e-mail in the output format, where `email` is the email address destination and `attributes` are the optional attributes.

```
11036 function self.auto_link_email(email, attributes)
11037 return writer.link(writer.escape(email),
11038 "mailto:".email,
11039 nil, attributes)
11040 end
11041
11042 parsers.AutoLinkUrl = parsers.auto_link_url
11043 / self.auto_link_url
11044
11045 parsers.AutoLinkEmail
11046 = parsers.auto_link_email
11047 / self.auto_link_email
11048
11049 parsers.AutoLinkRelativeReference
11050 = parsers.auto_link_relative_reference
11051 / self.auto_link_url
11052
11053 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
11054 / self.defer_link_and_emphasis_processing
11055
11056 parsers.EscapedChar = parsers.backslash
11057 * C(parsers.escapable) / writer.string
11058
11059 parsers.InlineHtml = Cs(parsers.html_inline_comment)
11060 / writer.inline_html_comment
11061 + Cs(parsers.html_any_empty_inline_tag
11062 + parsers.html_inline_instruction
11063 + parsers.html_inline_cdatasection
11064 + parsers.html_inline_declaration
11065 + parsers.html_any_open_inline_tag
11066 + parsers.html_any_close_tag)
11067 / writer.inline_html_tag
11068
11069 parsers.HtmlEntity = parsers.html_entities / writer.string
```



### 3.1.6.9 Block Elements (local)

```
11070 parsers.DisplayHtml = Cs(parsers.check_trail
11071 * (parsers.html_comment
11072 + parsers.html_special_block
11073 + parsers.html_block
11074 + parsers.html_any_block
11075 + parsers.html_instruction
11076 + parsers.html_cdatasection
11077 + parsers.html_declaration))
11078 / writer.block_html_element
11079
11080 parsers.indented_non_blank_line = parsers.indentedline
11081 - parsers.blankline
11082
11083 parsers.Verbatim
11084 = Cs(parsers.check_code_trail
11085 * (parsers.line - parsers.blankline)
11086 * ((parsers.check_minimal_blank_indent_and_full_code_trail
11087 * parsers.blankline)^0
11088 * ((parsers.check_minimal_indent / "")
11089 * parsers.check_code_trail
11090 * (parsers.line - parsers.blankline))^1)^0)
11091 / self.expandtabs / writer.verbatim
11092
11093 parsers.Blockquote = parsers.blockquote_body
11094 / writer.blockquote
11095
11096 parsers.ThematicBreak = parsers.thematic_break_lines
11097 / writer.thematic_break
11098
11099 parsers.Reference = parsers.define_reference_parser
11100 / self.register_link
11101
11102 parsers.Paragraph = parsers.freeze_trail
11103 * (Ct((parsers.Inline)^1)
11104 * (parsers.newline + parsers.eof)
11105 * parsers.unfreeze_trail
11106 / writer.paragraph)
11107
11108 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
11109 / writer.plain
```

### 3.1.6.10 Lists (local)

```
11110
11111 if options.taskLists then
11112 parsers.tickbox = (parsers.ticked_box
```

```

11113 + parsers.halfticked_box
11114 + parsers.unticked_box
11115) / writer.tickbox
11116 else
11117 parsers.tickbox = parsers.fail
11118 end
11119
11120 parsers.list_blank = parsers.conditionally_indented_blankline
11121
11122 parsers.ref_or_block_list_separated
11123 = parsers.sep_group_no_output(parsers.list_blank)
11124 * parsers.minimally_indented_ref
11125 + parsers.block_sep_group(parsers.list_blank)
11126 * parsers.minimally_indented_block
11127
11128 parsers.ref_or_block_non_separated
11129 = parsers.minimally_indented_ref
11130 + (parsers.succeed / writer.interblocksep)
11131 * parsers.minimally_indented_block
11132 - parsers.minimally_indented_blankline
11133
11134 parsers.tight_list_loop_body_pair =
11135 parsers.create_loop_body_pair(
11136 parsers.ref_or_block_non_separated,
11137 parsers.minimally_indented_par_or_plain_no_blank,
11138 (parsers.succeed / writer.interblocksep),
11139 (parsers.succeed / writer.paragraphsep))
11140
11141 parsers.loose_list_loop_body_pair =
11142 parsers.create_loop_body_pair(
11143 parsers.ref_or_block_list_separated,
11144 parsers.minimally_indented_par_or_plain,
11145 parsers.block_sep_group(parsers.list_blank),
11146 parsers.par_sep_group(parsers.list_blank))
11147
11148 parsers.tight_list_content_loop
11149 = V("Block")
11150 * parsers.tight_list_loop_body_pair.block^0
11151 + (V("Paragraph") + V("Plain"))
11152 * parsers.ref_or_block_non_separated
11153 * parsers.tight_list_loop_body_pair.block^0
11154 + (V("Paragraph") + V("Plain"))
11155 * parsers.tight_list_loop_body_pair.par^0
11156
11157 parsers.loose_list_content_loop
11158 = V("Block")
11159 * parsers.loose_list_loop_body_pair.block^0

```

```

11160 + (V("Paragraph") + V("Plain"))
11161 * parsers.ref_or_block_list_separated
11162 * parsers.loose_list_loop_body_pair.block^0
11163 + (V("Paragraph") + V("Plain"))
11164 * parsers.loose_list_loop_body_pair.par^0
11165
11166 parsers.list_item_tightness_condition
11167 = -#(parsers.list_blank^0
11168 * parsers.minimally_indented_ref_or_block_or_par)
11169 * remove_indent("li")
11170 + remove_indent("li")
11171 * parsers.fail
11172
11173 parsers.indented_content_tight
11174 = Ct((parsers.blankline / "")
11175 * #parsers.list_blank
11176 * remove_indent("li")
11177 + ((V("Reference") + (parsers.blankline / ""))
11178 * parsers.check_minimal_indent
11179 * parsers.tight_list_content_loop
11180 + (V("Reference") + (parsers.blankline / ""))
11181 + (parsers.tickbox^-1 / writer.escape)
11182 * parsers.tight_list_content_loop
11183)
11184 * parsers.list_item_tightness_condition)
11185
11186 parsers.indented_content_loose
11187 = Ct((parsers.blankline / "")
11188 * #parsers.list_blank
11189 + ((V("Reference") + (parsers.blankline / ""))
11190 * parsers.check_minimal_indent
11191 * parsers.loose_list_content_loop
11192 + (V("Reference") + (parsers.blankline / ""))
11193 + (parsers.tickbox^-1 / writer.escape)
11194 * parsers.loose_list_content_loop))
11195
11196 parsers.TightListItem = function(starter)
11197 return -parsers.ThematicBreak
11198 * parsers.add_indent(starter, "li")
11199 * parsers.indented_content_tight
11200 end
11201
11202 parsers.LooseListItem = function(starter)
11203 return -parsers.ThematicBreak
11204 * parsers.add_indent(starter, "li")
11205 * parsers.indented_content_loose
11206 * remove_indent("li")

```

```

11207 end
11208
11209 parsers.BulletListOfType = function(bullet_type)
11210 local bullet = parsers.bullet(bullet_type)
11211 return (Ct(parsers.TightListItem(bullet)
11212 * ((parsers.check_minimal_indent / "")
11213 * parsers.TightListItem(bullet)
11214)^0
11215)
11216 * Cc(true)
11217 * -#((parsers.list_blank^0 / "")
11218 * parsers.check_minimal_indent
11219 * (bullet - parsers.ThematicBreak)
11220)
11221 + Ct(parsers.LooseListItem(bullet)
11222 * ((parsers.list_blank^0 / "")
11223 * (parsers.check_minimal_indent / "")
11224 * parsers.LooseListItem(bullet)
11225)^0
11226)
11227 * Cc(false)
11228) / writer.bulletlist
11229 end
11230
11231 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
11232 + parsers.BulletListOfType(parsers.asterisk)
11233 + parsers.BulletListOfType(parsers.plus)
11234
11235 local function ordered_list(items,tight,starter)
11236 local startnum = starter[2][1]
11237 if options.startNumber then
11238 startnum = tonumber(startnum) or 1 -- fallback for '#'
11239 if startnum ~= nil then
11240 startnum = math.floor(startnum)
11241 end
11242 else
11243 startnum = nil
11244 end
11245 return writer.orderedlist(items,tight,startnum)
11246 end
11247
11248 parsers.OrderedListOfType = function(delimiter_type)
11249 local enumerator = parsers.enumerator(delimiter_type)
11250 return Cg(enumerator, "listtype")
11251 * (Ct(parsers.TightListItem(Cb("listtype"))
11252 * ((parsers.check_minimal_indent / "")
11253 * parsers.TightListItem(enumerator))^0

```

```

11254 * Cc(true)
11255 * -#((parsers.list_blank^0 / "")
11256 * parsers.check_minimal_indent * enumerator)
11257 + Ct(parsers.LooseListItem(Cb("listtype")))
11258 * ((parsers.list_blank^0 / "")
11259 * (parsers.check_minimal_indent / "")
11260 * parsers.LooseListItem(enumerator))^0)
11261 * Cc(false)
11262) * Ct(Cb("listtype")) / ordered_list
11263 end
11264
11265 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
11266 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

11267 parsers.Blank = parsers.blankline / ""
11268 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

11269 function parsers.parse_heading_text(s)
11270 local inlines = self.parser_functions.parse_inlines(s)
11271 local flatten_inlines = self.writer.flatten_inlines
11272 self.writer.flatten_inlines = true
11273 local flat_text = self.parser_functions.parse_inlines(s)
11274 flat_text = util.ropetostring(flat_text)
11275 self.writer.flatten_inlines = flatten_inlines
11276 return {flat_text, inlines}
11277 end
11278
11279 -- parse atx header
11280 parsers.AtxHeading = parsers.check_trail_no_rem
11281 * Cg(parsers.heading_start, "level")
11282 * ((C(parsers.optionalspace
11283 * parsers.hash^0
11284 * parsers.optionalspace
11285 * parsers.newline)
11286 + parsers.spacechar^1
11287 * C(parsers.line))
11288 / strip_atx_end
11289 / parsers.parse_heading_text)
11290 * Cb("level")
11291 / writer.heading
11292
11293 parsers.heading_line = parsers.linechar^1
11294 - parsers.thematic_break_lines
11295

```

```

11296 parsers.heading_text = parsers.heading_line
11297 * ((V("Endline") / "\n")
11298 * (parsers.heading_line
11299 - parsers.heading_level))^0
11300 * parsers.newline^-1
11301
11302 parsers.SettextHeading = parsers.freeze_trail
11303 * parsers.check_trail_no_rem
11304 * #(parsers.heading_text
11305 * parsers.check_minimal_indent
11306 * parsers.check_trail
11307 * parsers.heading_level)
11308 * Cs(parsers.heading_text)
11309 / parsers.parse_heading_text
11310 * parsers.check_minimal_indent_and_trail
11311 * parsers.heading_level
11312 * parsers.newline
11313 * parsers.unfreeze_trail
11314 / writer.heading
11315
11316 parsers.Heading = parsers.AtxHeading + parsers.SettextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain TeX output.

```

11317 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.

```

11318 local walkable_syntax = (function(global_walkable_syntax)
11319 local local_walkable_syntax = {}
11320 for lhs, rule in pairs(global_walkable_syntax) do
11321 local_walkable_syntax[lhs] = util.table_copy(rule)
11322 end
11323 return local_walkable_syntax
11324 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.

```

11325 local current_extension_name = nil
11326 self.insert_pattern = function(selector, pattern, pattern_name)

```

```

11327 assert(pattern_name == nil or type(pattern_name) == "string")
11328 local _, _, lhs, pos, rhs
11329 = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
11330 assert(lhs ~= nil,
11331 [[Expected selector in form]]
11332 .. [[LHS (before|after|instead of) RHS", not "]]
11333 .. selector .. [[]])
11334 assert(walkable_syntax[lhs] ~= nil,
11335 [[Rule]] .. lhs
11336 .. [[-> ... does not exist in markdown grammar]])
11337 assert(pos == "before" or pos == "after" or pos == "instead of",
11338 [[Expected positional specifier "before", "after",]]
11339 .. [[or "instead of", not "]]
11340 .. pos .. [[]])
11341 local rule = walkable_syntax[lhs]
11342 local index = nil
11343 for current_index, current_rhs in ipairs(rule) do
11344 if type(current_rhs) == "string" and current_rhs == rhs then
11345 index = current_index
11346 if pos == "after" then
11347 index = index + 1
11348 end
11349 break
11350 end
11351 end
11352 assert(index ~= nil,
11353 [[Rule]] .. lhs .. [[->]] .. rhs
11354 .. [[does not exist in markdown grammar]])
11355 local accountable_pattern
11356 if current_extension_name then
11357 accountable_pattern
11358 = {pattern, current_extension_name, pattern_name}
11359 else
11360 assert(type(pattern) == "string",
11361 [[reader->insert_pattern() was called outside]]
11362 .. [[an extension with]]
11363 .. [[a PEG pattern instead of a rule name]])
11364 accountable_pattern = pattern
11365 end
11366 if pos == "instead of" then
11367 rule[index] = accountable_pattern
11368 else
11369 table.insert(rule, index, accountable_pattern)
11370 end
11371 -- TODO: Remove all occurrences of `pattern` after `index`
11372 -- to improve speed?
11373 end

```

```

11374 if options.htmlOverLinks then
11375 self.insert_pattern("Inline before AutoLinkUrl", "InlineHtml")
11376 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.

```

11377 local syntax =
11378 { "Blocks",
11379
11380 Blocks = V("InitializeState")
11381 * V("ExpectedJekyllData")
11382 * V("Blank")^0

```

Only create interblock separators between pairs of blocks that are not both paragraphs. Between a pair of paragraphs, any number of blank lines will always produce a paragraph separator.

```

11383 * (V("Block")
11384 * (V("Blank")^0 * parsers.eof
11385 + (V("Blank")^2 / writer.paragraphsep
11386 + V("Blank")^0 / writer.interblocksep
11387)
11388)
11389 + (V("Paragraph") + V("Plain"))
11390 * (V("Blank")^0 * parsers.eof
11391 + (V("Blank")^2 / writer.paragraphsep
11392 + V("Blank")^0 / writer.interblocksep
11393)
11394)
11395 * V("Block")
11396 * (V("Blank")^0 * parsers.eof
11397 + (V("Blank")^2 / writer.paragraphsep
11398 + V("Blank")^0 / writer.interblocksep
11399)
11400)
11401 + (V("Paragraph") + V("Plain"))
11402 * (V("Blank")^0 * parsers.eof
11403 + V("Blank")^0 / writer.paragraphsep
11404)
11405)^0,
11406
11407 ExpectedJekyllData = parsers.succeed,
11408
11409 Blank = parsers.Blank,
11410 Reference = parsers.Reference,
11411
11412 Blockquote = parsers.Blockquote,
11413 Verbatim = parsers.Verbatim,
11414 ThematicBreak = parsers.ThematicBreak,

```



```

11415 BulletList = parsers.BulletList,
11416 OrderedList = parsers.OrderedList,
11417 DisplayHtml = parsers.DisplayHtml,
11418 Heading = parsers.Heading,
11419 Paragraph = parsers.Paragraph,
11420 Plain = parsers.Plain,
11421
11422 ListStarter = parsers.ListStarter,
11423 EndlineExceptions = parsers.EndlineExceptions,
11424 NoSoftLineBreakEndlineExceptions
11425 = parsers.NoSoftLineBreakEndlineExceptions,
11426
11427 Str = parsers.Str,
11428 Space = parsers.Space,
11429 NoSoftLineBreakSpace
11430 = parsers.NoSoftLineBreakSpace,
11431 OptionalIndent = parsers.OptionalIndent,
11432 Endline = parsers.Endline,
11433 EndlineNoSub = parsers.EndlineNoSub,
11434 NoSoftLineBreakEndline
11435 = parsers.NoSoftLineBreakEndline,
11436 EndlineBreak = parsers.EndlineBreak,
11437 LinkAndEmph = parsers.LinkAndEmph,
11438 Code = parsers.Code,
11439 AutoLinkUrl = parsers.AutoLinkUrl,
11440 AutoLinkEmail = parsers.AutoLinkEmail,
11441 AutoLinkRelativeReference
11442 = parsers.AutoLinkRelativeReference,
11443 InlineHtml = parsers.InlineHtml,
11444 HtmlEntity = parsers.HtmlEntity,
11445 EscapedChar = parsers.EscapedChar,
11446 Smart = parsers.Smart,
11447 Symbol = parsers.Symbol,
11448 SpecialChar = parsers.fail,
11449 InitializeState = parsers.succeed,
11450 }

```

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]`.

```

11451 self.update_rule = function(rule_name, get_pattern)
11452 assert(current_extension_name ~= nil)
11453 assert(syntax[rule_name] ~= nil,
11454 [[Rule]] .. rule_name
11455 .. [[-> ... does not exist in markdown grammar]])

```

```

11456 local previous_pattern
11457 local extension_name
11458 if walkable_syntax[rule_name] then
11459 local previous_accountable_pattern
11460 = walkable_syntax[rule_name][1]
11461 previous_pattern = previous_accountable_pattern[1]
11462 extension_name
11463 = previous_accountable_pattern[2]
11464 .. ", " .. current_extension_name
11465 else
11466 previous_pattern = nil
11467 extension_name = current_extension_name
11468 end
11469 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

```

```

11470 if type(get_pattern) == "function" then
11471 pattern = get_pattern(previous_pattern)
11472 else
11473 assert(previous_pattern == nil,
11474 [[Rule]] .. rule_name ..
11475 [[has already been updated by]] .. extension_name)
11476 pattern = get_pattern
11477 end
11478 local accountable_pattern = { pattern, extension_name, rule_name }
11479 walkable_syntax[rule_name] = { accountable_pattern }
11480 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.

```

11481 local special_characters = {}
11482 self.add_special_character = function(c)
11483 table.insert(special_characters, c)
11484 syntax.SpecialChar = S(table.concat(special_characters, ""))
11485 end
11486
11487 self.add_special_character("*")

```

```

11488 self.add_special_character("[")
11489 self.add_special_character("]")
11490 self.add_special_character("<")
11491 self.add_special_character("!")
11492 self.add_special_character("\\")

```

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

```

11493 self.initialize_named_group = function(name, value)
11494 local pattern = Ct("")
11495 if value ~= nil then
11496 pattern = pattern / value
11497 end
11498 syntax.InitializeState = syntax.InitializeState
11499 * Cg(pattern, name)
11500 end

```

Add a named group for indentation.

```

11501 self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

11502 for _, extension in ipairs(extensions) do
11503 current_extension_name = extension.name
11504 extension.extend_writer(writer)
11505 extension.extend_reader(self)
11506 end
11507 current_extension_name = nil

```

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

```

11508 if options.debugExtensions then
11509 local sorted_lhs = {}
11510 for lhs, _ in pairs(walkable_syntax) do
11511 table.insert(sorted_lhs, lhs)
11512 end
11513 table.sort(sorted_lhs)
11514
11515 local output_lines = {"{"}
11516 for lhs_index, lhs in ipairs(sorted_lhs) do
11517 local encoded_lhs = util.encode_json_string(lhs)
11518 table.insert(output_lines, [[]] .. encoded_lhs .. [[:]])
11519 local rule = walkable_syntax[lhs]
11520 for rhs_index, rhs in ipairs(rule) do
11521 local human_readable_rhs
11522 if type(rhs) == "string" then
11523 human_readable_rhs = rhs
11524 else
11525 local pattern_name

```

```

11526 if rhs[3] then
11527 pattern_name = rhs[3]
11528 else
11529 pattern_name = "Anonymous Pattern"
11530 end
11531 local extension_name = rhs[2]
11532 human_readable_rhs = pattern_name .. [[(]]
11533 .. extension_name .. [[]]]
11534 end
11535 local encoded_rhs
11536 = util.encode_json_string(human_readable_rhs)
11537 local output_line = [[]] .. encoded_rhs
11538 if rhs_index < #rule then
11539 output_line = output_line .. ","
11540 end
11541 table.insert(output_lines, output_line)
11542 end
11543 local output_line = "]"
11544 if lhs_index < #sorted_lhs then
11545 output_line = output_line .. ","
11546 end
11547 table.insert(output_lines, output_line)
11548 end
11549 table.insert(output_lines, "}")
11550
11551 local output = table.concat(output_lines, "\n")
11552 local output_filename = options.debugExtensionsFileName
11553 local output_file = assert(io.open(output_filename, "w"),
11554 [[Could not open file]] .. output_filename
11555 .. [[for writing]])
11556 assert(output_file:write(output))
11557 assert(output_file:close())
11558 end

```

Materialize `walkable_syntax` and merge it into `syntax` to produce the complete PEG grammar of markdown. Whenever a rule exists in both `walkable_syntax` and `syntax`, the rule from `walkable_syntax` overrides the rule from `syntax`.

```

11559 for lhs, rule in pairs(walkable_syntax) do
11560 syntax[lhs] = parsers.fail
11561 for _, rhs in ipairs(rule) do
11562 local pattern

```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

11563 if type(rhs) == "string" then

```

```

11564 pattern = V(rhs)
11565 else
11566 pattern = rhs[1]
11567 if type(pattern) == "string" then
11568 pattern = V(pattern)
11569 end
11570 end
11571 syntax[lhs] = syntax[lhs] + pattern
11572 end
11573 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.

```

11574 if options.underscores then
11575 self.add_special_character("_")
11576 end
11577
11578 if not options.codeSpans then
11579 syntax.Code = parsers.fail
11580 else
11581 self.add_special_character("`")
11582 end
11583
11584 if not options.html then
11585 syntax.DisplayHtml = parsers.fail
11586 syntax.InlineHtml = parsers.fail
11587 syntax.HtmlEntity = parsers.fail
11588 else
11589 self.add_special_character("&")
11590 end
11591
11592 if options.preserveTabs then
11593 options.stripIndent = false
11594 end
11595
11596 if not options.smartEllipses then
11597 syntax.Smart = parsers.fail
11598 else
11599 self.add_special_character(".")
11600 end
11601
11602 if not options.relativeReferences then
11603 syntax.AutoLinkRelativeReference = parsers.fail
11604 end
11605
11606 if options.contentLevel == "inline" then

```

```

11607 syntax[1] = "Inlines"
11608 syntax.Inlines = V("InitializeState")
11609 * parsers.Inline^0
11610 * (parsers.spacing^0
11611 * parsers.eof / "")
11612 syntax.Space = parsers.Space + parsers.blankline / writer.space
11613 end
11614
11615 local blocks_nested_t = util.table_copy(syntax)
11616 blocks_nested_t.ExpectedJekyllData = parsers.succeed
11617 parsers.blocks_nested = Ct(blocks_nested_t)
11618
11619 parsers.blocks = Ct(syntax)
11620
11621 local inlines_t = util.table_copy(syntax)
11622 inlines_t[1] = "Inlines"
11623 inlines_t.Inlines = V("InitializeState")
11624 * parsers.Inline^0
11625 * (parsers.spacing^0
11626 * parsers.eof / "")
11627 parsers.inlines = Ct(inlines_t)
11628
11629 local inlines_no_inline_note_t = util.table_copy(inlines_t)
11630 inlines_no_inline_note_t.InlineNote = parsers.fail
11631 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
11632
11633 local inlines_no_html_t = util.table_copy(inlines_t)
11634 inlines_no_html_t.DisplayHtml = parsers.fail
11635 inlines_no_html_t.InlineHtml = parsers.fail
11636 inlines_no_html_t.HtmlEntity = parsers.fail
11637 parsers.inlines_no_html = Ct(inlines_no_html_t)
11638
11639 local inlines_nbsp_t = util.table_copy(inlines_t)
11640 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
11641 inlines_nbsp_t.Space = parsers.NonbreakingSpace
11642 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
11643
11644 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
11645 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
11646 inlines_no_link_or_emphasis_t.EndlineExceptions
11647 = parsers.EndlineExceptions - parsers.eof
11648 parsers.inlines_no_link_or_emphasis
11649 = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain TeX output and returns it..

```

11650 return function(input)

```

Unicode-normalize the input.

```
11651 if options.unicodeNormalization then
11652 local form = options.unicodeNormalizationForm
11653 input = util.normalize(input, form)
11654 end
```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```
11655 input = input:gsub("\r\n?", "\n")
11656 if input:sub(-1) ~= "\n" then
11657 input = input .. "\n"
11658 end
```

Clear the table of references.

```
11659 references = {}
11660 local document = self.parser_functions.parse_blocks(input)
11661 local output = util.ropetostring(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
11662 local undosep_start, undosep_end
11663 local potential_secend_start, secend_start
11664 local potential_sep_start, sep_start
11665 while true do
11666 -- find a `writer->undosep`
11667 undosep_start, undosep_end
11668 = output:find(writer.undosep_text, 1, true)
11669 if undosep_start == nil then break end
11670 -- skip any preceding section ends
11671 secend_start = undosep_start
11672 while true do
11673 potential_secend_start = secend_start - #writer.secend_text
11674 if potential_secend_start < 1
11675 or output:sub(potential_secend_start,
11676 secend_start - 1) ~= writer.secend_text
11677 then
11678 break
11679 end
11680 secend_start = potential_secend_start
11681 end
11682 -- find an immediately preceding
11683 -- block element / paragraph separator
11684 sep_start = secend_start
11685 potential_sep_start = sep_start - #writer.interblocksep_text
11686 if potential_sep_start >= 1
11687 and output:sub(potential_sep_start,
11688 sep_start - 1) == writer.interblocksep_text
```

```

11689 then
11690 sep_start = potential_sep_start
11691 else
11692 potential_sep_start = sep_start - #writer.paragraphsep_text
11693 if potential_sep_start >= 1
11694 and output:sub(potential_sep_start,
11695 sep_start - 1) == writer.paragraphsep_text
11696 then
11697 sep_start = potential_sep_start
11698 end
11699 end
11700 -- remove `writer->undosep` and immediately preceding
11701 -- block element / paragraph separator
11702 output = output:sub(1, sep_start - 1)
11703 .. output:sub(secend_start, undosep_start - 1)
11704 .. output:sub(undosep_end + 1)
11705 end
11706 return output
11707 end
11708 end
11709 return self
11710 end

```

### 3.1.7 Built-In Syntax Extensions

Create `extensions` hash table that contains built-in syntax extensions. Syntax extensions are functions that produce objects with two methods: `extend_writer` and `extend_reader`. The `extend_writer` object takes a `writer` object as the only parameter and mutates it. Similarly, `extend_reader` takes a `reader` object as the only parameter and mutates it.

```

11711 M.extensions = {}

```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```

11712 M.extensions.bracketed_spans = function()
11713 return {
11714 name = "built-in bracketed_spans syntax extension",
11715 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.

```

11716 function self.span(s, attr)
11717 if self.flatten_inlines then return s end
11718 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
11719 self.attributes(attr),

```



```

11720 s,
11721 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"
11722 end
11723 end, extend_reader = function(self)
11724 local parsers = self.parsers
11725 local writer = self.writer
11726
11727 local span_label = parsers.lbracket
11728 * (Cs((parsers.alphanumeric^1
11729 + parsers.inticks
11730 + parsers.autolink
11731 + V("InlineHtml")
11732 + (parsers.backslash * parsers.backslash)
11733 + (parsers.backslash
11734 * (parsers.lbracket + parsers.rbracket)
11735 + V("Space") + V("Endline")
11736 + (parsers.any
11737 - (parsers.newline
11738 + parsers.lbracket
11739 + parsers.rbracket
11740 + parsers.blankline^2))))^1)
11741 / self.parser_functions.parse_inlines)
11742 * parsers.rbracket
11743
11744 local Span = span_label
11745 * Ct(parsers.attributes)
11746 / writer.span
11747
11748 self.insert_pattern("Inline before LinkAndEmph",
11749 Span, "Span")
11750 end
11751 }
11752 end

```

### 3.1.7.2 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```

11753 M.extensions.citations = function(citation_nbsps)
11754 return {
11755 name = "built-in citations syntax extension",
11756 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.

```

11757 function self.citations(text_cites, cites)
11758 local buffer = {}
11759 if self.flatten_inlines then
11760 for _,cite in ipairs(cites) do
11761 if cite.prenote then
11762 table.insert(buffer, {cite.prenote, " "})
11763 end
11764 table.insert(buffer, cite.name)
11765 if cite.postnote then
11766 table.insert(buffer, {" ", cite.postnote})
11767 end
11768 end
11769 else
11770 table.insert(buffer,
11771 {"\\markdownRenderer",
11772 text_cites and "TextCite" or "Cite",
11773 "{", #cites, "}"})
11774 for _,cite in ipairs(cites) do
11775 table.insert(buffer,
11776 {cite.suppress_author and "-" or "+", "{",
11777 cite.prenote or "", "}{" ,
11778 cite.postnote or "", "}{" , cite.name, "}"})
11779 end
11780 end
11781 return buffer
11782 end
11783 end, extend_reader = function(self)
11784 local parsers = self.parsers
11785 local writer = self.writer
11786
11787 local citation_chars
11788 = parsers.alphanumeric
11789 + S("#$%&-+<>~/_")
11790

```

```

11791 local citation_name
11792 = Cs(parsers.dash~-1) * parsers.at
11793 * Cs(citation_chars
11794 * (((citation_chars
11795 + parsers.internal_punctuation
11796 - parsers.comma - parsers.semicolon)
11797 * -#((parsers.internal_punctuation
11798 - parsers.comma
11799 - parsers.semicolon)^0
11800 * -(citation_chars
11801 + parsers.internal_punctuation
11802 - parsers.comma
11803 - parsers.semicolon)))^0
11804 * citation_chars)^-1)
11805
11806 local citation_body_prenote
11807 = Cs((parsers.alphanumeric~1
11808 + parsers.bracketed
11809 + parsers.inticks
11810 + parsers.autolink
11811 + V("InlineHtml")
11812 + V("Space") + V("EndlineNoSub")
11813 + (parsers.anyescaped
11814 - (parsers.newline
11815 + parsers.rbracket
11816 + parsers.blankline^2))
11817 - (parsers.spnl
11818 * parsers.dash~-1
11819 * parsers.at))^1)
11820
11821 local citation_body_postnote
11822 = Cs((parsers.alphanumeric~1
11823 + parsers.bracketed
11824 + parsers.inticks
11825 + parsers.autolink
11826 + V("InlineHtml")
11827 + V("Space") + V("EndlineNoSub")
11828 + (parsers.anyescaped
11829 - (parsers.newline
11830 + parsers.rbracket
11831 + parsers.semicolon
11832 + parsers.blankline^2))
11833 - (parsers.spnl * parsers.rbracket))^1)
11834
11835 local citation_body_chunk
11836 = (citation_body_prenote
11837 * parsers.spnlc_sep

```

```

11838 + Cc("")
11839 * parsers.spnlc
11840)
11841 * citation_name
11842 * (parsers.internal_punctuation
11843 - parsers.semicolon)^-1
11844 * (parsers.spnlc / function(_) return end
11845 * citation_body_postnote
11846 + Cc("")
11847 * parsers.spnlc
11848)
11849
11850 local citation_body
11851 = citation_body_chunk
11852 * (parsers.semicolon
11853 * parsers.spnlc
11854 * citation_body_chunk
11855)^0
11856
11857 local citation_headless_body_postnote
11858 = Cs((parsers.alphanumeric^1
11859 + parsers.bracketed
11860 + parsers.inticks
11861 + parsers.autolink
11862 + V("InlineHtml")
11863 + V("Space") + V("Endline")
11864 + (parsers.anyescaped
11865 - (parsers.newline
11866 + parsers.rbracket
11867 + parsers.at
11868 + parsers.semicolon + parsers.blankline^2))
11869 - (parsers.spnl * parsers.rbracket))^0)
11870
11871 local citation_headless_body
11872 = citation_headless_body_postnote
11873 * (parsers.semicolon
11874 * parsers.spnlc
11875 * citation_body_chunk
11876)^0
11877
11878 local citations
11879 = function(text_cites, raw_cites)
11880 local function normalize(str)
11881 if str == "" then
11882 str = nil
11883 else
11884 str = (citation_nbsps and

```

```

11885 self.parser_functions.parse_inlines_nbsp or
11886 self.parser_functions.parse_inlines)(str)
11887 end
11888 return str
11889 end
11890
11891 local cites = {}
11892 for i = 1,#raw_cites,4 do
11893 cites[#cites+1] = {
11894 prenote = normalize(raw_cites[i]),
11895 suppress_author = raw_cites[i+1] == "-",
11896 name = writer.identifier(raw_cites[i+2]),
11897 postnote = normalize(raw_cites[i+3]),
11898 }
11899 end
11900 return writer.citations(text_cites, cites)
11901 end
11902
11903 local TextCitations
11904 = Ct((parsers.spnlc
11905 * Cc("")
11906 * citation_name
11907 * ((parsers.spnlc
11908 * parsers.lbracket
11909 * citation_headless_body
11910 * parsers.rbracket) + Cc("")))~1)
11911 / function(raw_cites)
11912 return citations(true, raw_cites)
11913 end
11914
11915 local ParenthesizedCitations
11916 = Ct((parsers.spnlc
11917 * parsers.lbracket
11918 * citation_body
11919 * parsers.rbracket)^1)
11920 / function(raw_cites)
11921 return citations(false, raw_cites)
11922 end
11923
11924 local Citations = TextCitations + ParenthesizedCitations
11925
11926 self.insert_pattern("Inline before LinkAndEmph",
11927 Citations, "Citations")
11928
11929 self.add_special_character("@")
11930 self.add_special_character("-")
11931 end

```

```

11932 }
11933 end

```

### 3.1.7.3 Content Blocks

The `extensions.content_blocks` function implements the iA Writer content blocks syntax extension. The `language_map` parameter specifies the filename of the JSON file that maps filename extensions to programming language names.

```

11934 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.

```

11935 local languages_json = (function()
11936 local base, prev, curr
11937 for _, pathname in ipairs(util.find_files(language_map)) do
11938 local file = io.open(pathname, "r")
11939 if not file then goto continue end
11940 local input = assert(file:read("*a"))
11941 assert(file:close())
11942 local json = input:gsub('^[\n]-'):','[%1]=')
11943 curr = load("_ENV = {}; return "..json")()
11944 if type(curr) == "table" then
11945 if base == nil then
11946 base = curr
11947 else
11948 setmetatable(prev, { __index = curr })
11949 end
11950 prev = curr
11951 end
11952 ::continue::
11953 end
11954 return base or {}
11955 end)()
11956
11957 return {
11958 name = "built-in content_blocks syntax extension",
11959 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.

```

11960 function self.contentblock(src,suf,type,tit)
11961 if not self.is_writing then return "" end
11962 src = src..".."..suf

```

```

11963 suf = suf:lower()
11964 if type == "onlineimage" then
11965 return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
11966 "{" ,self.string(src),"} ",
11967 "{" ,self.uri(src),"} ",
11968 "{" ,self.string(tit or ""),"} "}
11969 elseif languages_json[suf] then
11970 return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
11971 "{" ,self.string(languages_json[suf]),"} ",
11972 "{" ,self.string(src),"} ",
11973 "{" ,self.uri(src),"} ",
11974 "{" ,self.string(tit or ""),"} "}
11975 else
11976 return {"\\markdownRendererContentBlock{" ,suf,"} ",
11977 "{" ,self.string(src),"} ",
11978 "{" ,self.uri(src),"} ",
11979 "{" ,self.string(tit or ""),"} "}
11980 end
11981 end
11982 end, extend_reader = function(self)
11983 local parsers = self.parsers
11984 local writer = self.writer
11985
11986 local contentblock_tail
11987 = parsers.optionalttitle
11988 * (parsers.newline + parsers.eof)
11989
11990 -- case insensitive online image suffix:
11991 local onlineimagesuffix
11992 = (function(...)
11993 local parser = nil
11994 for _, suffix in ipairs({...}) do
11995 local pattern=nil
11996 for i=1,#suffix do
11997 local char=suffix:sub(i,i)
11998 char = S(char:lower()..char:upper())
11999 if pattern == nil then
12000 pattern = char
12001 else
12002 pattern = pattern * char
12003 end
12004 end
12005 if parser == nil then
12006 parser = pattern
12007 else
12008 parser = parser + pattern
12009 end

```

```

12010 end
12011 return parser
12012 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
12013
12014 -- online image url for iA Writer content blocks with
12015 -- mandatory suffix, allowing nested brackets:
12016 local onlineimageurl
12017 = (parsers.less
12018 * Cs((parsers.anyescaped
12019 - parsers.more
12020 - parsers.spacing
12021 - #(parsers.period
12022 * onlineimagesuffix
12023 * parsers.more
12024 * contentblock_tail))^0)
12025 * parsers.period
12026 * Cs(onlineimagesuffix)
12027 * parsers.more
12028 + (Cs((parsers.inparens
12029 + (parsers.anyescaped
12030 - parsers.spacing
12031 - parsers.rparent
12032 - #(parsers.period
12033 * onlineimagesuffix
12034 * contentblock_tail))))^0)
12035 * parsers.period
12036 * Cs(onlineimagesuffix))
12037) * Cc("onlineimage")
12038
12039 -- filename for iA Writer content blocks with mandatory suffix:
12040 local localfilepath
12041 = parsers.slash
12042 * Cs((parsers.anyescaped
12043 - parsers.tab
12044 - parsers.newline
12045 - #(parsers.period
12046 * parsers.alphanumeric^1
12047 * contentblock_tail))^1)
12048 * parsers.period
12049 * Cs(parsers.alphanumeric^1)
12050 * Cc("localfile")
12051
12052 local ContentBlock
12053 = parsers.check_trail_no_rem
12054 * (localfilepath + onlineimageurl)
12055 * contentblock_tail
12056 / writer.contentblock

```



```

12057
12058 self.insert_pattern("Block before Blockquote",
12059 ContentBlock, "ContentBlock")
12060 end
12061 }
12062 end

```

### 3.1.7.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

12063 M.extensions.definition_lists = function(tight_lists)
12064 return {
12065 name = "built-in definition_lists syntax extension",
12066 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.

```

12067 local function dlitem(term, defs)
12068 local retVal = {"\\markdownRendererDlItem{",term,""}
12069 for _, def in ipairs(defs) do
12070 retVal[#retVal+1]
12071 = {"\\markdownRendererDlDefinitionBegin ",def,
12072 "\\markdownRendererDlDefinitionEnd "}
12073 end
12074 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
12075 return retVal
12076 end
12077
12078 function self.definitionlist(items,tight)
12079 if not self.is_writing then return "" end
12080 local buffer = {}
12081 for _,item in ipairs(items) do
12082 buffer[#buffer + 1] = dlitem(item.term, item.definitions)
12083 end
12084 if tight and tight_lists then
12085 return {"\\markdownRendererDlBeginTight\\n", buffer,
12086 "\\n\\markdownRendererDlEndTight"}
12087 else
12088 return {"\\markdownRendererDlBegin\\n", buffer,
12089 "\\n\\markdownRendererDlEnd"}
12090 end
12091 end
12092 end, extend_reader = function(self)

```

```

12093 local parsers = self.parsers
12094 local writer = self.writer
12095
12096 local defstartchar = S("~:")
12097
12098 local defstart
12099 = parsers.check_trail_length(0) * defstartchar
12100 * #parsers.spacing
12101 * (parsers.tab + parsers.space^-3)
12102 + parsers.check_trail_length(1)
12103 * defstartchar * #parsers.spacing
12104 * (parsers.tab + parsers.space^-2)
12105 + parsers.check_trail_length(2)
12106 * defstartchar * #parsers.spacing
12107 * (parsers.tab + parsers.space^-1)
12108 + parsers.check_trail_length(3)
12109 * defstartchar * #parsers.spacing
12110
12111 local indented_line
12112 = (parsers.check_minimal_indent / "")
12113 * parsers.check_code_trail * parsers.line
12114
12115 local blank
12116 = parsers.check_minimal_blank_indent_and_any_trail
12117 * parsers.optionalspace * parsers.newline
12118
12119 local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
12120
12121 local indented_blocks = function(bl)
12122 return Cs(bl
12123 * (blank^1 * (parsers.check_minimal_indent / "")
12124 * parsers.check_code_trail * -parsers.blankline * bl)^0
12125 * (blank^1 + parsers.eof))
12126 end
12127
12128 local function definition_list_item(term, defs, _)
12129 return { term = self.parser_functions.parse_inlines(term),
12130 definitions = defs }
12131 end
12132
12133 local DefinitionListItemLoose
12134 = C(parsers.line) * blank^0
12135 * Ct((parsers.check_minimal_indent * (defstart
12136 * indented_blocks(dlchunk)
12137 / self.parser_functions.parse_blocks_nested))^1)
12138 * Cc(false) / definition_list_item
12139

```

```

12140 local DefinitionListItemTight
12141 = C(parsers.line)
12142 * Ct((parsers.check_minimal_indent * (defstart * dlchunk
12143 / self.parser_functions.parse_blocks_nested))^1)
12144 * Cc(true) / definition_list_item
12145
12146 local DefinitionList
12147 = (Ct(DefinitionListItemLoose^1) * Cc(false)
12148 + Ct(DefinitionListItemTight^1)
12149 * (blank^0
12150 * -DefinitionListItemLoose * Cc(true))
12151) / writer.definitionlist
12152
12153 self.insert_pattern("Block after Heading",
12154 DefinitionList, "DefinitionList")
12155 end
12156 }
12157 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

12158 M.extensions.fancy_lists = function()
12159 return {
12160 name = "built-in fancy_lists syntax extension",
12161 extend_writer = function(self)
12162 local options = self.options
12163

```

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.

```

12164 function self.fancylist(items,tight,startnum,numstyle,numdelim)
12165 if not self.is_writing then return "" end
12166 local buffer = {}
12167 local num = startnum
12168 for _,item in ipairs(items) do
12169 if item ~= "" then
12170 buffer[#buffer + 1] = self.fancyitem(item,num)
12171 end
12172 if num ~= nil and item ~= "" then
12173 num = num + 1
12174 end
12175 end
12176 local contents = util.intersperse(buffer,"\n")
12177 if tight and options.tightLists then
12178 return {"\\markdownRendererFancyOlBeginTight{",
12179 numstyle,"}{",numdelim,"}",contents,
12180 "\\n\\markdownRendererFancyOlEndTight "}
12181 else
12182 return {"\\markdownRendererFancyOlBegin{",
12183 numstyle,"}{",numdelim,"}",contents,
12184 "\\n\\markdownRendererFancyOlEnd "}
12185 end
12186 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.

```

12187 function self.fancyitem(s,num)
12188 if num ~= nil then
12189 return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
12190 "\\markdownRendererFancyOlItemEnd "}
12191 else
12192 return {"\\markdownRendererFancyOlItem ",s,
12193 "\\markdownRendererFancyOlItemEnd "}
12194 end
12195 end
12196 end, extend_reader = function(self)
12197 local parsers = self.parsers
12198 local options = self.options
12199 local writer = self.writer
12200
12201 local function combine_markers_and_delims(markers, delims)
12202 local markers_table = {}
12203 for _,marker in ipairs(markers) do

```

```

12204 local start_marker
12205 local continuation_marker
12206 if type(marker) == "table" then
12207 start_marker = marker[1]
12208 continuation_marker = marker[2]
12209 else
12210 start_marker = marker
12211 continuation_marker = marker
12212 end
12213 for _,delim in ipairs(delims) do
12214 table.insert(markers_table,
12215 {start_marker, continuation_marker, delim})
12216 end
12217 end
12218 return markers_table
12219 end
12220
12221 local function join_table_with_func(func, markers_table)
12222 local pattern = func(table.unpack(markers_table[1]))
12223 for i = 2, #markers_table do
12224 pattern = pattern + func(table.unpack(markers_table[i]))
12225 end
12226 return pattern
12227 end
12228
12229 local lowercase_letter_marker = R("az")
12230 local uppercase_letter_marker = R("AZ")
12231
12232 local roman_marker = function(chars)
12233 local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
12234 local l, x, v, i
12235 = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
12236 return m^-3
12237 * (c*m + c*d + d^-1 * c^-3)
12238 * (x*c + x*l + l^-1 * x^-3)
12239 * (i*x + i*v + v^-1 * i^-3)
12240 end
12241
12242 local lowercase_roman_marker
12243 = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
12244 local uppercase_roman_marker
12245 = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
12246
12247 local lowercase_opening_roman_marker = P("i")
12248 local uppercase_opening_roman_marker = P("I")
12249
12250 local digit_marker = parsers.dig * parsers.dig^-8

```

```

12251
12252 local markers = {
12253 {lowercase_opening_roman_marker, lowercase_roman_marker},
12254 {uppercase_opening_roman_marker, uppercase_roman_marker},
12255 lowercase_letter_marker,
12256 uppercase_letter_marker,
12257 lowercase_roman_marker,
12258 uppercase_roman_marker,
12259 digit_marker
12260 }
12261
12262 local delims = {
12263 parsers.period,
12264 parsers.rparent
12265 }
12266
12267 local markers_table = combine_markers_and_delims(markers, delims)
12268
12269 local function enumerator(start_marker, _,
12270 delimiter_type, interrupting)
12271 local delimiter_range
12272 local allowed_end
12273 if interrupting then
12274 delimiter_range = P("1")
12275 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
12276 else
12277 delimiter_range = start_marker
12278 allowed_end = C(parsers.spacechar^1)
12279 + #(parsers.newline + parsers.eof)
12280 end
12281
12282 return parsers.check_trail
12283 * Ct(C(delimiter_range) * C(delimiter_type))
12284 * allowed_end
12285 end
12286
12287 local starter = join_table_with_func(enumerator, markers_table)
12288
12289 local TightListItem = function(starter)
12290 return parsers.add_indent(starter, "li")
12291 * parsers.indented_content_tight
12292 end
12293
12294 local LooseListItem = function(starter)
12295 return parsers.add_indent(starter, "li")
12296 * parsers.indented_content_loose
12297 * remove_indent("li")

```

```

12298 end
12299
12300 local function roman2number(roman)
12301 local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
12302 ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
12303 local numeral = 0
12304
12305 local i = 1
12306 local len = string.len(roman)
12307 while i < len do
12308 local z1, z2 = romans[string.sub(roman, i, i)],
12309 romans[string.sub(roman, i+1, i+1)]
12310 if z1 < z2 then
12311 numeral = numeral + (z2 - z1)
12312 i = i + 2
12313 else
12314 numeral = numeral + z1
12315 i = i + 1
12316 end
12317 end
12318 if i <= len then
12319 numeral = numeral + romans[string.sub(roman,i,i)]
12320 end
12321 return numeral
12322 end
12323
12324 local function sniffstyle(numstr, delimend)
12325 local numdelim
12326 if delimend == ")" then
12327 numdelim = "OneParen"
12328 elseif delimend == "." then
12329 numdelim = "Period"
12330 else
12331 numdelim = "Default"
12332 end
12333
12334 local num
12335 num = numstr:match("^([I])$")
12336 if num then
12337 return roman2number(num), "UpperRoman", numdelim
12338 end
12339 num = numstr:match("^([i])$")
12340 if num then
12341 return roman2number(string.upper(num)), "LowerRoman", numdelim
12342 end
12343 num = numstr:match("^([A-Z])$")
12344 if num then

```

```

12345 return string.byte(num) - string.byte("A") + 1,
12346 "UpperAlpha", numdelim
12347 end
12348 num = numstr:match("^([a-z])$")
12349 if num then
12350 return string.byte(num) - string.byte("a") + 1,
12351 "LowerAlpha", numdelim
12352 end
12353 num = numstr:match("^([IVXLCDM]+)")
12354 if num then
12355 return roman2number(num), "UpperRoman", numdelim
12356 end
12357 num = numstr:match("^([ivxlcdm]+)")
12358 if num then
12359 return roman2number(string.upper(num)), "LowerRoman", numdelim
12360 end
12361 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
12362 end
12363
12364 local function fancylist(items,tight,start)
12365 local startnum, numstyle, numdelim
12366 = sniffstyle(start[2][1], start[2][2])
12367 return writer.fancylist(items,tight,
12368 options.startNumber and startnum or 1,
12369 numstyle or "Decimal",
12370 numdelim or "Default")
12371 end
12372
12373 local FancyListOfType
12374 = function(start_marker, continuation_marker, delimiter_type)
12375 local enumerator_start
12376 = enumerator(start_marker, continuation_marker,
12377 delimiter_type)
12378 local enumerator_cont
12379 = enumerator(continuation_marker, continuation_marker,
12380 delimiter_type)
12381 return Cg(enumerator_start, "listtype")
12382 * (Ct(TightListItem(Cb("listtype"))
12383 * ((parsers.check_minimal_indent / "")
12384 * TightListItem(enumerator_cont))^0)
12385 * Cc(true)
12386 * -#((parsers.conditionally_indented_blankline^0 / "")
12387 * parsers.check_minimal_indent * enumerator_cont)
12388 + Ct(LooseListItem(Cb("listtype"))
12389 * ((parsers.conditionally_indented_blankline^0 / "")
12390 * (parsers.check_minimal_indent / "")
12391 * LooseListItem(enumerator_cont))^0)

```



```

12392 * Cc(false)
12393) * Ct(Cb("listtype")) / fancylist
12394 end
12395
12396 local FancyList
12397 = join_table_with_func(FancyListOfType, markers_table)
12398
12399 local ListStarter = starter
12400
12401 self.update_rule("OrderedList", FancyList)
12402 self.update_rule("ListStarter", ListStarter)
12403 end
12404 }
12405 end

```

### 3.1.7.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the syntax extension permits the specification of raw blocks using the Pandoc raw attribute syntax extension.

```

12406 M.extensions.fenced_code = function(blank_before_code_fence,
12407 allow_attributes,
12408 allow_raw_blocks)
12409 return {
12410 name = "built-in fenced_code syntax extension",
12411 extend_writer = function(self)
12412 local options = self.options
12413

```

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.

```

12414 function self.fencedCode(s, i, attr)
12415 if not self.is_writing then return "" end
12416 s = s:gsub("\n$", "")
12417 local buf = {}
12418 if attr ~= nil then
12419 table.insert(buf,
12420 {"\\markdownRendererFencedCodeAttributeContextBegin",
12421 self.attributes(attr)})
12422 end
12423 local name = util.cache_verbatim(options.cacheDir, s)
12424 table.insert(buf,

```

```

12425 {"\\markdownRendererInputFencedCode{" ,
12426 name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
12427 if attr ~= nil then
12428 table.insert(buf,
12429 "\\markdownRendererFencedCodeAttributeContextEnd{")
12430 end
12431 return buf
12432 end
12433

```

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

```

12434 if allow_raw_blocks then
12435 function self.rawBlock(s, attr)
12436 if not self.is_writing then return "" end
12437 s = s:gsub("\\n$", "")
12438 local name = util.cache_verbatim(options.cacheDir, s)
12439 return {"\\markdownRendererInputRawBlock{" ,
12440 name,"}{" , self.string(attr),"}"}
12441 end
12442 end
12443 end, extend_reader = function(self)
12444 local parsers = self.parsers
12445 local writer = self.writer
12446
12447 local function captures_geq_length(_,i,a,b)
12448 return #a >= #b and i
12449 end
12450
12451 local function strip_enclosing_whitespaces(str)
12452 return str:gsub("^%s*(.*)%s*$", "%1")
12453 end
12454
12455 local tilde_infostring = Cs(Cs((V("HtmlEntity")
12456 + parsers.anyescaped
12457 - parsers.newline)^0)
12458 / strip_enclosing_whitespaces)
12459
12460 local backtick_infostring
12461 = Cs(Cs((V("HtmlEntity")
12462 + (-#(parsers.backslash * parsers.backtick)
12463 * parsers.anyescaped)
12464 - parsers.newline
12465 - parsers.backtick)^0)
12466 / strip_enclosing_whitespaces)
12467
12468 local fenceindent

```

```

12469
12470 local function has_trail(indent_table)
12471 return indent_table ~= nil and
12472 indent_table.trail ~= nil and
12473 next(indent_table.trail) ~= nil
12474 end
12475
12476 local function has_indents(indent_table)
12477 return indent_table ~= nil and
12478 indent_table.indents ~= nil and
12479 next(indent_table.indents) ~= nil
12480 end
12481
12482 local function get_last_indent_name(indent_table)
12483 if has_indents(indent_table) then
12484 return indent_table.indents[#indent_table.indents].name
12485 end
12486 end
12487
12488 local count_fenced_start_indent =
12489 function(_, _, indent_table, trail)
12490 local last_indent_name = get_last_indent_name(indent_table)
12491 fenceindent = 0
12492 if last_indent_name ~= "li" then
12493 fenceindent = #trail
12494 end
12495 return true
12496 end
12497
12498 local fencehead = function(char, infostring)
12499 return Cmt(Cb("indent_info")
12500 * parsers.check_trail, count_fenced_start_indent)
12501 * Cg(char^3, "fencelength")
12502 * parsers.optionalspace
12503 * infostring
12504 * (parsers.newline + parsers.eof)
12505 end
12506
12507 local fencetail = function(char)
12508 return parsers.check_trail_no_rem
12509 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
12510 * parsers.optionalspace * (parsers.newline + parsers.eof)
12511 + parsers.eof
12512 end
12513
12514 local process_fenced_line =
12515 function(s, i, -- luacheck: ignore s i

```

```

12516 indent_table, line_content, is_blank)
12517 local remainder = ""
12518 if has_trail(indent_table) then
12519 remainder = indent_table.trail.internal_remainder
12520 end
12521
12522 if is_blank
12523 and get_last_indent_name(indent_table) == "li" then
12524 remainder = ""
12525 end
12526
12527 local str = remainder .. line_content
12528 local index = 1
12529 local remaining = fenceindent
12530
12531 while true do
12532 local c = str:sub(index, index)
12533 if c == " " and remaining > 0 then
12534 remaining = remaining - 1
12535 index = index + 1
12536 elseif c == "\t" and remaining > 3 then
12537 remaining = remaining - 4
12538 index = index + 1
12539 else
12540 break
12541 end
12542 end
12543
12544 return true, str:sub(index)
12545 end
12546
12547 local fencedline = function(char)
12548 return Cmt(Cb("indent_info")
12549 * C(parsers.line - fencetail(char))
12550 * Cc(false), process_fenced_line)
12551 end
12552
12553 local blankfencedline
12554 = Cmt(Cb("indent_info")
12555 * C(parsers.blankline)
12556 * Cc(true), process_fenced_line)
12557
12558 local TildeFencedCode
12559 = fencehead(parsers.tilde, tilde_infostring)
12560 * Cs((parsers.check_minimal_blank_indent / "")
12561 * blankfencedline
12562 + (parsers.check_minimal_indent / ""))

```

```

12563 * fencedline(parsers.tilde))^0)
12564 * ((parsers.check_minimal_indent / "")
12565 * fencetail(parsers.tilde) + parsers.succeed)
12566
12567 local BacktickFencedCode
12568 = fencehead(parsers.backtick, backtick_infostring)
12569 * Cs(((parsers.check_minimal_blank_indent / "")
12570 * blankfencedline
12571 + (parsers.check_minimal_indent / "")
12572 * fencedline(parsers.backtick))^0)
12573 * ((parsers.check_minimal_indent / "")
12574 * fencetail(parsers.backtick) + parsers.succeed)
12575
12576 local infostring_with_attributes
12577 = Ct(C((parsers.linechar
12578 - (parsers.optionalspace
12579 * parsers.attributes))^0)
12580 * parsers.optionalspace
12581 * Ct(parsers.attributes))
12582
12583 local FencedCode
12584 = ((TildeFencedCode + BacktickFencedCode)
12585 / function(infostring, code)
12586 local expanded_code = self.expandtabs(code)
12587
12588 if allow_raw_blocks then
12589 local raw_attr = lpeg.match(parsers.raw_attribute,
12590 infostring)
12591
12592 if raw_attr then
12593 return writer.rawBlock(expanded_code, raw_attr)
12594 end
12595 end
12596
12597 local attr = nil
12598 if allow_attributes then
12599 local match = lpeg.match(infostring_with_attributes,
12600 infostring)
12601
12602 if match then
12603 infostring, attr = table.unpack(match)
12604 end
12605 end
12606 return writer.fencedCode(expanded_code, infostring, attr)
12607 end)
12608
12609 self.insert_pattern("Block after Verbatim",
12610 FencedCode, "FencedCode")

```

```

12610 local fencestart
12611 if blank_before_code_fence then
12612 fencestart = parsers.fail
12613 else
12614 fencestart = fencehead(parsers.backtick, backtick_infostring)
12615 + fencehead(parsers.tilde, tilde_infostring)
12616 end
12617
12618 self.update_rule("EndlineExceptions", function(previous_pattern)
12619 if previous_pattern == nil then
12620 previous_pattern = parsers.EndlineExceptions
12621 end
12622 return previous_pattern + fencestart
12623 end)
12624
12625 self.add_special_character("`")
12626 self.add_special_character("~")
12627 end
12628 }
12629 end

```

### 3.1.7.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```

12630 M.extensions.fenced_divs = function(blank_before_div_fence)
12631 return {
12632 name = "built-in fenced_divs syntax extension",
12633 extend_writer = function(self)

```

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

```

12634 function self.div_begin(attributes)
12635 local start_output
12636 = {"\\markdownRenderFencedDivAttributeContextBegin\n",
12637 self.attributes(attributes)}
12638 local end_output
12639 = {"\\markdownRenderFencedDivAttributeContextEnd{}}"}
12640 return self.push_attributes(
12641 "div", attributes, start_output, end_output)
12642 end

```

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

```

12643 function self.div_end()

```

```

12644 return self.pop_attributes("div")
12645 end
12646 end, extend_reader = function(self)
12647 local parsers = self.parsers
12648 local writer = self.writer

```

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

```

12649 local fenced_div_infostring
12650 = Ct(parsers.attributes)
12651 + (
12652 C(parsers.attribute_classname
12653 - parsers.colon)^1
12654 / function (infostring)
12655 return {"." .. infostring}
12656 end
12657)
12658
12659 local fenced_div_begin = parsers.nonindentspace
12660 * parsers.colon^3
12661 * parsers.optionalspace
12662 * fenced_div_infostring
12663 * (parsers.spacechar^1
12664 * parsers.colon^1)^0
12665 * parsers.optionalspace
12666 * (parsers.newline + parsers.eof)
12667
12668 local fenced_div_end = parsers.nonindentspace
12669 * parsers.colon^3
12670 * parsers.optionalspace
12671 * (parsers.newline + parsers.eof)

```

Initialize a named group named `fenced_div_level` for tracking how deep we are nested in divs and the named group `fenced_div_num_opening_indents` for tracking the indent of the starting div fence. The former named group is immutable and should roll back properly when we fail to match a fenced div. The latter is mutable and may contain items from unsuccessful matches on top. However, we always know how many items at the head of the latter we can trust by consulting the former.

```

12672 self.initialize_named_group("fenced_div_level", "0")
12673 self.initialize_named_group("fenced_div_num_opening_indents")
12674
12675 local function increment_div_level()
12676 local push_indent_table =
12677 function(s, i, indent_table, -- luacheck: ignore s i
12678 fenced_div_num_opening_indents, fenced_div_level)
12679 fenced_div_level = tonumber(fenced_div_level) + 1
12680 local num_opening_indents = 0
12681 if indent_table.indents ~= nil then

```

```

12682 num_opening_indents = #indent_table.indents
12683 end
12684 fenced_div_num_opening_indents[fenced_div_level]
12685 = num_opening_indents
12686 return true, fenced_div_num_opening_indents
12687 end
12688
12689 local increment_level =
12690 function(s, i, fenced_div_level) -- luacheck: ignore s i
12691 fenced_div_level = tonumber(fenced_div_level) + 1
12692 return true, tostring(fenced_div_level)
12693 end
12694
12695 return Cg(Cmt(Cb("indent_info")
12696 * Cb("fenced_div_num_opening_indents")
12697 * Cb("fenced_div_level"), push_indent_table)
12698 , "fenced_div_num_opening_indents")
12699 * Cg(Cmt(Cb("fenced_div_level"), increment_level)
12700 , "fenced_div_level")
12701 end
12702
12703 local function decrement_div_level()
12704 local pop_indent_table =
12705 function(s, i, -- luacheck: ignore s i
12706 fenced_div_indent_table, fenced_div_level)
12707 fenced_div_level = tonumber(fenced_div_level)
12708 fenced_div_indent_table[fenced_div_level] = nil
12709 return true, tostring(fenced_div_level - 1)
12710 end
12711
12712 return Cg(Cmt(Cb("fenced_div_num_opening_indents")
12713 * Cb("fenced_div_level"), pop_indent_table)
12714 , "fenced_div_level")
12715 end
12716
12717
12718 local non_fenced_div_block
12719 = parsers.check_minimal_indent * V("Block")
12720 - parsers.check_minimal_indent_and_trail * fenced_div_end
12721
12722 local non_fenced_div_paragraph
12723 = parsers.check_minimal_indent * V("Paragraph")
12724 - parsers.check_minimal_indent_and_trail * fenced_div_end
12725
12726 local blank = parsers.minimally_indented_blank
12727
12728 local block_separated = parsers.block_sep_group(blank)

```



```

12729 * non_fenced_div_block
12730
12731 local loop_body_pair
12732 = parsers.create_loop_body_pair(block_separated,
12733 non_fenced_div_paragraph,
12734 parsers.block_sep_group(blank),
12735 parsers.par_sep_group(blank))
12736
12737 local content_loop = (non_fenced_div_block
12738 * loop_body_pair.block^0
12739 + non_fenced_div_paragraph
12740 * block_separated
12741 * loop_body_pair.block^0
12742 + non_fenced_div_paragraph
12743 * loop_body_pair.par^0)
12744 * blank^0
12745
12746 local FencedDiv = fenced_div_begin
12747 / writer.div_begin
12748 * increment_div_level()
12749 * parsers.skipblanklines
12750 * Ct(content_loop)
12751 * parsers.minimally_indented_blank^0
12752 * parsers.check_minimal_indent_and_trail
12753 * fenced_div_end
12754 * decrement_div_level()
12755 * (Cc("") / writer.div_end)
12756
12757 self.insert_pattern("Block after Verbatim",
12758 FencedDiv, "FencedDiv")
12759
12760 self.add_special_character(":")
12761

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div and the indentation matches the opening div fence.

```

12762 local function is_inside_div()
12763 local check_div_level =
12764 function(s, i, fenced_div_level) -- luacheck: ignore s i
12765 fenced_div_level = tonumber(fenced_div_level)
12766 return fenced_div_level > 0
12767 end
12768
12769 return Cmt(Cb("fenced_div_level"), check_div_level)
12770 end
12771

```

```

12772 local function check_indent()
12773 local compare_indent =
12774 function(s, i, indent_table, -- luacheck: ignore s i
12775 fenced_div_num_opening_indents, fenced_div_level)
12776 fenced_div_level = tonumber(fenced_div_level)
12777 local num_current_indents
12778 = (indent_table.current_line_indents ~= nil and
12779 #indent_table.current_line_indents) or 0
12780 local num_opening_indents
12781 = fenced_div_num_opening_indents[fenced_div_level]
12782 return num_current_indents == num_opening_indents
12783 end
12784
12785 return Cmt(Cb("indent_info")
12786 * Cb("fenced_div_num_opening_indents")
12787 * Cb("fenced_div_level"), compare_indent)
12788 end
12789
12790 local fencestart = is_inside_div()
12791 * fenced_div_end
12792 * check_indent()
12793
12794 if not blank_before_div_fence then
12795 self.update_rule("EndlineExceptions", function(previous_pattern)
12796 if previous_pattern == nil then
12797 previous_pattern = parsers.EndlineExceptions
12798 end
12799 return previous_pattern + fencestart
12800 end)
12801 end
12802 end
12803 }
12804 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

12805 M.extensions.header_attributes = function()
12806 return {
12807 name = "built-in header_attributes syntax extension",
12808 extend_writer = function()
12809 end, extend_reader = function(self)
12810 local parsers = self.parsers
12811 local writer = self.writer
12812
12813 local function strip_atx_end(s)

```

```

12814 return s:gsub("%s+##%s*$","")
12815 end
12816
12817 local AtxHeading = Cg(parsers.heading_start, "level")
12818 * parsers.optionalspace
12819 * (C(((parsers.linechar
12820 - (parsers.attributes
12821 * parsers.optionalspace
12822 * parsers.newline))
12823 * (parsers.linechar
12824 - parsers.lbrace)^0)^1)
12825 / strip_atx_end
12826 / parsers.parse_heading_text)
12827 * Cg(Ct(parsers.newline
12828 + (parsers.attributes
12829 * parsers.optionalspace
12830 * parsers.newline)), "attributes")
12831 * Cb("level")
12832 * Cb("attributes")
12833 / writer.heading
12834
12835 local function strip_trailing_spaces(s)
12836 return s:gsub("%s*$","")
12837 end
12838
12839 local heading_line = (parsers.linechar
12840 - (parsers.attributes
12841 * parsers.optionalspace
12842 * parsers.newline))^1
12843 - parsers.thematic_break_lines
12844
12845 local heading_text
12846 = heading_line
12847 * ((V("Endline") / "\n")
12848 * (heading_line - parsers.heading_level))^0
12849 * parsers.newline^-1
12850
12851 local SettextHeading
12852 = parsers.freeze_trail * parsers.check_trail_no_rem
12853 * #(heading_text
12854 * (parsers.attributes
12855 * parsers.optionalspace
12856 * parsers.newline)^-1
12857 * parsers.check_minimal_indent
12858 * parsers.check_trail
12859 * parsers.heading_level)
12860 * Cs(heading_text) / strip_trailing_spaces

```

```

12861 / parsers.parse_heading_text
12862 * Cg(Ct((parsers.attributes
12863 * parsers.optionalspace
12864 * parsers.newline)~1), "attributes")
12865 * parsers.check_minimal_indent_and_trail * parsers.heading_level
12866 * Cb("attributes")
12867 * parsers.newline
12868 * parsers.unfreeze_trail
12869 / writer.heading
12870
12871 local Heading = AtxHeading + SetextHeading
12872 self.update_rule("Heading", Heading)
12873 end
12874 }
12875 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

12876 M.extensions.inline_code_attributes = function()
12877 return {
12878 name = "built-in inline_code_attributes syntax extension",
12879 extend_writer = function()
12880 end, extend_reader = function(self)
12881 local writer = self.writer
12882
12883 local CodeWithAttributes = parsers.inticks
12884 * Ct(parsers.attributes)
12885 / writer.code
12886
12887 self.insert_pattern("Inline before Code",
12888 CodeWithAttributes,
12889 "CodeWithAttributes")
12890 end
12891 }
12892 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

12893 M.extensions.line_blocks = function()
12894 return {
12895 name = "built-in line_blocks syntax extension",
12896 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.

```

12897 function self.lineblock(lines)
12898 if not self.is_writing then return "" end
12899 local buffer = {}
12900 for i = 1, #lines - 1 do
12901 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
12902 end
12903 buffer[#buffer + 1] = lines[#lines]
12904
12905 return {"\\markdownRendererLineBlockBegin\\n"
12906 ,buffer,
12907 "\\n\\markdownRendererLineBlockEnd "}
12908 end
12909 end, extend_reader = function(self)
12910 local parsers = self.parsers
12911 local writer = self.writer
12912
12913 local LineBlock
12914 = Ct((Cs(((parsers.pipe * parsers.space) / " "
12915 * ((parsers.space)/entities.char_entity("nbsp"))^0
12916 * parsers.linechar^0 * (parsers.newline/"")
12917 * (-parsers.pipe
12918 * (parsers.space^1/" ")
12919 * parsers.linechar^1
12920 * (parsers.newline/"")
12921)^0
12922 * (parsers.blankline/"")^0)
12923 / self.parser_functions.parse_inlines)^1)
12924 / writer.lineblock
12925
12926 self.insert_pattern("Block after Blockquote",
12927 LineBlock, "LineBlock")
12928 end
12929 }
12930 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

12931 M.extensions.mark = function()
12932 return {
12933 name = "built-in mark syntax extension",
12934 extend_writer = function(self)

```

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

```

12935 function self.mark(s)
12936 if self.flatten_inlines then return s end
12937 return {"\\markdownRendererMark{" , s, "}"}
12938 end
12939 end, extend_reader = function(self)
12940 local parsers = self.parsers
12941 local writer = self.writer
12942
12943 local doubleequals = P("==")
12944
12945 local Mark
12946 = parsers.between(V("Inline"), doubleequals, doubleequals)
12947 / function (inlines) return writer.mark(inlines) end
12948
12949 self.add_special_character("=")
12950 self.insert_pattern("Inline before LinkAndEmph",
12951 Mark, "Mark")
12952 end
12953 }
12954 end

```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

12955 M.extensions.link_attributes = function()
12956 return {
12957 name = "built-in link_attributes syntax extension",
12958 extend_writer = function()
12959 end, extend_reader = function(self)
12960 local parsers = self.parsers
12961 local options = self.options
12962

```

The following patterns define link reference definitions with attributes.

```

12963 local define_reference_parser
12964 = (parsers.check_trail / "")
12965 * parsers.link_label
12966 * parsers.colon
12967 * parsers.spnlc * parsers.url
12968 * (parsers.spnlc_sep * parsers.title
12969 * (parsers.spnlc * Ct(parsers.attributes))
12970 * parsers.only_blank
12971 + parsers.spnlc_sep * parsers.title * parsers.only_blank
12972 + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
12973 * parsers.only_blank
12974 + Cc("") * parsers.only_blank)
12975

```

```

12976 local ReferenceWithAttributes = define_reference_parser
12977 / self.register_link
12978
12979 self.update_rule("Reference", ReferenceWithAttributes)
12980

```

The following patterns define direct and indirect links with attributes.

```

12981
12982 local LinkWithAttributesAndEmph
12983 = Ct(parsers.link_and_emph_table * Cg(Cc(true),
12984 "match_link_attributes"))
12985 / self.defer_link_and_emphasis_processing
12986
12987 self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
12988

```

The following patterns define autolinks with attributes.

```

12989 local AutoLinkUrlWithAttributes
12990 = parsers.auto_link_url
12991 * Ct(parsers.attributes)
12992 / self.auto_link_url
12993
12994 self.insert_pattern("Inline before AutoLinkUrl",
12995 AutoLinkUrlWithAttributes,
12996 "AutoLinkUrlWithAttributes")
12997
12998 local AutoLinkEmailWithAttributes
12999 = parsers.auto_link_email
13000 * Ct(parsers.attributes)
13001 / self.auto_link_email
13002
13003 self.insert_pattern("Inline before AutoLinkEmail",
13004 AutoLinkEmailWithAttributes,
13005 "AutoLinkEmailWithAttributes")
13006
13007 if options.relativeReferences then
13008
13009 local AutoLinkRelativeReferenceWithAttributes
13010 = parsers.auto_link_relative_reference
13011 * Ct(parsers.attributes)
13012 / self.auto_link_url
13013
13014 self.insert_pattern(
13015 "Inline before AutoLinkRelativeReference",
13016 AutoLinkRelativeReferenceWithAttributes,
13017 "AutoLinkRelativeReferenceWithAttributes")
13018
13019 end

```

```

13020
13021 end
13022 }
13023 end

```

### 3.1.7.13 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```

13024 M.extensions.notes = function(notes, inline_notes)
13025 assert(notes or inline_notes)
13026 return {
13027 name = "built-in notes syntax extension",
13028 extend_writer = function(self)

```

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

```

13029 function self.note(s)
13030 if self.flatten_inlines then return "" end
13031 return {"\\markdownRendererNote{",s,""}
13032 end
13033 end, extend_reader = function(self)
13034 local parsers = self.parsers
13035 local writer = self.writer
13036
13037 local rawnotes = parsers.rawnotes
13038
13039 if inline_notes then
13040 local InlineNote
13041 = parsers.circumflex
13042 * (parsers.link_label
13043 / self.parser_functions.parse_inlines_no_inline_note)
13044 / writer.note
13045
13046 self.insert_pattern("Inline after LinkAndEmph",
13047 InlineNote, "InlineNote")
13048 end
13049 if notes then
13050 local function strip_first_char(s)
13051 return s:sub(2)
13052 end
13053
13054 local RawNoteRef
13055 = #(parsers.lbracket * parsers.circumflex)
13056 * parsers.link_label / strip_first_char

```



```

13057
13058 -- like indirect_link
13059 local function lookup_note(ref)
13060 return writer.defer_call(function()
13061 local found = rawnotes[self.normalize_tag(ref)]
13062 if found then
13063 return writer.note(
13064 self.parser_functions.parse_blocks_nested(found))
13065 else
13066 local text = string.format(
13067 'Undefined note reference "%s"', ref)
13068 local more = string.format(
13069 "Look for the text `[~%s]`.", ref)
13070 return {writer.warning(text, more), "[",
13071 self.parser_functions.parse_inlines("^" .. ref), ""]}
13072 end
13073 end)
13074 end
13075
13076 local function register_note(ref,rawnote)
13077 local normalized_tag = self.normalize_tag(ref)
13078 if rawnotes[normalized_tag] == nil then
13079 rawnotes[normalized_tag] = rawnote
13080 return ""
13081 else
13082 local text
13083 = string.format('Multiply defined note reference "%s"',
13084 ref)
13085 local more
13086 = string.format("Look for the text `[~%s]: ...`.", ref)
13087 return writer.warning(text, more)
13088 end
13089 end
13090
13091 local NoteRef = RawNoteRef / lookup_note
13092
13093 local optionally_indented_line
13094 = parsers.check_optional_indent_and_any_trail * parsers.line
13095
13096 local blank
13097 = parsers.check_optional_blank_indent_and_any_trail
13098 * parsers.optionalspace * parsers.newline
13099
13100 local chunk
13101 = Cs(parsers.line
13102 * (optionally_indented_line - blank)^0)
13103

```

```

13104 local indented_blocks = function(bl)
13105 return Cs(bl
13106 * (blank^1 * (parsers.check_optional_indent / "")
13107 * parsers.check_code_trail
13108 * -parsers.blankline * bl)^0)
13109 end
13110
13111 local NoteBlock
13112 = parsers.check_trail_no_rem
13113 * RawNoteRef * parsers.colon
13114 * parsers.spnlc * indented_blocks(chunk)
13115 / register_note
13116
13117 self.update_rule("Reference", function(previous_pattern)
13118 if previous_pattern == nil then
13119 previous_pattern = parsers.Reference
13120 end
13121 return NoteBlock + previous_pattern
13122 end)
13123
13124 self.insert_pattern("Inline before LinkAndEmph",
13125 NoteRef, "NoteRef")
13126 end
13127
13128 self.add_special_character("^")
13129 end
13130 }
13131 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```

13132 M.extensions.pipe_tables = function(table_captions, table_attributes)
13133
13134 local function make_pipe_table_rectangular(rows)
13135 local num_columns = #rows[2]
13136 local rectangular_rows = {}
13137 for i = 1, #rows do
13138 local row = rows[i]
13139 local rectangular_row = {}
13140 for j = 1, num_columns do
13141 rectangular_row[j] = row[j] or ""

```

```

13142 end
13143 table.insert(rectangular_rows, rectangular_row)
13144 end
13145 return rectangular_rows
13146 end
13147
13148 local function pipe_table_row(allow_empty_first_column
13149 , nonempty_column
13150 , column_separator
13151 , column)
13152 local row_beginning
13153 if allow_empty_first_column then
13154 row_beginning = -- empty first column
13155 #(parsers.spacechar^4
13156 * column_separator)
13157 * parsers.optionalspace
13158 * column
13159 * parsers.optionalspace
13160 -- non-empty first column
13161 + parsers.nonindentSPACE
13162 * nonempty_column^-1
13163 * parsers.optionalspace
13164 else
13165 row_beginning = parsers.nonindentSPACE
13166 * nonempty_column^-1
13167 * parsers.optionalspace
13168 end
13169
13170 return Ct(row_beginning
13171 * (-- single column with no leading pipes
13172 #(column_separator
13173 * parsers.optionalspace
13174 * parsers.newline)
13175 * column_separator
13176 * parsers.optionalspace
13177 -- single column with leading pipes or
13178 -- more than a single column
13179 + (column_separator
13180 * parsers.optionalspace
13181 * column
13182 * parsers.optionalspace)^1
13183 * (column_separator
13184 * parsers.optionalspace)^-1))
13185 end
13186
13187 return {
13188 name = "built-in pipe_tables syntax extension",

```

```
13189 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.

```
13190 function self.table(rows, caption, attributes)
13191 if not self.is_writing then return "" end
13192 local buffer = {}
13193 if attributes ~= nil then
13194 table.insert(buffer,
13195 "\\markdownRendererTableAttributeContextBegin\\n")
13196 table.insert(buffer, self.attributes(attributes))
13197 end
13198 table.insert(buffer,
13199 {"\\markdownRendererTable{",
13200 caption or "", "}{" , #rows - 1, "}{" ,
13201 #rows[1], "}"})
13202 local temp = rows[2] -- put alignments on the first row
13203 rows[2] = rows[1]
13204 rows[1] = temp
13205 for i, row in ipairs(rows) do
13206 table.insert(buffer, "{")
13207 for _, column in ipairs(row) do
13208 if i > 1 then -- do not use braces for alignments
13209 table.insert(buffer, "{")
13210 end
13211 table.insert(buffer, column)
13212 if i > 1 then
13213 table.insert(buffer, "}")
13214 end
13215 end
13216 table.insert(buffer, "}")
13217 end
13218 if attributes ~= nil then
13219 table.insert(buffer,
13220 "\\markdownRendererTableAttributeContextEnd{")
13221 end
13222 return buffer
13223 end
13224 end, extend_reader = function(self)
13225 local parsers = self.parsers
13226 local writer = self.writer
13227
13228 local table_hline_separator = parsers.pipe + parsers.plus
13229
13230 local table_hline_column = (parsers.dash
13231 - #(parsers.dash
13232 * (parsers.spacechar
```

```

13233 + table_hline_separator
13234 + parsers.newline)))^1
13235 * (parsers.colon * Cc("r")
13236 + parsers.dash * Cc("d"))
13237 + parsers.colon
13238 * (parsers.dash
13239 - #(parsers.dash
13240 * (parsers.spacechar
13241 + table_hline_separator
13242 + parsers.newline)))^1
13243 * (parsers.colon * Cc("c")
13244 + parsers.dash * Cc("l"))
13245
13246 local table_hline = pipe_table_row(false
13247 , table_hline_column
13248 , table_hline_separator
13249 , table_hline_column)
13250
13251 local table_caption_beginning
13252 = (parsers.check_minimal_blank_indent_and_any_trail_no_rem
13253 * parsers.optionalspace * parsers.newline)^0
13254 * parsers.check_minimal_indent_and_trail
13255 * (P("Table")^-1 * parsers.colon)
13256 * parsers.optionalspace
13257
13258 local function strip_trailing_spaces(s)
13259 return s:gsub("%s*$","")
13260 end
13261
13262 local table_row
13263 = pipe_table_row(true
13264 , (C((parsers.linechar - parsers.pipe)^1)
13265 / strip_trailing_spaces
13266 / self.parser_functions.parse_inlines)
13267 , parsers.pipe
13268 , (C((parsers.linechar - parsers.pipe)^0)
13269 / strip_trailing_spaces
13270 / self.parser_functions.parse_inlines))
13271
13272 local table_caption
13273 if table_captions then
13274 table_caption = #table_caption_beginning
13275 * table_caption_beginning
13276 if table_attributes then
13277 table_caption = table_caption
13278 * (C(((parsers.linechar
13279 - (parsers.attributes

```

```

13280 * parsers.optionalspace
13281 * parsers.newline
13282 * -(parsers.optionalspace
13283 * parsers.linechar)))
13284 + (parsers.newline
13285 * #(parsers.optionalspace
13286 * parsers.linechar)
13287 * C(parsers.optionalspace)
13288 / writer.space))
13289 * (parsers.linechar
13290 - parsers.lbrace)^0)^1)
13291 / self.parser_functions.parse_inlines)
13292 * (parsers.newline
13293 + (Ct(parsers.attributes)
13294 * parsers.optionalspace
13295 * parsers.newline))
13296 else
13297 table_caption = table_caption
13298 * C((parsers.linechar
13299 + (parsers.newline
13300 * #(parsers.optionalspace
13301 * parsers.linechar)
13302 * C(parsers.optionalspace)
13303 / writer.space))^1)
13304 / self.parser_functions.parse_inlines
13305 * parsers.newline
13306 end
13307 else
13308 table_caption = parsers.fail
13309 end
13310
13311 local PipeTable
13312 = Ct(table_row * parsers.newline
13313 * (parsers.check_minimal_indent_and_trail / {})
13314 * table_hline * parsers.newline
13315 * ((parsers.check_minimal_indent / {})
13316 * table_row * parsers.newline)^0)
13317 / make_pipe_table_rectangular
13318 * table_caption^-1
13319 / writer.table
13320
13321 self.insert_pattern("Block after Blockquote",
13322 PipeTable, "PipeTable")
13323 end
13324 }
13325 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
13326 M.extensions.raw_inline = function()
13327 return {
13328 name = "built-in raw_inline syntax extension",
13329 extend_writer = function(self)
13330 local options = self.options
13331
```

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

```
13332 function self.rawInline(s, attr)
13333 if not self.is_writing then return "" end
13334 if self.flatten_inlines then return s end
13335 local name = util.cache_verbatim(options.cacheDir, s)
13336 return {"\\markdownRendererInputRawInline{",
13337 name,"}{", self.string(attr),""}
13338 end
13339 end, extend_reader = function(self)
13340 local writer = self.writer
13341
13342 local RawInline = parsers.inticks
13343 * parsers.raw_attribute
13344 / writer.rawInline
13345
13346 self.insert_pattern("Inline before Code",
13347 RawInline, "RawInline")
13348 end
13349 }
13350 end
```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
13351 M.extensions.strike_through = function()
13352 return {
13353 name = "built-in strike_through syntax extension",
13354 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.

```
13355 function self.strike_through(s)
13356 if self.flatten_inlines then return s end
13357 return {"\\markdownRendererStrikeThrough{",s,""}
13358 end
```

```

13359 end, extend_reader = function(self)
13360 local parsers = self.parsers
13361 local writer = self.writer
13362
13363 local StrikeThrough = (
13364 parsers.between(parsers.Inline, parsers.doubletildes,
13365 parsers.doubletildes)
13366) / writer.strike_through
13367
13368 self.insert_pattern("Inline after LinkAndEmph",
13369 StrikeThrough, "StrikeThrough")
13370
13371 self.add_special_character("~")
13372 end
13373 }
13374 end

```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

13375 M.extensions.subscripts = function()
13376 return {
13377 name = "built-in subscripts syntax extension",
13378 extend_writer = function(self)

```

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

```

13379 function self.subscript(s)
13380 if self.flatten_inlines then return s end
13381 return {"\\markdownRendererSubscript{" , s, "}"}
13382 end
13383 end, extend_reader = function(self)
13384 local parsers = self.parsers
13385 local writer = self.writer
13386
13387 local Subscript = (
13388 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
13389) / writer.subscript
13390
13391 self.insert_pattern("Inline after LinkAndEmph",
13392 Subscript, "Subscript")
13393
13394 self.add_special_character("~")
13395 end
13396 }
13397 end

```



### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```
13398 M.extensions.superscripts = function()
13399 return {
13400 name = "built-in superscripts syntax extension",
13401 extend_writer = function(self)
```

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

```
13402 function self.superscript(s)
13403 if self.flatten_inlines then return s end
13404 return {"\\markdownRendererSuperscript{" , s, "}"}
13405 end
13406 end, extend_reader = function(self)
13407 local parsers = self.parsers
13408 local writer = self.writer
13409
13410 local Superscript = (
13411 parsers.between(parsers.Str, parsers.circumflex,
13412 parsers.circumflex)
13413) / writer.superscript
13414
13415 self.insert_pattern("Inline after LinkAndEmph",
13416 Superscript, "Superscript")
13417
13418 self.add_special_character("^")
13419 end
13420 }
13421 end
```

### 3.1.7.19 TeX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
13422 M.extensions.tex_math = function(tex_math_dollars,
13423 tex_math_single_backslash,
13424 tex_math_double_backslash)
13425 return {
13426 name = "built-in tex_math syntax extension",
13427 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.

```
13428 function self.display_math(s)
13429 if self.flatten_inlines then return s end
13430 return {"\\markdownRendererDisplayMath{" , self.math(s), "}"}
13431 end
13432 end
13433 }
```

13431       end

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

```
13432 function self.inline_math(s)
13433 if self.flatten_inlines then return s end
13434 return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
13435 end
13436 end, extend_reader = function(self)
13437 local parsers = self.parsers
13438 local writer = self.writer
13439
13440 local function between(p, starter, ender)
13441 return (starter * Cs(p * (p - ender)^0) * ender)
13442 end
13443
13444 local function strip_preceding_spaces(str)
13445 return str:gsub("^%s*(.-)$", "%1")
13446 end
13447
13448 local allowed_before_closing
13449 = B(parsers.backslash * parsers.any
13450 + parsers.any * (parsers.any - parsers.backslash))
13451
13452 local allowed_before_closing_no_space
13453 = B(parsers.backslash * parsers.any
13454 + parsers.any * (parsers.nonspacechar - parsers.backslash))
13455
```

The following patterns implement the Pandoc dollar math syntax extension.

```
13456 local dollar_math_content
13457 = (parsers.newline * (parsers.check_optional_indent / "")
13458 + parsers.backslash^-1
13459 * parsers.linechar)
13460 - parsers.blankline^2
13461 - parsers.dollar
13462
13463 local inline_math_opening_dollars = parsers.dollar
13464 * #(parsers.nonspacechar)
13465
13466 local inline_math_closing_dollars
13467 = allowed_before_closing_no_space
13468 * parsers.dollar
13469 * -#(parsers.digit)
13470
13471 local inline_math_dollars = between(Cs(dollar_math_content),
13472 inline_math_opening_dollars,
13473 inline_math_closing_dollars)
```

```

13474
13475 local display_math_opening_dollars = parsers.dollar
13476 * parsers.dollar
13477
13478 local display_math_closing_dollars = parsers.dollar
13479 * parsers.dollar
13480
13481 local display_math_dollars = between(Cs(dollar_math_content),
13482 display_math_opening_dollars,
13483 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

13484 local backslash_math_content
13485 = (parsers.newline * (parsers.check_optional_indent / ""))
13486 + parsers.linechar)
13487 - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

13488 local inline_math_opening_double = parsers.backslash
13489 * parsers.backslash
13490 * parsers.lparent
13491
13492 local inline_math_closing_double = allowed_before_closing
13493 * parsers.spacechar^0
13494 * parsers.backslash
13495 * parsers.backslash
13496 * parsers.rparent
13497
13498 local inline_math_double = between(Cs(backslash_math_content),
13499 inline_math_opening_double,
13500 inline_math_closing_double)
13501 / strip_preceding_whitespaces
13502
13503 local display_math_opening_double = parsers.backslash
13504 * parsers.backslash
13505 * parsers.lbracket
13506
13507 local display_math_closing_double = allowed_before_closing
13508 * parsers.spacechar^0
13509 * parsers.backslash
13510 * parsers.backslash
13511 * parsers.rbracket
13512
13513 local display_math_double = between(Cs(backslash_math_content),
13514 display_math_opening_double,
13515 display_math_closing_double)

```

```

13516 / strip_preceding_whitespaces
The following patterns implement the Pandoc single backslash math syntax extension.
13517 local inline_math_opening_single = parsers.backslash
13518 * parsers.lparent
13519
13520 local inline_math_closing_single = allowed_before_closing
13521 * parsers.spacechar^0
13522 * parsers.backslash
13523 * parsers.rparent
13524
13525 local inline_math_single = between(Cs(backslash_math_content),
13526 inline_math_opening_single,
13527 inline_math_closing_single)
13528 / strip_preceding_whitespaces
13529
13530 local display_math_opening_single = parsers.backslash
13531 * parsers.lbracket
13532
13533 local display_math_closing_single = allowed_before_closing
13534 * parsers.spacechar^0
13535 * parsers.backslash
13536 * parsers.rbracket
13537
13538 local display_math_single = between(Cs(backslash_math_content),
13539 display_math_opening_single,
13540 display_math_closing_single)
13541 / strip_preceding_whitespaces
13542
13543 local display_math = parsers.fail
13544
13545 local inline_math = parsers.fail
13546
13547 if tex_math_dollars then
13548 display_math = display_math + display_math_dollars
13549 inline_math = inline_math + inline_math_dollars
13550 end
13551
13552 if tex_math_double_backslash then
13553 display_math = display_math + display_math_double
13554 inline_math = inline_math + inline_math_double
13555 end
13556
13557 if tex_math_single_backslash then
13558 display_math = display_math + display_math_single
13559 inline_math = inline_math + inline_math_single
13560 end
13561

```

```

13562 local TexMath = display_math / writer.display_math
13563 + inline_math / writer.inline_math
13564
13565 self.insert_pattern("Inline after LinkAndEmph",
13566 TexMath, "TexMath")
13567
13568 if tex_math_dollars then
13569 self.add_special_character("$")
13570 end
13571
13572 if tex_math_single_backslash or tex_math_double_backslash then
13573 self.add_special_character("\\")
13574 self.add_special_character("[")
13575 self.add_special_character("]")
13576 self.add_special_character("(")
13577 self.add_special_character("(")
13578 end
13579 end
13580 }
13581 end

```

### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. When both `expect_jekyll_data` and `ensure_jekyll_data` parameters are `true`, then a a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata.

```

13582 M.extensions.jekyll_data = function(expect_jekyll_data,
13583 ensure_jekyll_data)
13584 return {
13585 name = "built-in jekyll_data syntax extension",
13586 extend_writer = function(self)

```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t` for the typographic output format used by the `\markdownRendererJekyllDataTypographicString` macro.

```

13587 function self.jekyllData(d, t, p)
13588 if not self.is_writing then return "" end
13589
13590 local buf = {}
13591

```

```

13592 local keys = {}
13593 for k, _ in pairs(d) do
13594 table.insert(keys, k)
13595 end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

13596 table.sort(keys, function(first, second)
13597 if type(first) ~= type(second) then
13598 return type(first) < type(second)
13599 else
13600 return first < second
13601 end
13602 end)
13603
13604 if not p then
13605 table.insert(buf, "\\markdownRendererJekyllDataBegin")
13606 end
13607
13608 local is_sequence = false
13609 if #d > 0 and #d == #keys then
13610 for i=1, #d do
13611 if d[i] == nil then
13612 goto not_a_sequence
13613 end
13614 end
13615 is_sequence = true
13616 end
13617 ::not_a_sequence::
13618
13619 if is_sequence then
13620 table.insert(buf,
13621 "\\markdownRendererJekyllDataSequenceBegin{")
13622 table.insert(buf, self.identifier(p or "null"))
13623 table.insert(buf, "{")
13624 table.insert(buf, #keys)
13625 table.insert(buf, "}")
13626 else
13627 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
13628 table.insert(buf, self.identifier(p or "null"))
13629 table.insert(buf, "{")
13630 table.insert(buf, #keys)
13631 table.insert(buf, "}")
13632 end
13633
13634 for _, k in ipairs(keys) do
13635 local v = d[k]

```

```

13636 local typ = type(v)
13637 k = tostring(k or "null")
13638 if typ == "table" and next(v) ~= nil then
13639 table.insert(
13640 buf,
13641 self.jekyllData(v, t, k)
13642)
13643 else
13644 k = self.identifier(k)
13645 v = tostring(v)
13646 if typ == "boolean" then
13647 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
13648 table.insert(buf, k)
13649 table.insert(buf, "{")
13650 table.insert(buf, v)
13651 table.insert(buf, "}")
13652 elseif typ == "number" then
13653 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
13654 table.insert(buf, k)
13655 table.insert(buf, "{")
13656 table.insert(buf, v)
13657 table.insert(buf, "}")
13658 elseif typ == "string" then
13659 table.insert(buf,
13660 "\\markdownRendererJekyllDataProgrammaticString{")
13661 table.insert(buf, k)
13662 table.insert(buf, "{")
13663 table.insert(buf, self.identifier(v))
13664 table.insert(buf, "}")
13665 table.insert(buf,
13666 "\\markdownRendererJekyllDataTypographicString{")
13667 table.insert(buf, k)
13668 table.insert(buf, "{")
13669 table.insert(buf, t(v))
13670 table.insert(buf, "}")
13671 elseif typ == "table" then
13672 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
13673 table.insert(buf, k)
13674 table.insert(buf, "}")
13675 else
13676 local error = self.error(format(
13677 "Unexpected type %s for value of "
13678 .. "YAML key %s.", typ, k))
13679 table.insert(buf, error)
13680 end
13681 end
13682 end

```

```

13683
13684 if is_sequence then
13685 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
13686 else
13687 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
13688 end
13689
13690 if not p then
13691 table.insert(buf, "\\markdownRendererJekyllDataEnd")
13692 end
13693
13694 return buf
13695 end
13696 end, extend_reader = function(self)
13697 local parsers = self.parsers
13698 local writer = self.writer
13699
13700 local JekyllData
13701 = Cmt(C((parsers.line - P("---") - P("..."))^0)
13702 , function(s, i, text) -- luacheck: ignore s i
13703 local data
13704 local ran_ok, _ = pcall(function()
13705 local tinyyaml = require("tinyyaml")
13706 data = tinyyaml.parse(text, {timestamps=false})
13707 end)
13708 if ran_ok and data ~= nil then
13709 return true, writer.jekyllData(data, function(s)
13710 return self.parser_functions.parse_blocks_nested(s)
13711 end, nil)
13712 else
13713 return false
13714 end
13715 end
13716)
13717
13718 local UnexpectedJekyllData
13719 = P("---")
13720 * parsers.blankline / 0
13721 -- if followed by blank, it's thematic break
13722 * #(-parsers.blankline)
13723 * JekyllData
13724 * (P("---") + P("..."))
13725
13726 local ExpectedJekyllData
13727 = (P("---")
13728 * parsers.blankline / 0
13729 -- if followed by blank, it's thematic break

```



```

13730 * #(-parsers.blankline)
13731)^-1
13732 * JekyllData
13733 * (P("----") + P("..."))^-1
13734
13735 if ensure_jekyll_data then
13736 ExpectedJekyllData = ExpectedJekyllData
13737 * parsers.eof
13738 else
13739 ExpectedJekyllData = (ExpectedJekyllData
13740 * (V("Blank")^0 / writer.interblocksep)
13741)^-1
13742 end
13743
13744 self.insert_pattern("Block before Blockquote",
13745 UnexpectedJekyllData, "UnexpectedJekyllData")
13746 if expect_jekyll_data then
13747 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
13748 end
13749 end
13750 }
13751 end

```

### 3.1.8 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its function `new` unless option `eagerCache` or `finalizeCache` has been enabled and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```

13752 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```

13753 options = options or {}
13754 setmetatable(options, { __index = function (_, key)
13755 return defaultOptions[key] end })
13756 if options.experimental then
13757 setmetatable(options, { __index = function (_, key)
13758 return experimentalOptions[key] end })
13759 end

```

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```

13760 local parser_convert = nil
13761 return function(input, include_flat_output)
13762 local function convert(input)

```

```
13763 if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
13764 local parser = require("markdown-parser")
13765 if metadata.version ~= parser.metadata.version then
13766 warn("markdown.lua " .. metadata.version .. " used with " ..
13767 "markdown-parser.lua " .. parser.metadata.version .. ".")
13768 end
13769 parser_convert = parser.new(options)
13770 end
13771 return parser_convert(input)
13772 end
```

If we cache markdown documents, produce the cache file and transform its filename to plain  $\text{T}_{\text{E}}\text{X}$  output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```
13773 local raw_output, flat_output
13774 if options.eagerCache or options.finalizeCache then
13775 local salt = util.salt(options)
13776 local name, result = util.cache(options.cacheDir, input, salt,
13777 convert, ".md.tex")
13778 raw_output = [[\input{]] .. name .. [[}\relax]]
13779 flat_output = function()
13780 if result == nil then
13781 local input_file = assert(io.open(name, "r"),
13782 [[Could not open file "]] .. name .. [[for reading]])
13783 result = assert(input_file:read("*a"))
13784 assert(input_file:close())
13785 end
13786 return result
13787 end
```

Otherwise, return the result of the conversion directly.

```
13788 else
13789 raw_output = convert(input)
13790 flat_output = function()
13791 return raw_output
13792 end
13793 end
```

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

```
13794 if options.finalizeCache then
13795 local file, mode
```

```

13796 if options.frozenCacheCounter > 0 then
13797 mode = "a"
13798 else
13799 mode = "w"
13800 end
13801 file = assert(io.open(options.frozenCacheFileName, mode),
13802 [[Could not open file]] .. options.frozenCacheFileName
13803 .. [[for writing]])
13804 assert(file:write(
13805 [[\expandafter\global\expandafter\def\csname]]
13806 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
13807 .. [[\endcsname{}]] .. raw_output .. [[]]] .. "\n"))
13808 assert(file:close())
13809 end

```

Besides the canonical output of the conversion, which may contain cached files behind `\input`, also return a function that always produces a flat output regardless of caching as the second return value.

```

13810 if include_flat_output then
13811 return raw_output, flat_output
13812 else
13813 return raw_output
13814 end
13815 end
13816 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```

13817 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```

13818 options = options or {}
13819 setmetatable(options, { __index = function (_, key)
13820 return defaultOptions[key] end })
13821 if options.experimental then
13822 setmetatable(options, { __index = function (_, key)
13823 return experimentalOptions[key] end })
13824 end

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

13825 if options.singletonCache and singletonCache.convert then
13826 for k, v in pairs(defaultOptions) do
13827 if type(v) == "table" then
13828 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
13829 if singletonCache.options[k][i] ~= options[k][i] then
13830 goto miss
13831 end

```

```
13832 end
```

The `cacheDir` option is disregarded.

```
13833 elseif k ~= "cacheDir"
13834 and singletonCache.options[k] ~= options[k] then
13835 goto miss
13836 end
13837 end
13838 return singletonCache.convert
13839 end
13840 ::miss::
```

Apply built-in syntax extensions based on `options`.

```
13841 local extensions = {}
13842
13843 if options.bracketedSpans then
13844 local bracketed_spans_extension = M.extensions.bracketed_spans()
13845 table.insert(extensions, bracketed_spans_extension)
13846 end
13847
13848 if options.contentBlocks then
13849 local content_blocks_extension = M.extensions.content_blocks(
13850 options.contentBlocksLanguageMap)
13851 table.insert(extensions, content_blocks_extension)
13852 end
13853
13854 if options.definitionLists then
13855 local definition_lists_extension = M.extensions.definition_lists(
13856 options.tightLists)
13857 table.insert(extensions, definition_lists_extension)
13858 end
13859
13860 if options.fencedCode then
13861 local fenced_code_extension = M.extensions.fenced_code(
13862 options.blankBeforeCodeFence,
13863 options.fencedCodeAttributes,
13864 options.rawAttribute)
13865 table.insert(extensions, fenced_code_extension)
13866 end
13867
13868 if options.fencedDivs then
13869 local fenced_div_extension = M.extensions.fenced_divs(
13870 options.blankBeforeDivFence)
13871 table.insert(extensions, fenced_div_extension)
13872 end
13873
13874 if options.headerAttributes then
13875 local header_attributes_extension = M.extensions.header_attributes()
```

```

13876 table.insert(extensions, header_attributes_extension)
13877 end
13878
13879 if options.inlineCodeAttributes then
13880 local inline_code_attributes_extension =
13881 M.extensions.inline_code_attributes()
13882 table.insert(extensions, inline_code_attributes_extension)
13883 end
13884
13885 if options.jekyllData then
13886 local jekyll_data_extension = M.extensions.jekyll_data(
13887 options.expectJekyllData, options.ensureJekyllData)
13888 table.insert(extensions, jekyll_data_extension)
13889 end
13890
13891 if options.linkAttributes then
13892 local link_attributes_extension =
13893 M.extensions.link_attributes()
13894 table.insert(extensions, link_attributes_extension)
13895 end
13896
13897 if options.lineBlocks then
13898 local line_block_extension = M.extensions.line_blocks()
13899 table.insert(extensions, line_block_extension)
13900 end
13901
13902 if options.mark then
13903 local mark_extension = M.extensions.mark()
13904 table.insert(extensions, mark_extension)
13905 end
13906
13907 if options.pipeTables then
13908 local pipe_tables_extension = M.extensions.pipe_tables(
13909 options.tableCaptions, options.tableAttributes)
13910 table.insert(extensions, pipe_tables_extension)
13911 end
13912
13913 if options.rawAttribute then
13914 local raw_inline_extension = M.extensions.raw_inline()
13915 table.insert(extensions, raw_inline_extension)
13916 end
13917
13918 if options.strikeThrough then
13919 local strike_through_extension = M.extensions.strike_through()
13920 table.insert(extensions, strike_through_extension)
13921 end
13922

```

```

13923 if options.subscripts then
13924 local subscript_extension = M.extensions.subscripts()
13925 table.insert(extensions, subscript_extension)
13926 end
13927
13928 if options.superscripts then
13929 local superscript_extension = M.extensions.superscripts()
13930 table.insert(extensions, superscript_extension)
13931 end
13932
13933 if options.texMathDollars or
13934 options.texMathSingleBackslash or
13935 options.texMathDoubleBackslash then
13936 local tex_math_extension = M.extensions.tex_math(
13937 options.texMathDollars,
13938 options.texMathSingleBackslash,
13939 options.texMathDoubleBackslash)
13940 table.insert(extensions, tex_math_extension)
13941 end
13942
13943 if options.notes or options.inlineNotes then
13944 local notes_extension = M.extensions.notes(
13945 options.notes, options.inlineNotes)
13946 table.insert(extensions, notes_extension)
13947 end
13948
13949 if options.citations then
13950 local citations_extension
13951 = M.extensions.citations(options.citationNbsps)
13952 table.insert(extensions, citations_extension)
13953 end
13954
13955 if options.fancyLists then
13956 local fancy_lists_extension = M.extensions.fancy_lists()
13957 table.insert(extensions, fancy_lists_extension)
13958 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

13959 for _, user_extension_filename in ipairs(options.extensions) do
13960 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

13961 local pathname = assert(util.find_file(filename),
13962 [[Could not locate user-defined syntax extension "]]
13963 .. filename)
13964 local input_file = assert(io.open(pathname, "r"),
13965 [[Could not open user-defined syntax extension "]]
13966 .. pathname .. ["] for reading]])

```

```

13967 local input = assert(input_file:read("*a"))
13968 assert(input_file:close())
13969 local user_extension, err = load([[
13970 local sandbox = {}
13971 setmetatable(sandbox, {__index = _G})
13972 _ENV = sandbox
13973]] .. input)()
13974 assert(user_extension,
13975 [[Failed to compile user-defined syntax extension "]]
13976 .. pathname .. [[:]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

13977 assert(user_extension.api_version ~= nil,
13978 [[User-defined syntax extension "]] .. pathname
13979 .. [[:] does not specify mandatory field "api_version"]])
13980 assert(type(user_extension.api_version) == "number",
13981 [[User-defined syntax extension "]] .. pathname
13982 .. [[:] specifies field "api_version" of type]]
13983 .. type(user_extension.api_version)
13984 .. [[:] but "number" was expected]])
13985 assert(user_extension.api_version > 0
13986 and user_extension.api_version
13987 <= metadata.user_extension_api_version,
13988 [[User-defined syntax extension "]] .. pathname
13989 .. [[:] uses syntax extension API version]]
13990 .. user_extension.api_version .. [[: but markdown.lua]]
13991 .. metadata.version .. [[: uses API version]]
13992 .. metadata.user_extension_api_version
13993 .. [[:], which is incompatible]])
13994
13995 assert(user_extension.grammar_version ~= nil,
13996 [[User-defined syntax extension "]] .. pathname
13997 .. [[:] does not specify mandatory field "grammar_version"]])
13998 assert(type(user_extension.grammar_version) == "number",
13999 [[User-defined syntax extension "]] .. pathname
14000 .. [[:] specifies field "grammar_version" of type]]
14001 .. type(user_extension.grammar_version)
14002 .. [[:] but "number" was expected]])
14003 assert(user_extension.grammar_version == metadata.grammar_version,
14004 [[User-defined syntax extension "]] .. pathname
14005 .. [[:] uses grammar version]]
14006 .. user_extension.grammar_version
14007 .. [[: but markdown.lua]] .. metadata.version
14008 .. [[: uses grammar version]] .. metadata.grammar_version
14009 .. [[:], which is incompatible]])
14010
14011 assert(user_extension.finalize_grammar ~= nil,
14012 [[User-defined syntax extension "]] .. pathname

```

```

14013 .. [" does not specify mandatory "finalize_grammar" field]])
14014 assert(type(user_extension.finalize_grammar) == "function",
14015 [[User-defined syntax extension "]] .. pathname
14016 .. [" specifies field "finalize_grammar" of type "]]
14017 .. type(user_extension.finalize_grammar)
14018 .. [" but "function" was expected]])

```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.7.)

```

14019 local extension = {
14020 name = [[user-defined "]] .. pathname .. [" syntax extension]],
14021 extend_reader = user_extension.finalize_grammar,
14022 extend_writer = function() end,
14023 }
14024 return extension
14025 end)(user_extension_filename)
14026 table.insert(extensions, user_extension)
14027 end

```

Produce a conversion function from markdown to plain TeX.

```

14028 local writer = M.writer.new(options)
14029 local reader = M.reader.new(writer, options)
14030 local convert = reader.finalize_grammar(extensions)

```

Force garbage collection to reclaim memory for temporary objects created in `writer.new`, `reader.new`, and `reader->finalize_grammar`.

```

14031 collectgarbage("collect")

```

Update the singleton cache.

```

14032 if options.singletonCache then
14033 local singletonCacheOptions = {}
14034 for k, v in pairs(options) do
14035 singletonCacheOptions[k] = v
14036 end
14037 setmetatable(singletonCacheOptions,
14038 { __index = function (_, key)
14039 return defaultOptions[key] end })
14040 singletonCache.options = singletonCacheOptions
14041 singletonCache.convert = convert
14042 end

```

Return the conversion function from markdown to plain TeX.

```

14043 return convert
14044 end

14045 return M

```



### 3.1.9 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.7.

```
14046
14047 local input
14048 if input_filename then
14049 local input_file = assert(io.open(input_filename, "r"),
14050 [[Could not open file]] .. input_filename .. [[for reading]])
14051 input = assert(input_file:read("*a"))
14052 assert(input_file:close())
14053 else
14054 input = assert(io.read("*a"))
14055 end
14056
```

First, ensure that the `options.cacheDir` directory exists.

```
14057 local lfs = require("lfs")
14058 if options.cacheDir and not lfs.isdir(options.cacheDir) then
14059 assert(lfs.mkdir(options["cacheDir"]))
14060 end
```

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

```
14061 local kpse
14062 (function()
14063 local should_initialize = package.loaded.kpse == nil
14064 or tex.initialize ~= nil
14065 kpse = require("kpse")
14066 if should_initialize then
14067 kpse.set_program_name("luatex")
14068 end
14069 end)()
14070 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.

```
14071 if metadata.version ~= md.metadata.version then
14072 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
14073 "markdown.lua " .. md.metadata.version .. ".")
14074 end
14075
14076 local convert = md.new(options)
14077 local raw_output, flat_output = convert(input, true)
14078 local output
14079 if flat_output == nil then
14080 if options.eagerCache then
14081 warn("markdown.lua has not produced flat output, so I am using " ..
```

```

14082 "backwards-compatible raw output instead. This may cause " ..
14083 'the conversion result to be hidden behind "\\input".'))
14084 end
14085 output = raw_output
14086 else
14087 output = flat_output()
14088 end
14089
14090 if output_filename then
14091 local output_file = assert(io.open(output_filename, "w"),
14092 [[Could not open file]] .. output_filename .. [[for writing]])
14093 assert(output_file:write(output))
14094 assert(output_file:close())
14095 else
14096 assert(io.write(output))
14097 end

```

Remove the `options.cacheDir` directory if it is empty.

```

14098 if options.cacheDir then
14099 lfs.rmdir(options.cacheDir)
14100 end

```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```

14101 \ExplSyntaxOn
14102 \cs_if_free:NT
14103 \markdownInfo
14104 {
14105 \cs_new:Npn
14106 \markdownInfo #1
14107 {
14108 \msg_info:nne
14109 { markdown }
14110 { generic-message }
14111 { #1 }
14112 }
14113 }
14114 \cs_if_free:NT
14115 \markdownWarning
14116 {

```

```

14117 \cs_new:Npn
14118 \markdownWarning #1
14119 {
14120 \msg_warning:nne
14121 { markdown }
14122 { generic-message }
14123 { #1 }
14124 }
14125 }
14126 \cs_if_free:NT
14127 \markdownError
14128 {
14129 \cs_new:Npn
14130 \markdownError #1 #2
14131 {
14132 \msg_error:nnee
14133 { markdown }
14134 { generic-message-with-help-text }
14135 { #1 }
14136 { #2 }
14137 }
14138 }
14139 \msg_new:nnn
14140 { markdown }
14141 { generic-message }
14142 { #1 }
14143 \msg_new:nnnn
14144 { markdown }
14145 { generic-message-with-help-text }
14146 { #1 }
14147 { #2 }
14148 \cs_generate_variant:Nn
14149 \msg_info:nnn
14150 { nne }
14151 \cs_generate_variant:Nn
14152 \msg_warning:nnn
14153 { nne }
14154 \cs_generate_variant:Nn
14155 \msg_error:nnnn
14156 { nnee }
14157 \ExplSyntaxOff

```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain T<sub>E</sub>X themes provided with the Markdown package.

```

14158 \ExplSyntaxOn
14159 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
14160 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
14161 \cs_new:Nn
14162 \@@_plain_tex_load_theme:nnn
14163 {
14164 \prop_get:NnNTF
14165 \g_@@_plain_tex_loaded_themes_linenos_prop
14166 { #1 }
14167 \l_tmpa_tl
14168 {
14169 \prop_get:NnN
14170 \g_@@_plain_tex_loaded_themes_versions_prop
14171 { #1 }
14172 \l_tmpb_tl
14173 \str_if_eq:nVTF
14174 { #2 }
14175 \l_tmpb_tl
14176 {
14177 \msg_warning:nnnVn
14178 { markdown }
14179 { repeatedly-loaded-plain-tex-theme }
14180 { #1 }
14181 \l_tmpa_tl
14182 { #2 }
14183 }
14184 {
14185 \msg_error:nnnnVV
14186 { markdown }
14187 { different-versions-of-plain-tex-theme }
14188 { #1 }
14189 { #2 }
14190 \l_tmpb_tl
14191 \l_tmpa_tl
14192 }
14193 }
14194 {
14195 \prop_gput:Nnx
14196 \g_@@_plain_tex_loaded_themes_linenos_prop
14197 { #1 }
14198 { \tex_the:D \tex_inputlineno:D } % noqa: W200
14199 \prop_gput:Nnn
14200 \g_@@_plain_tex_loaded_themes_versions_prop
14201 { #1 }
14202 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

14203 \prop_if_in:NnTF
14204 \g_@@_plain_tex_built_in_themes_prop
14205 { #1 }
14206 {
14207 \msg_info:nnnn
14208 { markdown }
14209 { loading-built-in-plain-tex-theme }
14210 { #1 }
14211 { #2 }
14212 \prop_item:Nn
14213 \g_@@_plain_tex_built_in_themes_prop
14214 { #1 }
14215 }
14216 {
14217 \msg_info:nnnn
14218 { markdown }
14219 { loading-plain-tex-theme }
14220 { #1 }
14221 { #2 }
14222 \file_input:n
14223 { markdown theme #3 }
14224 }
14225 }
14226 }
14227 \msg_new:nnn
14228 { markdown }
14229 { loading-plain-tex-theme }
14230 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
14231 \msg_new:nnn
14232 { markdown }
14233 { loading-built-in-plain-tex-theme }
14234 { Loading~version~#2~of~built-in-plain~TeX~Markdown~theme~#1 }
14235 \msg_new:nnn
14236 { markdown }
14237 { repeatedly-loaded-plain-tex-theme }
14238 {
14239 Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
14240 loaded~on~line~#2,~not~loading~it~again
14241 }
14242 \msg_new:nnn
14243 { markdown }
14244 { different-versions-of-plain-tex-theme }
14245 {
14246 Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
14247 but~version~#3~has~already~been~loaded~on~line~#4

```

```

14248 }
14249 \cs_generate_variant:Nn
14250 \prop_gput:Nnn
14251 { Nnx }
14252 \cs_gset_eq:NN
14253 \@@_load_theme:nnn
14254 \@@_plain_tex_load_theme:nnn
14255 \cs_generate_variant:Nn
14256 \@@_load_theme:nnn
14257 { VeV }
14258 \cs_generate_variant:Nn
14259 \msg_error:nnnnnn
14260 { nnnnVV }
14261 \cs_generate_variant:Nn
14262 \msg_warning:nnnnn
14263 { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T<sub>E</sub>X theme from within themes for higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

14264 \cs_new:Npn
14265 \markdownLoadPlainTeXTheme
14266 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

14267 \tl_set:NV
14268 \l_tmpa_tl
14269 \g_@@_current_theme_tl
14270 \tl_reverse:N
14271 \l_tmpa_tl
14272 \tl_set:Ne
14273 \l_tmpb_tl
14274 {
14275 \tl_tail:V
14276 \l_tmpa_tl
14277 }
14278 \tl_reverse:N
14279 \l_tmpb_tl

```

Next, we munge the theme name.

```

14280 \str_set:NV
14281 \l_tmpa_str
14282 \l_tmpb_tl
14283 \str_replace_all:Nnn
14284 \l_tmpa_str
14285 { / }
14286 { _ }

```

Finally, we load the plain T<sub>E</sub>X theme.

```

14287 \@@_plain_tex_load_theme:VeV
14288 \l_tmpb_tl
14289 { \markdownThemeVersion }
14290 \l_tmpa_str
14291 }
14292 \cs_generate_variant:Nn
14293 \tl_set:Nn
14294 { Ne }
14295 \cs_generate_variant:Nn
14296 \@@_plain_tex_load_theme:nnn
14297 { VeV }

```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```

14298 \prop_gput:Nnn
14299 \g_@@_plain_tex_built_in_themes_prop
14300 { witiko / dot }
14301 {
14302 \str_if_eq:enF
14303 { \markdownThemeVersion }
14304 { silent }
14305 {
14306 \markdownWarning
14307 {
14308 The~theme~name~"witiko/dot"~has~been~soft-deprecated.
14309 \iow_newline:
14310 Consider~changing~the~name~to~"witiko/diagrams@v1".
14311 }
14312 }

```

We enable the `fencedCode` Lua option.

```

14313 \markdownSetup { fencedCode }

```

We store the previous definition of the fenced code token renderer prototype:

```

14314 \cs_set_eq:NN
14315 \@@_dot_previous_definition:nnn
14316 \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:

```

14317 \regex_const:Nn
14318 \c_@@_dot_infostring_regex
14319 { ^dot(\s+(.+)?)? }
14320 \seq_new:N
14321 \l_@@_dot_matches_seq

```

```

14322 \markdownSetup {
14323 rendererPrototypes = {
14324 inputFencedCode = {
14325 \regex_extract_once:NnNTF
14326 \c_@@_dot_infostring_regex
14327 { #2 }
14328 \l_@@_dot_matches_seq
14329 {
14330 \@@_if_option:nF
14331 { frozenCache }
14332 {
14333 \sys_shell_now:n
14334 {
14335 if~!~test~-e~#1.pdf.source~
14336 ||~!~diff~#1~#1.pdf.source;
14337 then~
14338 dot~-Tpdf~-o~#1.pdf~#1;
14339 cp~#1~#1.pdf.source;
14340 fi
14341 }
14342 }
14343 }

```

We include the typeset image using the image token renderer:

```

14343 \exp_args:NNne
14344 \exp_last_unbraced:No
14345 \markdownRendererImage
14346 {
14347 { Graphviz~image }
14348 { #1.pdf }
14349 { #1.pdf }
14350 }
14351 {
14352 \seq_item:Nn
14353 \l_@@_dot_matches_seq
14354 { 3 }
14355 }
14356 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

14357 {
14358 \@@_dot_previous_definition:nnn
14359 { #1 }
14360 { #2 }
14361 { #3 }
14362 }
14363 },
14364 },

```



```

14365 }
14366 }

```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

14367 \prop_gput:Nnn
14368 \g_@@_plain_tex_built_in_themes_prop
14369 { witiko / diagrams }
14370 {
14371 \str_case:enF
14372 { \markdownThemeVersion }
14373 {
14374 { latest }
14375 {
14376 \markdownWarning
14377 {
14378 Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
14379 theme~"witiko/diagrams".~This~will~keep~your~documents~
14380 from~suddenly~breaking~when~we~have~released~future~
14381 versions~of~the~theme~with~backwards~incompatible~
14382 syntax~and~behavior.
14383 }
14384 \markdownSetup
14385 {
14386 import = witiko/diagrams/v2,
14387 }
14388 }
14389 { v2 }
14390 {
14391 \markdownSetup
14392 {
14393 import = witiko/diagrams/v2,
14394 }
14395 }
14396 { v1 }
14397 {
14398 \markdownSetup
14399 {
14400 import = witiko/dot@silent,
14401 }
14402 }
14403 }
14404 {
14405 \msg_error:nnen
14406 { markdown }
14407 { unknown-theme-version }
14408 { witiko/diagrams }

```

```

14409 { \markdownThemeVersion }
14410 { v1 }
14411 }
14412 }
14413 \cs_generate_variant:Nn
14414 \msg_error:nnnnn
14415 { nnnen }
14416 \msg_new:nnnn
14417 { markdown }
14418 { unknown-theme-version }
14419 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
14420 { Known~versions~are:~#3 }

```

Next, we implement the theme [witiko/diagrams/v2](#).

```

14421 \prop_gput:Nnn
14422 \g_@@_plain_tex_built_in_themes_prop
14423 { witiko / diagrams / v2 }
14424 {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

14425 \@@_setup:n
14426 {
14427 fencedCode = true,
14428 fencedCodeAttributes = true,
14429 }

```

Store the previous fenced code token renderer prototype.

```

14430 \cs_set_eq:NN
14431 \@@_diagrams_previous_fenced_code:nnn
14432 \markdownRendererInputFencedCodePrototype

```

Store the caption and the desired format of the diagram.

```

14433 \tl_new:N
14434 \l_@@_diagrams_caption_tl
14435 \tl_new:N
14436 \l_@@_diagrams_format_tl
14437 \tl_set:Nn
14438 \l_@@_diagrams_format_tl
14439 { pdf }
14440 \@@_setup:n
14441 {
14442 rendererPrototypes = {

```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```

14443 fencedCodeAttributeContextBegin = {
14444 \group_begin:
14445 \markdownRendererImageAttributeContextBegin
14446 \cs_set_eq:NN
14447 \@@_diagrams_previous_key_value:nn

```

```

14448 \markdownRendererAttributeKeyValuePrototype
14449 \@@_setup:n
14450 {
14451 rendererPrototypes = {
14452 attributeKeyValue = {
14453 \str_case:nnF
14454 { ##1 }
14455 {
14456 { caption }
14457 {
14458 \tl_set:Nn
14459 \l_@@_diagrams_caption_tl
14460 { ##2 }
14461 }
14462 { format }
14463 {
14464 \tl_set:Nn
14465 \l_@@_diagrams_format_tl
14466 { ##2 }
14467 }
14468 }
14469 {
14470 \@@_diagrams_previous_key_value:nn
14471 { ##1 }
14472 { ##2 }
14473 }
14474 },
14475 },
14476 }
14477 },
14478 fencedCodeAttributeContextEnd = {
14479 \markdownRendererImageAttributeContextEnd
14480 \group_end:
14481 },
14482 },
14483 }
14484 \cs_new:Nn
14485 \@@_diagrams_render_diagram:nnnn
14486 {
14487 \@@_if_option:nF
14488 { frozenCache }
14489 {
14490 \sys_shell_now:n
14491 {
14492 if~!~test~-e~#2.source~
14493 ||~!~diff~#1~#2.source;
14494 then~

```

```

14495 (#3);
14496 cp~#1~#2.source;
14497 fi
14498 }
14499 \exp_args:NnV
14500 \exp_last_unbraced:No
14501 \markdownRendererImage
14502 {
14503 { #4 }
14504 { #2 }
14505 { #2 }
14506 }
14507 \l_@@_diagrams_caption_tl
14508 }
14509 }

```

Use the prop `\g_markdown_diagrams_infostrings_prop` to determine how the code with a given infostring should be processed and routed to the token renderer prototype(s) for images.

```

14510 \prop_new:N
14511 \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

14512 \@@_setup:n
14513 {
14514 rendererPrototypes = {
14515 inputFencedCode = {
14516 \prop_get:NnNTF
14517 \g_markdown_diagrams_infostrings_prop
14518 { #2 }
14519 \l_tmpa_tl
14520 {
14521 \cs_set:NV
14522 \@@_diagrams_infostrings_current:n
14523 \l_tmpa_tl
14524 \@@_diagrams_infostrings_current:n
14525 { #1 }
14526 }

```

Otherwise, use the previous fenced code token renderer prototype.

```

14527 {
14528 \@@_diagrams_previous_fenced_code:nnn
14529 { #1 }
14530 { #2 }
14531 { #3 }
14532 }
14533 },
14534 }

```

```

14535 }
14536 \cs_generate_variant:Nn
14537 \cs_set:Nn
14538 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

14539 \cs_set:Nn
14540 \@@_diagrams_infostrings_current:n
14541 {
14542 \tl_set:Nn
14543 \l_tmpb_tl
14544 { dot~T }
14545 \tl_put_right:NV
14546 \l_tmpb_tl
14547 \l_@@_diagrams_format_tl
14548 \tl_put_right:Nn
14549 \l_tmpb_tl
14550 { ~-o~#1. }
14551 \tl_put_right:NV
14552 \l_tmpb_tl
14553 \l_@@_diagrams_format_tl
14554 \tl_put_right:Nn
14555 \l_tmpb_tl
14556 { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

14557 \str_if_eq:VnT
14558 \l_@@_diagrams_format_tl
14559 { svg }
14560 {
14561 \tl_put_right:Nn
14562 \l_tmpb_tl
14563 { ;~inkscape~#1. }
14564 \tl_put_right:NV
14565 \l_tmpb_tl
14566 \l_@@_diagrams_format_tl
14567 \tl_put_right:Nn
14568 \l_tmpb_tl
14569 { ~---export-area-drawing~---export-dpi=300~-o~#1.pdf }
14570 }
14571 \@@_diagrams_render_diagram:nnVn
14572 { #1 }
14573 { #1.pdf }
14574 \l_tmpb_tl
14575 { Graphviz~image }
14576 }
14577 \cs_generate_variant:Nn

```

```

14578 \@@_diagrams_render_diagram:nnnn
14579 { nnVn }
14580 \@@_tl_set_from_cs:NNn
14581 \l_tmpa_tl
14582 \@@_diagrams_infostrings_current:n
14583 { 1 }
14584 \prop_gput:NnV
14585 \g_markdown_diagrams_infostrings_prop
14586 { dot }
14587 \l_tmpa_tl

```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`. The exact command can be configured by redefining or appending to the `\g_@@_diagrams_mmdc_command_tl` token list or by redefining the `\mmdcCommand` macro. Unlike the token list, the macro can be redefined even before loading the Markdown package.

```

14588 \tl_new:N
14589 \g_@@_diagrams_mmdc_command_tl
14590 \tl_gset:Nn
14591 \g_@@_diagrams_mmdc_command_tl
14592 { mmdc }
14593 \cs_if_free:NT
14594 \mmdcCommand
14595 {
14596 \cs_new:Npn
14597 \mmdcCommand
14598 {
14599 \tl_use:N
14600 \g_@@_diagrams_mmdc_command_tl
14601 }
14602 }
14603 \cs_set:Nn
14604 \@@_diagrams_infostrings_current:n
14605 {
14606 \tl_set:Nx
14607 \l_tmpb_tl
14608 { \mmdcCommand }
14609 \tl_put_right:Nn
14610 \l_tmpb_tl
14611 { ---pdfFit~-i~#1~-o~#1.pdf }
14612 \@@_diagrams_render_diagram:nnVn
14613 { #1 }
14614 { #1.pdf }
14615 \l_tmpb_tl
14616 { Mermaid~image }
14617 }
14618 \@@_tl_set_from_cs:NNn

```

```

14619 \l_tmpa_tl
14620 \c_@@_diagrams_infostrings_current:n
14621 { 1 }
14622 \prop_gput:NnV
14623 \g_markdown_diagrams_infostrings_prop
14624 { mermaid }
14625 \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

14626 \regex_const:Nn
14627 \c_@@_diagrams_filename_suffix_regex
14628 { \.[^\.]*$ }
14629 \cs_set:Nn
14630 \c_@@_diagrams_infostrings_current:n
14631 {

```

Use the output format provided by the user.

```

14632 \tl_set:Nn
14633 \l_tmpa_tl
14634 { #1 }
14635 \regex_replace_once:NxN
14636 \c_@@_diagrams_filename_suffix_regex
14637 {
14638 .
14639 \tl_use:N
14640 \l_@@_diagrams_format_tl
14641 }
14642 \l_tmpa_tl
14643 \tl_set:Nn
14644 \l_tmpb_tl
14645 { plantuml~-t }
14646 \tl_put_right:NV
14647 \l_tmpb_tl
14648 \l__markdown_diagrams_format_tl
14649 \tl_put_right:Nn
14650 \l_tmpb_tl
14651 { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

14652 \str_if_eq:VnT
14653 \l_@@_diagrams_format_tl
14654 { svg }
14655 {
14656 \tl_put_right:Nn
14657 \l_tmpb_tl
14658 { ;~inkscape~ }
14659 \tl_put_right:NV

```

```

14660 \l_tmpb_tl
14661 \l_tmpa_tl
14662 \tl_put_right:Nn
14663 \l_tmpb_tl
14664 { ---export-area-drawing---export-dpi=300~-o~ }
14665 \tl_set:Nn
14666 \l_tmpa_tl
14667 { #1 }
14668 \regex_replace_once:NnN
14669 \c_@@_diagrams_filename_suffix_regex
14670 { .pdf }
14671 \l_tmpa_tl
14672 \tl_put_right:NV
14673 \l_tmpb_tl
14674 \l_tmpa_tl
14675 }
14676 \@@_diagrams_render_diagram:nVVn
14677 { #1 }
14678 \l_tmpa_tl
14679 \l_tmpb_tl
14680 { PlantUML~image }
14681 }
14682 \cs_generate_variant:Nn
14683 \@@_diagrams_render_diagram:nnnn
14684 { nVVn }
14685 \cs_generate_variant:Nn
14686 \regex_replace_once:NnN
14687 { NxN }
14688 \@@_tl_set_from_cs:NNn
14689 \l_tmpa_tl
14690 \@@_diagrams_infostrings_current:n
14691 { 1 }
14692 \prop_gput:NnV
14693 \g_markdown_diagrams_infostrings_prop
14694 { plantuml }
14695 \l_tmpa_tl
14696 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

14697 \group_begin:
14698 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](https://github.com/witiko/graphicx/http) theme stores the previous definition of the image token renderer prototype:

```

14699 \prop_gput:Nnn
14700 \g_@@_plain_tex_built_in_themes_prop
14701 { witiko / graphicx / http }

```



```

14702 {
14703 \cs_set_eq:NN
14704 \@@_graphicx_http_previous_definition:nnnn
14705 \markdownRendererImagePrototype

```

We define variables and functions to enumerate the images for caching and to store the pathname of the file containing the pathname of the downloaded image file.

```

14706 \int_new:N
14707 \g_@@_graphicx_http_image_number_int
14708 \int_gset:Nn
14709 \g_@@_graphicx_http_image_number_int
14710 { 0 }
14711 \cs_new:Nn
14712 \@@_graphicx_http_filename:
14713 {
14714 \markdownOptionCacheDir
14715 / witiko_graphicx_http .
14716 \int_use:N
14717 \g_@@_graphicx_http_image_number_int
14718 }

```

We define a function that will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The function produces a shell command that tries to download the online image to the pathname.

```

14719 \cs_new:Nn
14720 \@@_graphicx_http_download:nn
14721 {
14722 wget~-0~#2~#1~
14723 ||~curl~--location~-o~#2~#1~
14724 ||~rm~-f~#2
14725 }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

14726 \str_new:N
14727 \l_@@_graphicx_http_filename_str
14728 \ior_new:N
14729 \g_@@_graphicx_http_filename_ior
14730 \markdownSetup {
14731 rendererPrototypes = {
14732 image = {
14733 \@@_if_option:nF
14734 { frozenCache }
14735 {

```

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:

```

14736 \sys_shell_now:e
14737 {
14738 mkdir~p~" \markdownOptionCacheDir ";
14739 if~printf~'%s'~"#3"~|~grep~q~E~'^https?:';
14740 then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

14741 OUTPUT_PREFIX=" \markdownOptionCacheDir ";
14742 OUTPUT_BODY="$(printf~'%s'~'#3'
14743 |~md5sum~|~cut~-d'~'~-f1)";
14744 OUTPUT_SUFFIX="$(printf~'%s'~'#3'
14745 |~sed~'s/.*[.]//')";
14746 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

14747 if~!~[~e~"$OUTPUT"~];
14748 then~
14749 \@@_graphicx_http_download:nn
14750 { '#3' }
14751 { "$OUTPUT" } ;
14752 printf~'%s'~"$OUTPUT"~
14753 >~" \@@_graphicx_http_filename: ";
14754 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:

```

14755 else~
14756 printf~'%s'~'#3'~
14757 >~" \@@_graphicx_http_filename: ";
14758 fi
14759 }
14760 }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

14761 \ior_open:Ne
14762 \g_@@_graphicx_http_filename_ior
14763 { \@@_graphicx_http_filename: }
14764 \ior_str_get:NN
14765 \g_@@_graphicx_http_filename_ior
14766 \l_@@_graphicx_http_filename_str
14767 \ior_close:N
14768 \g_@@_graphicx_http_filename_ior
14769 \@@_graphicx_http_previous_definition:nnVn
14770 { #1 }
14771 { #2 }
14772 \l_@@_graphicx_http_filename_str

```

```

14773 { #4 }
14774 \int_gincr:N
14775 \g_@@_graphicx_http_image_number_int
14776 }
14777 }
14778 }
14779 \cs_generate_variant:Nn
14780 \ior_open:Nn
14781 { Ne }
14782 \cs_generate_variant:Nn
14783 \@@_graphicx_http_previous_definition:nnnn
14784 { nnVn }
14785 }
14786 \group_end:

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

14787 \prop_gput:Nnn
14788 \g_@@_plain_tex_built_in_themes_prop
14789 { witiko / tilde }
14790 {
14791 \markdownSetup {
14792 rendererPrototypes = {
14793 tilde = {~},
14794 },
14795 }
14796 }

```

The themes `witiko/example/foo` and `witiko/example/bar` are supposed to be used in code examples. They don't do anything.

```

14797 \clist_map_inline:nn
14798 { foo, bar }
14799 {
14800 \prop_gput:Nnn
14801 \g_@@_plain_tex_built_in_themes_prop
14802 { witiko / example / #1 }
14803 {
14804 \markdownWarning
14805 {
14806 The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
14807 examples.~Using~it~in~actual~code~has~no~effect,~except~
14808 this~warning~message,~and~is~usually~a~mistake.
14809 }
14810 }
14811 }
14812 \ExplSyntaxOff

```

The `witiko/markdown/defaults` plain T<sub>E</sub>X theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

14813 \def\markdownRendererInterblockSeparatorPrototype{\par}%
14814 \def\markdownRendererParagraphSeparatorPrototype{%
14815 \markdownRendererInterblockSeparator}%
14816 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
14817 \def\markdownRendererSoftLineBreakPrototype{ }%
14818 \let\markdownRendererEllipsisPrototype\dots
14819 \def\markdownRendererNbspPrototype{~}%
14820 \def\markdownRendererLeftBracePrototype{\char`\{}%
14821 \def\markdownRendererRightBracePrototype{\char`\}%
14822 \def\markdownRendererDollarSignPrototype{\char`$}%
14823 \def\markdownRendererPercentSignPrototype{\char`\}%
14824 \def\markdownRendererAmpersandPrototype{&}%
14825 \def\markdownRendererUnderscorePrototype{\char`_}%
14826 \def\markdownRendererHashPrototype{\char`\#}%
14827 \def\markdownRendererCircumflexPrototype{\char`^}%
14828 \def\markdownRendererBackslashPrototype{\char`\}%
14829 \def\markdownRendererTildePrototype{\char`~}%
14830 \def\markdownRendererPipePrototype{|}%
14831 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
14832 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
14833 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
14834 \markdownInput{#3}}%
14835 \def\markdownRendererContentBlockOnlineImagePrototype{%
14836 \markdownRendererImage}%
14837 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
14838 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
14839 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
14840 \def\markdownRendererUlBeginPrototype{}%
14841 \def\markdownRendererUlBeginTightPrototype{}%
14842 \def\markdownRendererUlItemPrototype{}%
14843 \def\markdownRendererUlItemEndPrototype{}%
14844 \def\markdownRendererUlEndPrototype{}%
14845 \def\markdownRendererUlEndTightPrototype{}%
14846 \def\markdownRendererOlBeginPrototype{}%
14847 \def\markdownRendererOlBeginTightPrototype{}%
14848 \def\markdownRendererFancyOlBeginPrototype#1#2{%
14849 \markdownRendererOlBegin}%
14850 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
14851 \markdownRendererOlBeginTight}%
14852 \def\markdownRendererOlItemPrototype{}%
14853 \def\markdownRendererOlItemWithNumberPrototype#1{}%
```

```

14854 \def\markdownRendererOlItemEndPrototype{}%
14855 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
14856 \def\markdownRendererFancyOlItemWithNumberPrototype{%
14857 \markdownRendererOlItemWithNumber}%
14858 \def\markdownRendererFancyOlItemEndPrototype{}%
14859 \def\markdownRendererOlEndPrototype{}%
14860 \def\markdownRendererOlEndTightPrototype{}%
14861 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
14862 \def\markdownRendererFancyOlEndTightPrototype{%
14863 \markdownRendererOlEndTight}%
14864 \def\markdownRendererDlBeginPrototype{}%
14865 \def\markdownRendererDlBeginTightPrototype{}%
14866 \def\markdownRendererDlItemPrototype#1{#1}%
14867 \def\markdownRendererDlItemEndPrototype{}%
14868 \def\markdownRendererDlDefinitionBeginPrototype{}%
14869 \def\markdownRendererDlDefinitionEndPrototype{\par}%
14870 \def\markdownRendererDlEndPrototype{}%
14871 \def\markdownRendererDlEndTightPrototype{}%
14872 \def\markdownRendererEmphasisPrototype#1{\it#1}%
14873 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
14874 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
14875 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
14876 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
14877 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
14878 \def\markdownRendererInputVerbatimPrototype#1{%
14879 \par{\tt\input#1\relax}}\par}%
14880 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
14881 \markdownRendererInputVerbatim{#1}}%
14882 \def\markdownRendererHeadingOnePrototype#1{#1}%
14883 \def\markdownRendererHeadingTwoPrototype#1{#1}%
14884 \def\markdownRendererHeadingThreePrototype#1{#1}%
14885 \def\markdownRendererHeadingFourPrototype#1{#1}%
14886 \def\markdownRendererHeadingFivePrototype#1{#1}%
14887 \def\markdownRendererHeadingSixPrototype#1{#1}%
14888 \def\markdownRendererThematicBreakPrototype{}%
14889 \def\markdownRendererNotePrototype#1{#1}%
14890 \def\markdownRendererCitePrototype#1{}%
14891 \def\markdownRendererTextCitePrototype#1{}%
14892 \def\markdownRendererTickedBoxPrototype{[X]}%
14893 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
14894 \def\markdownRendererUntickedBoxPrototype{[]}%
14895 \def\markdownRendererStrikeThroughPrototype#1{#1}%
14896 \def\markdownRendererSuperscriptPrototype#1{#1}%
14897 \def\markdownRendererSubscriptPrototype#1{#1}%
14898 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
14899 \def\markdownRendererInlineMathPrototype#1{$#1$}%
14900 \ExplSyntaxOn

```

```

14901 \cs_gset:Npn
14902 \markdownRendererHeaderAttributeContextBeginPrototype
14903 {
14904 \group_begin:
14905 \color_group_begin:
14906 }
14907 \cs_gset:Npn
14908 \markdownRendererHeaderAttributeContextEndPrototype
14909 {
14910 \color_group_end:
14911 \group_end:
14912 }
14913 \cs_gset_eq:NN
14914 \markdownRendererBracketedSpanAttributeContextBeginPrototype
14915 \markdownRendererHeaderAttributeContextBeginPrototype
14916 \cs_gset_eq:NN
14917 \markdownRendererBracketedSpanAttributeContextEndPrototype
14918 \markdownRendererHeaderAttributeContextEndPrototype
14919 \cs_gset_eq:NN
14920 \markdownRendererFencedDivAttributeContextBeginPrototype
14921 \markdownRendererHeaderAttributeContextBeginPrototype
14922 \cs_gset_eq:NN
14923 \markdownRendererFencedDivAttributeContextEndPrototype
14924 \markdownRendererHeaderAttributeContextEndPrototype
14925 \cs_gset_eq:NN
14926 \markdownRendererFencedCodeAttributeContextBeginPrototype
14927 \markdownRendererHeaderAttributeContextBeginPrototype
14928 \cs_gset_eq:NN
14929 \markdownRendererFencedCodeAttributeContextEndPrototype
14930 \markdownRendererHeaderAttributeContextEndPrototype
14931 \cs_gset:Npn
14932 \markdownRendererReplacementCharacterPrototype
14933 { \codepoint_str_generate:n { fffd } }
14934 \ExplSyntaxOff
14935 \def\markdownRendererSectionBeginPrototype{}%
14936 \def\markdownRendererSectionEndPrototype{}%
14937 \ExplSyntaxOn
14938 \cs_gset:Npn
14939 \markdownRendererWarningPrototype
14940 #1#2#3#4
14941 {
14942 \tl_set:Nn
14943 \l_tmpa_tl
14944 { #2 }
14945 \tl_if_empty:nF
14946 { #4 }
14947 {

```

```

14948 \tl_put_right:Nn
14949 \l_tmpa_tl
14950 { \iow_newline: #4 }
14951 }
14952 \exp_args:NV
14953 \markdownWarning
14954 \l_tmpa_tl
14955 }
14956 \ExplSyntaxOff
14957 \def\markdownRendererErrorPrototype#1#2#3#4{%
14958 \markdownError{#2}{#4}}%

```

### 3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

14959 \ExplSyntaxOn
14960 \cs_new:Nn
14961 \@@_plain_tex_default_input_raw_inline:nn
14962 {
14963 \str_case:nn
14964 { #2 }
14965 {
14966 { md } { \markdownInput{#1} }
14967 { tex } { \markdownEscape{#1} \unskip }
14968 }
14969 }
14970 \cs_new:Nn
14971 \@@_plain_tex_default_input_raw_block:nn
14972 {
14973 \str_case:nn
14974 { #2 }
14975 {
14976 { md } { \markdownInput{#1} }
14977 { tex } { \markdownEscape{#1} }
14978 }
14979 }
14980 \cs_gset:Npn
14981 \markdownRendererInputRawInlinePrototype#1#2
14982 {
14983 \@@_plain_tex_default_input_raw_inline:nn
14984 { #1 }
14985 { #2 }
14986 }
14987 \cs_gset:Npn
14988 \markdownRendererInputRawBlockPrototype#1#2

```

```

14989 {
14990 \@@_plain_tex_default_input_raw_block:nn
14991 { #1 }
14992 { #2 }
14993 }
14994 \ExplSyntaxOff

```

### 3.2.3.2 Simple YAML Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key-value [markdown/jekyllData](#). See also Section 2.2.6.1.

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

14995 \ExplSyntaxOn
14996 \seq_new:N \g_@@_jekyll_data_datatypes_seq
14997 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
14998 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
14999 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\@@_jekyll_data_push_address_segment:n` macro.

```

15000 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
15001 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
15002 {
15003 \seq_if_empty:NF
15004 \g_@@_jekyll_data_datatypes_seq
15005 {
15006 \seq_get_right:NN
15007 \g_@@_jekyll_data_datatypes_seq
15008 \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.

```

15009 \str_if_eq:NNTF
15010 \l_tmpa_tl
15011 \c_@@_jekyll_data_sequence_tl
15012 {
15013 \seq_put_right:Nn
15014 \g_@@_jekyll_data_wildcard_absolute_address_seq
15015 { * }
15016 }
15017 {
15018 \seq_put_right:Nn
15019 \g_@@_jekyll_data_wildcard_absolute_address_seq
15020 { #1 }
15021 }
15022 }
15023 }
```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\@@_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\@@_jekyll_data_update_address_tls:` macro.

```

15024 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
15025 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
15026 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
```

```

15027 {
15028 \seq_pop_left:NN #1 \l_tmpa_tl
15029 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
15030 \seq_put_left:NV #1 \l_tmpa_tl
15031 }
15032 \cs_new:Nn \@@_jekyll_data_update_address_tls:
15033 {
15034 \@@_jekyll_data_concatenate_address:NN
15035 \g_@@_jekyll_data_wildcard_absolute_address_seq
15036 \g_@@_jekyll_data_wildcard_absolute_address_tl
15037 \seq_get_right:NN
15038 \g_@@_jekyll_data_wildcard_absolute_address_seq
15039 \g_@@_jekyll_data_wildcard_relative_address_tl
15040 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\@@_jekyll_data_push:nN` and `\@@_jekyll_data_pop:` macros.

```

15041 \cs_new:Nn \@@_jekyll_data_push:nN
15042 {
15043 \@@_jekyll_data_push_address_segment:n
15044 { #1 }
15045 \seq_put_right:NV
15046 \g_@@_jekyll_data_datatypes_seq
15047 #2
15048 \@@_jekyll_data_update_address_tls:
15049 }
15050 \cs_new:Nn \@@_jekyll_data_pop:
15051 {
15052 \seq_pop_right:NN
15053 \g_@@_jekyll_data_wildcard_absolute_address_seq
15054 \l_tmpa_tl
15055 \seq_pop_right:NN
15056 \g_@@_jekyll_data_datatypes_seq
15057 \l_tmpa_tl
15058 \@@_jekyll_data_update_address_tls:
15059 }

```

To set a single key–value, we will use the `\@@_jekyll_data_set_keyval_known:nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\@@_jekyll_data_set_keyvals_known:nn` macro.

```

15060 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
15061 {
15062 \keys_set_known:nn
15063 { markdown/jekyllData }
15064 { { #1 } = { #2 } }
15065 }
15066 \cs_generate_variant:Nn
15067 \@@_jekyll_data_set_keyval_known:nn

```

```

15068 { Vn }
15069 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
15070 {
15071 \@@_jekyll_data_push:nN
15072 { #1 }
15073 \c_@@_jekyll_data_scalar_tl
15074 \@@_jekyll_data_set_keyval_known:Vn
15075 \g_@@_jekyll_data_wildcard_absolute_address_tl
15076 { #2 }
15077 \@@_jekyll_data_set_keyval_known:Vn
15078 \g_@@_jekyll_data_wildcard_relative_address_tl
15079 { #2 }
15080 \@@_jekyll_data_pop:
15081 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

15082 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
15083 \@@_jekyll_data_push:nN
15084 { #1 }
15085 \c_@@_jekyll_data_sequence_tl
15086 }
15087 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
15088 \@@_jekyll_data_push:nN
15089 { #1 }
15090 \c_@@_jekyll_data_mapping_tl
15091 }
15092 \def\markdownRendererJekyllDataSequenceEndPrototype{
15093 \@@_jekyll_data_pop:
15094 }
15095 \def\markdownRendererJekyllDataMappingEndPrototype{
15096 \@@_jekyll_data_pop:
15097 }
15098 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
15099 \@@_jekyll_data_set_keyvals_known:nn
15100 { #1 }
15101 { #2 }
15102 }
15103 \def\markdownRendererJekyllDataEmptyPrototype#1{}
15104 \def\markdownRendererJekyllDataNumberPrototype#1#2{
15105 \@@_jekyll_data_set_keyvals_known:nn
15106 { #1 }
15107 { #2 }
15108 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

15109 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}

```

```

15110 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
15111 \@@_jekyll_data_set_keyvals_known:nn
15112 { #1 }
15113 { #2 }
15114 }
15115 \ExplSyntaxOff

```

### 3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key-values using the option `jekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

15116 \ExplSyntaxOn
15117 \@@_with_various_cases:nn
15118 { jekyllDataKeyValue }
15119 {
15120 \keys_define:nn
15121 { markdown/options }
15122 {
15123 #1 .code:n = {
15124 \@@_route_jekyll_data_to_key_values:n
15125 { ##1 }
15126 },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

15127 #1 .default:n = { },
15128 }
15129 }
15130 \seq_new:N
15131 \l_@@_jekyll_data_current_position_seq
15132 \tl_new:N
15133 \l_@@_jekyll_data_current_position_tl
15134 \cs_new:Nn
15135 \@@_route_jekyll_data_to_key_values:n
15136 {
15137 \markdownSetup
15138 {
15139 renderers = {
15140 jekyllData(Sequence|Mapping)Begin = {
15141 \bool_lazy_and:nnTF
15142 {
15143 \seq_if_empty_p:N
15144 \l_@@_jekyll_data_current_position_seq
15145 }
15146 {
15147 \str_if_eq_p:nn

```

```

15148 { ##1 }
15149 { null }
15150 }
15151 {
15152 \tl_if_empty:nF
15153 { #1 }
15154 {
15155 \seq_put_right:Nn
15156 \l_@@_jekyll_data_current_position_seq
15157 { #1 }
15158 }
15159 }
15160 {
15161 \seq_put_right:Nn
15162 \l_@@_jekyll_data_current_position_seq
15163 { ##1 }
15164 }
15165 },
15166 jekyllData(Sequence|Mapping)End = {
15167 \seq_pop_right:NN
15168 \l_@@_jekyll_data_current_position_seq
15169 \l_tmpa_tl
15170 },

```

For every YAML key `path.to.<key>` with a value of type *<non-string type>*, set the key *<non-string type>* of the key-value *<module>/path/to/<key>* if it is known and the key *<key>* of the key-value *<module>/path/to* otherwise. *<Non-string type>* is one of `boolean`, `number`, and `empty`.

```

15171 jekyllDataBoolean = {
15172 \tl_set:Nx
15173 \l_@@_jekyll_data_current_position_tl
15174 {
15175 \seq_use:Nn
15176 \l_@@_jekyll_data_current_position_seq
15177 { / }
15178 }
15179 \keys_if_exist:VnTF
15180 \l_@@_jekyll_data_current_position_tl
15181 { ##1 / boolean }
15182 {
15183 \@@_keys_set:xn
15184 {
15185 \tl_use:N
15186 \l_@@_jekyll_data_current_position_tl
15187 / ##1 / boolean
15188 }
15189 { ##2 }

```

```

15190 }
15191 {
15192 \@@_keys_set:xn
15193 {
15194 \tl_use:N
15195 \l_@@_jekyll_data_current_position_tl
15196 / ##1
15197 }
15198 { ##2 }
15199 }
15200 },
15201 jekyllDataNumber = {
15202 \tl_set:Nx
15203 \l_@@_jekyll_data_current_position_tl
15204 {
15205 \seq_use:Nn
15206 \l_@@_jekyll_data_current_position_seq
15207 { / }
15208 }
15209 \keys_if_exist:VnTF
15210 \l_@@_jekyll_data_current_position_tl
15211 { ##1 / number }
15212 {
15213 \@@_keys_set:xn
15214 {
15215 \tl_use:N
15216 \l_@@_jekyll_data_current_position_tl
15217 / ##1 / number
15218 }
15219 { ##2 }
15220 }
15221 {
15222 \@@_keys_set:xn
15223 {
15224 \tl_use:N
15225 \l_@@_jekyll_data_current_position_tl
15226 / ##1
15227 }
15228 { ##2 }
15229 }
15230 },

```

For the *<non-string type>* of `empty`, no value is passed to the key–value. Therefore, a default value should always be defined for nullable keys using the key property `.default:n`.

```

15231 jekyllDataEmpty = {
15232 \tl_set:Nx

```

```

15233 \l_@@_jekyll_data_current_position_tl
15234 {
15235 \seq_use:Nn
15236 \l_@@_jekyll_data_current_position_seq
15237 { / }
15238 }
15239 \keys_if_exist:VnTF
15240 \l_@@_jekyll_data_current_position_tl
15241 { ##1 / empty }
15242 {
15243 \keys_set:xn
15244 {
15245 \tl_use:N
15246 \l_@@_jekyll_data_current_position_tl
15247 / ##1
15248 }
15249 { empty }
15250 }
15251 {
15252 \keys_set:Vn
15253 \l_@@_jekyll_data_current_position_tl
15254 { ##1 }
15255 }
15256 },

```

For every YAML key `path.to.<key>` with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key-value `<module>/path/to/<key>` if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key `<key>` of the key-value `<module>/path/to` with the typographic string of the value unless the key `typographicString` is known. If the key `programmaticString` is known, only set the key `<key>` if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set `<key>` normally, i.e. regardless of whether it is known or unknown.

```

15257 jekyllDataTypographicString = {
15258 \tl_set:Nx
15259 \l_@@_jekyll_data_current_position_tl
15260 {
15261 \seq_use:Nn
15262 \l_@@_jekyll_data_current_position_seq
15263 { / }
15264 }
15265 \keys_if_exist:VnTF
15266 \l_@@_jekyll_data_current_position_tl
15267 { ##1 / typographicString }
15268 {

```

```

15269 \@@_keys_set:xn
15270 {
15271 \tl_use:N
15272 \l_@@_jekyll_data_current_position_tl
15273 / ##1 / typographicString
15274 }
15275 { ##2 }
15276 }
15277 {
15278 \keys_if_exist:VnTF
15279 \l_@@_jekyll_data_current_position_tl
15280 { ##1 / programmaticString }
15281 {
15282 \@@_keys_set_known:xn
15283 {
15284 \tl_use:N
15285 \l_@@_jekyll_data_current_position_tl
15286 / ##1
15287 }
15288 { ##2 }
15289 }
15290 {
15291 \@@_keys_set:xn
15292 {
15293 \tl_use:N
15294 \l_@@_jekyll_data_current_position_tl
15295 / ##1
15296 }
15297 { ##2 }
15298 }
15299 }
15300 },
15301 jekyllDataProgrammaticString = {
15302 \tl_set:Nx
15303 \l_@@_jekyll_data_current_position_tl
15304 {
15305 \seq_use:Nn
15306 \l_@@_jekyll_data_current_position_seq
15307 { / }
15308 }
15309 \keys_if_exist:VnT
15310 \l_@@_jekyll_data_current_position_tl
15311 { ##1 / programmaticString }
15312 {
15313 \@@_keys_set:xn
15314 {
15315 \tl_use:N

```



```

15316 \l_@@_jekyll_data_current_position_tl
15317 / ##1 / programmaticString
15318 }
15319 { ##2 }
15320 }
15321 },
15322 },
15323 }
15324 }
15325 \cs_new:Nn
15326 \@@_keys_set:nn
15327 {
15328 \keys_set:nn
15329 { }
15330 { { #1 } = { #2 } }
15331 }
15332 \cs_new:Nn
15333 \@@_keys_set_known:nn
15334 {
15335 \keys_set_known:nn
15336 { }
15337 { { #1 } = { #2 } }
15338 }
15339 \cs_generate_variant:Nn
15340 \@@_keys_set:nn
15341 { xn }
15342 \cs_generate_variant:Nn
15343 \@@_keys_set_known:nn
15344 { xn }
15345 \cs_generate_variant:Nn
15346 \keys_set:nn
15347 { xn, Vn }
15348 \prg_generate_conditional_variant:Nnn
15349 \keys_if_exist:nn
15350 { Vn }
15351 { T, TF }
15352 \ExplSyntaxOff

```

If plain T<sub>E</sub>X is the top layer, we load the [witiko/markdown/defaults](#) plain T<sub>E</sub>X theme with the default definitions for token renderer prototypes unless the option [noDefaults](#) has been enabled (see Section 2.2.2.3).

```

15353 \ExplSyntaxOn
15354 \str_if_eq:VVT
15355 \c_@@_top_layer_tl
15356 \c_@@_option_layer_plain_tex_tl
15357 {
15358 \use:c

```

```

15359 { ExplSyntaxOff }
15360 \@@_if_option:nF
15361 { noDefaults }
15362 {
15363 \@@_if_option:nTF
15364 { experimental }
15365 {
15366 \@@_setup:n
15367 { theme = witiko/markdown/defaults@experimental }
15368 }
15369 {
15370 \@@_setup:n
15371 { theme = witiko/markdown/defaults }
15372 }
15373 }
15374 \use:c
15375 { ExplSyntaxOn }
15376 }
15377 \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain  $\text{\TeX}$  options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

15378 \ExplSyntaxOn
15379 \tl_new:N \g_@@_formatted_lua_options_tl
15380 \cs_new:Nn \@@_format_lua_options:
15381 {
15382 \tl_gclear:N
15383 \g_@@_formatted_lua_options_tl
15384 \seq_map_function:NN
15385 \g_@@_lua_options_seq
15386 \@@_format_lua_option:n
15387 }
15388 \cs_new:Nn \@@_format_lua_option:n
15389 {
15390 \@@_typecheck_option:n
15391 { #1 }
15392 \@@_get_option_type:nN
15393 { #1 }
15394 \l_tmpa_tl
15395 \bool_case_true:nF
15396 {
15397 {
15398 \str_if_eq_p:VV

```

```

15399 \l_tmpa_tl
15400 \c_@@_option_type_boolean_tl ||
15401 \str_if_eq_p:VV
15402 \l_tmpa_tl
15403 \c_@@_option_type_number_tl ||
15404 \str_if_eq_p:VV
15405 \l_tmpa_tl
15406 \c_@@_option_type_counter_tl
15407 }
15408 {
15409 \@@_get_option_value:nN
15410 { #1 }
15411 \l_tmpa_tl
15412 \tl_gput_right:Nx
15413 \g_@@_formatted_lua_options_tl
15414 { #1~== \l_tmpa_tl ,~ }
15415 }
15416 {
15417 \str_if_eq_p:VV
15418 \l_tmpa_tl
15419 \c_@@_option_type_clist_tl
15420 }
15421 {
15422 \@@_get_option_value:nN
15423 { #1 }
15424 \l_tmpa_tl
15425 \tl_gput_right:Nx
15426 \g_@@_formatted_lua_options_tl
15427 { #1~==\c_left_brace_str }
15428 \clist_map_inline:Vn
15429 \l_tmpa_tl
15430 {
15431 \@@_lua_escape:xN
15432 { ##1 }
15433 \l_tmpb_tl
15434 \tl_gput_right:Nn
15435 \g_@@_formatted_lua_options_tl
15436 { " }
15437 \tl_gput_right:NV
15438 \g_@@_formatted_lua_options_tl
15439 \l_tmpb_tl
15440 \tl_gput_right:Nn
15441 \g_@@_formatted_lua_options_tl
15442 { " ,~ }
15443 }
15444 \tl_gput_right:Nx
15445 \g_@@_formatted_lua_options_tl

```

```

15446 { \c_right_brace_str ,~ }
15447 }
15448 }
15449 {
15450 \@@_get_option_value:nN
15451 { #1 }
15452 \l_tmpa_tl
15453 \@@_lua_escape:xN
15454 { \l_tmpa_tl }
15455 \l_tmpb_tl
15456 \tl_gput_right:Nn
15457 \g_@@_formatted_lua_options_tl
15458 { #1~==~ " }
15459 \tl_gput_right:NV
15460 \g_@@_formatted_lua_options_tl
15461 \l_tmpb_tl
15462 \tl_gput_right:Nn
15463 \g_@@_formatted_lua_options_tl
15464 { " ,~ }
15465 }
15466 }
15467 \cs_generate_variant:Nn
15468 \clist_map_inline:nn
15469 { Vn }
15470 \let
15471 \markdownPrepareLuaOptions
15472 \@@_format_lua_options:
15473 \def
15474 \markdownLuaOptions
15475 {
15476 {
15477 \g_@@_formatted_lua_options_tl
15478 }
15479 }
15480 \sys_if_engine luatex:TF
15481 {
15482 \cs_new:Nn
15483 \@@_lua_escape:nN
15484 {
15485 \tl_set:Nx
15486 #2
15487 {
15488 \lua_escape:n
15489 { #1 }
15490 }
15491 }
15492 }

```

```

15493 {
15494 \regex_const:Nn
15495 \c_@@_lua_escape_regex
15496 { [\\'"'] }
15497 \cs_new:Nn
15498 \@@_lua_escape:nN
15499 {
15500 \tl_set:Nn
15501 #2
15502 { #1 }
15503 \regex_replace_all:NnN
15504 \c_@@_lua_escape_regex
15505 { \u { c_backslash_str } \0 }
15506 #2
15507 }
15508 }
15509 \cs_generate_variant:Nn
15510 \@@_lua_escape:nN
15511 { xN }

```

After the `\markdownPrepareInputFilename` macro has been fully expanded, the `\markdownInputFilename` macro will expand to a Lua string that contains the input filename passed as the first argument.

```

15512 \tl_new:N
15513 \markdownInputFilename
15514 \cs_new:Npn
15515 \markdownPrepareInputFilename
15516 #1
15517 {
15518 \@@_lua_escape:xN
15519 { #1 }
15520 \markdownInputFilename
15521 \tl_gset:Nx
15522 \markdownInputFilename
15523 { " \markdownInputFilename " }
15524 }

```

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.

```

15525 \cs_new:Npn
15526 \markdownPrepare
15527 {

```

First, ensure that the `cacheDir` directory exists.

```

15528 local~lfs = require("lfs")
15529 local~options = \markdownLuaOptions
15530 if~not~lfs.isdir(options.cacheDir) then~

```

```

15531 assert(lfs.mkdir(options.cacheDir))
15532 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.

```

15533 local~md = require("markdown")
15534 local~convert = md.new(options)
15535 }

```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain T<sub>E</sub>X. It opens the input file, converts it, and prints the conversion result.

```

15536 \cs_new:Npn
15537 \markdownConvert
15538 {
15539 local~filename = \markdownInputFilename
15540 local~file = assert(io.open(filename, "r"),
15541 [[Could~not~open~file~]] .. filename .. [[~for~reading]])
15542 local~input = assert(file:read("*a"))
15543 assert(file:close())
15544 print(convert(input))
15545 }
15546 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```

15547 \def\markdownCleanup{%

```

Remove the `options.cacheDir` directory if it is empty.

```

15548 if options.cacheDir then
15549 lfs.rmdir(options.cacheDir)
15550 end
15551 }%

```

### 3.2.5 Buffering Block-Level Markdown Input

The macros `\markdownInputFileStream` and `\markdownOutputFileStream` contain the number of the input and output file streams that will be used for the IO operations of the package.

```

15552 \csname newread\endcsname\markdownInputFileStream
15553 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

15554 \begingroup
15555 \catcode\^^I=12%
15556 \gdef\markdownReadAndConvertTab{^^I}%
15557 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> `\filecontents` macro to plain T<sub>E</sub>X.

```
15558 \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.

```
15559 \catcode\^^M=13%
15560 \catcode\^^I=13%
15561 \catcode|=0%
15562 \catcode\=12%
15563 |catcode%=14%
15564 |catcode|=12@
15565 |gdef|markdownReadAndConvert#1#2{@
15566 |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
15567 |markdownIfOption{frozenCache}{@
15568 |immediate|openout|markdownOutputFileStream@
15569 |markdownOptionInputTempFileName|relax@
15570 |markdownInfo{@
15571 |Buffering block-level markdown input into the temporary @
15572 |input file "|markdownOptionInputTempFileName" and scanning @
15573 |for the closing token sequence "#1"}@
15574 }@
```

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.

```
15575 |def|do##1{|catcode`##1=12}|dospecials@
15576 |catcode`=12@
15577 |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.

```
15578 |def|markdownReadAndConvertStripPercentSign##1{@
15579 |markdownIfOption{stripPercentSigns}{@
15580 |if##1%@
15581 |expandafter|expandafter|expandafter@
15582 |markdownReadAndConvertProcessLine@
15583 |else@
15584 |expandafter|expandafter|expandafter@
```

```

15585 |markdownReadAndConvertProcessLine@
15586 |expandafter|expandafter|expandafter##1@
15587 |fi@
15588 }{@
15589 |expandafter@
15590 |markdownReadAndConvertProcessLine@
15591 |expandafter##1@
15592 }@
15593 }@

```

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.

```

15594 |def|markdownReadAndConvertProcessLine##1##2##1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

15595 |ifx|relax##3|relax@
15596 |markdownIfOption{frozenCache}{@}{@
15597 |immediate|write|markdownOutputFileStream{##1}@
15598 }@
15599 |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 TEX, `\input` the result of the conversion, and expand the ending control sequence.

```

15600 |def^^M{@
15601 |markdownInfo{The ending token sequence was found}@
15602 |markdownIfOption{frozenCache}{@}{@
15603 |immediate|closeout|markdownOutputFileStream@
15604 }@
15605 |endgroup@
15606 |markdownInput{@
15607 |markdownOptionOutputDir@
15608 /|markdownOptionInputTempFileName@
15609 }@
15610 #2}@
15611 |fi@

```

Repeat with the next line.

```

15612 ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

15613 |catcode`|^I=13@

```



```
15614 |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.

```
15615 |catcode`\^^M=13@
15616 |def^^M##1^^M{@
15617 |def^^M###1^^M{@
15618 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
15619 ^^M}@
15620 ^^M}@
```

Reset the character categories back to the former state.

```
15621 |endgroup
```

Use the `lt3luabridge` library to define the `\markdownLuaExecute` macro, which takes in a Lua scripts and expands to the standard output produced by its execution.

```
15622 \ExplSyntaxOn
15623 \cs_new:Npn
15624 \markdownLuaExecute
15625 #1
15626 {
15627 \int_compare:nNt
15628 { \g_luabridge_method_int }
15629 =
15630 { \c_luabridge_method_shell_int }
15631 {
15632 \sys_if_shell_unrestricted:F
15633 {
15634 \sys_if_shell:TF
15635 {
15636 \msg_error:nn
15637 { markdown }
15638 { restricted-shell-access }
15639 }
15640 {
15641 \msg_error:nn
15642 { markdown }
15643 { disabled-shell-access }
15644 }
15645 }
15646 }
15647 \str_gset:NV
15648 \g_luabridge_output_dirname_str
15649 \markdownOptionOutputDir
15650 \luabridge_now:e
15651 { #1 }
15652 }
```

```

15653 \cs_generate_variant:Nn
15654 \msg_new:nnnn
15655 { nnnV }
15656 \tl_set:Nn
15657 \l_tmpa_tl
15658 {
15659 You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
15660 --enable-write18~flag,~or~write~shell_escape=t~in~the~
15661 texmf.cnf~file.
15662 }
15663 \msg_new:nnnV
15664 { markdown }
15665 { restricted-shell-access }
15666 { Shell~escape~is~restricted }
15667 \l_tmpa_tl
15668 \msg_new:nnnV
15669 { markdown }
15670 { disabled-shell-access }
15671 { Shell~escape~is~disabled }
15672 \l_tmpa_tl
15673 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

15674 \ExplSyntaxOn
15675 \tl_new:N
15676 \g_@@_after_markinline_tl
15677 \tl_gset:Nn
15678 \g_@@_after_markinline_tl
15679 { \unskip }
15680 \cs_new:Npn
15681 \markinline
15682 {

```

Locally change the category of the special plain  $\TeX$  characters to *other* in order to prevent unwanted interpretation of the input markdown text as  $\TeX$  code.

```

15683 \group_begin:
15684 \cctab_select:N
15685 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

15686 \@@_if_option:nF
15687 { frozenCache }
15688 {
15689 \immediate

```

```

15690 \openout
15691 \markdownOutputFileStream
15692 \markdownOptionInputTempFileName
15693 \relax
15694 \msg_info:nne
15695 { markdown }
15696 { buffering-markinline }
15697 { \markdownOptionInputTempFileName }
15698 }

```

Peek ahead and extract the inline markdown text.

```

15699 \peek_regex_replace_once:nnF
15700 { { (.*) } }
15701 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

15702 \c { @@_if_option:nF }
15703 \cB { frozenCache \cE }
15704 \cB {
15705 \c { immediate }
15706 \c { write }
15707 \c { markdownOutputFileStream }
15708 \cB { \1 \cE }
15709 \c { immediate }
15710 \c { closeout }
15711 \c { markdownOutputFileStream }
15712 \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

15713 \c { group_end: }
15714 \c { group_begin: }
15715 \c { @@_setup:n }
15716 \cB { contentLevel = inline \cE }
15717 \c { markdownInput }
15718 \cB {
15719 \c { markdownOptionOutputDir } /
15720 \c { markdownOptionInputTempFileName }
15721 \cE }
15722 \c { group_end: }
15723 \c { tl_use:N }
15724 \c { g_@@_after_markinline_tl }
15725 }
15726 {
15727 \msg_error:nn
15728 { markdown }
15729 { markinline-peek-failure }
15730 }
15731 \group_end:

```

```

15731 \tl_use:N
15732 \g_@@_after_markinline_tl
15733 }
15734 }
15735 \msg_new:nnn
15736 { markdown }
15737 { buffering-markinline }
15738 { Buffering~inline~markdown~input~into~
15739 the~temporary~input~file~"#1". }
15740 \msg_new:nnnn
15741 { markdown }
15742 { markinline-peek-failure }
15743 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
15744 { The~macro~should~be~followed~by~inline~
15745 markdown~text~in~curly~braces }
15746 \ExplSyntaxOff

```

### 3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T<sub>E</sub>X.

```

15747 \ExplSyntaxOn
15748 \cs_new:Npn
15749 \markdownInput
15750 #1
15751 {
15752 \@@_if_option:nTF
15753 { frozenCache }
15754 {
15755 \markdownInputRaw
15756 { #1 }
15757 }
15758 {

```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On L<sup>A</sup>T<sub>E</sub>X, this also includes the directories specified in `\input@path`.

```

15759 \tl_set:Nx
15760 \l_tmpa_tl
15761 { #1 }
15762 \file_get_full_name:VNTF
15763 \l_tmpa_tl
15764 \l_tmpb_tl
15765 {
15766 \exp_args:NV
15767 \markdownInputRaw

```

```

15768 \l_tmpb_tl
15769 }
15770 {
15771 \msg_error:nnV
15772 { markdown }
15773 { markdown-file-does-not-exist }
15774 \l_tmpa_tl
15775 }
15776 }
15777 }
15778 \msg_new:nnn
15779 { markdown }
15780 { markdown-file-does-not-exist }
15781 {
15782 Markdown~file~#1~does~not~exist
15783 }
15784 \ExplSyntaxOff
15785 \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.

```

15786 \catcode`\=0%
15787 \catcode`\=12%
15788 \catcode`\&=6%
15789 \gdef|markdownInputRaw#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the [hybrid](#) Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

15790 |begingroup
15791 |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.

```

15792 |catcode`\#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache⟨number⟩` macro, and increment `frozenCacheCounter`.

```

15793 |markdownIfOption{frozenCache}{%
15794 |ifnum|markdownOptionFrozenCacheCounter=0|relax
15795 |markdownInfo{Reading frozen cache from
15796 "||markdownOptionFrozenCacheFileName"}%
15797 |input|markdownOptionFrozenCacheFileName|relax
15798 |fi
15799 |markdownInfo{Including markdown document number
15800 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%

```

```

15801 |csname markdownFrozenCache%
15802 |the|markdownOptionFrozenCacheCounter|endcsname
15803 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
15804 }{%
15805 |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as  $\text{\LaTeX}$ Mk to track changes to the markdown document.

```

15806 |openin|markdownInputFileStream{&1}%
15807 |closein|markdownInputFileStream
15808 |markdownPrepareLuaOptions
15809 |markdownPrepareInputFilename{&1}%
15810 |markdownLuaExecute{%
15811 |markdownPrepare
15812 |markdownConvert
15813 |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

15814 |markdownIfOption{finalizeCache}{%
15815 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
15816 }%
15817 |endgroup
15818 }%
15819 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of  $\text{\TeX}$  to execute a  $\text{\TeX}$  document in the middle of a markdown document fragment.

```

15820 \gdef\markdownEscape#1{%
15821 \catcode`\%=14\relax
15822 \catcode`\#=6\relax
15823 \input #1\relax
15824 \catcode`\%=12\relax
15825 \catcode`\#=12\relax
15826 }%

```

### 3.3 $\text{\LaTeX}$ Implementation

The  $\text{\LaTeX}$  implementation makes use of the fact that, apart from some subtle differences,  $\text{\LaTeX}$  implements the majority of the plain  $\text{\TeX}$  format [19, Section 9]. As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```

15827 \def\markdownVersionSpace{ }%
15828 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
15829 \markdownVersion\markdownVersionSpace markdown renderer]%

```

### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.3).

```
15830 \ExplSyntaxOn
15831 \cs_gset_eq:NN
15832 \markinlinePlainTeX
15833 \markinline
15834 \cs_gset:Npn
15835 \markinline
15836 {
15837 \peek_regex_replace_once:nn
15838 { (\[(.*) \]) ? }
15839 {
```

Apply the options locally.

```
15840 \c { group_begin: }
15841 \c { @@_setup:n }
15842 \cB { \2 \cE }
15843 \c { tl_put_right:Nn }
15844 \c { g_@@_after_markinline_tl }
15845 \cB { \c { group_end: } \cE }
15846 \c { markinlinePlainTeX }
15847 }
15848 }
15849 \ExplSyntaxOff
```

The `\markdownInputPlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.3).

```
15850 \let\markdownInputPlainTeX\markdownInput
15851 \renewcommand\markdownInput[2][]{%
15852 \begingroup
15853 \markdownSetup{#1}%
15854 \markdownInputPlainTeX{#2}%
15855 \endgroup}%
15856 \renewcommand\yamlInput[2][]{%
15857 \begingroup
15858 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
15859 \markdownInputPlainTeX{#2}%
15860 \endgroup}%
```

The `markdown`, `markdown*`, and `yaml`  $\LaTeX$  environments are implemented using the `\markdownReadAndConvert` macro.

```

15861 \ExplSyntaxOn
15862 \renewenvironment
15863 { markdown }
15864 {

```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

<pre> \begin{markdown} [smartEllipses] % This is an optional argument ~ % ... \end{markdown} </pre>	<pre> \begin{markdown} [smartEllipses] % ~ This is link \end{markdown} </pre>
-----------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```

15865 \group_begin:
15866 \char_set_catcode_active:n { 13 }

```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 11 (letter).

```

15867 \char_set_catcode_letter:n { 35 }

```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```

15868 \peek_regex_replace_once:nnF
15869 { \ *\[r*([~]*)\[^\r]* }
15870 {

```

After we have matched the optional argument, we switch back the category code of carriage returns and hash signs and we retokenize the content. This will cause single new lines to produce a space token and multiple new lines to produce `\par` tokens. Furthermore, this will cause hash signs followed by a number to be recognized as parameter numbers, which is necessary when we use the optional argument to redefine token renderers and token renderer prototypes.



```

15871 \c { group_end: }
15872 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }

```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```

15873 \c { @@_setup:V } \c { l_tmpa_tl }

```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the  $\text{\LaTeX}$  environment.

We also make provision for using the `\markdown` command as a part of a different  $\text{\LaTeX}$  environment as follows:

```

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

```

```

15874 \c { exp_args:NV }
15875 \c { markdownReadAndConvert@ }
15876 \c { @currenvir }
15877 }
15878 {
15879 \group_end:
15880 \exp_args:NV
15881 \markdownReadAndConvert@
15882 \@currenvir
15883 }
15884 }
15885 { \markdownEnd }
15886 \renewenvironment
15887 { markdown* }
15888 [1]
15889 {
15890 \@@_if_option:nTF
15891 { experimental }
15892 {
15893 \msg_error:nn
15894 { markdown }
15895 { latex-markdown-star-deprecated }
15896 }
15897 {
15898 \msg_warning:nn
15899 { markdown }
15900 { latex-markdown-star-deprecated }
15901 }
15902 \@@_setup:n
15903 { #1 }
15904 \markdownReadAndConvert@
15905 { markdown* }

```

```

15906 }
15907 { \markdownEnd }
15908 \renewenvironment
15909 { yaml }
15910 {
15911 \group_begin:
15912 \yamlSetup
15913 { jekyllData, expectJekyllData, ensureJekyllData }
15914 \markdown
15915 }
15916 { \yamlEnd }
15917 \msg_new:nnn
15918 { markdown }
15919 { latex-markdown-star-deprecated }
15920 {
15921 The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
15922 be~removed~in~the~next~major~version~of~the~Markdown~package.
15923 }
15924 \cs_generate_variant:Nn % noqa: w402
15925 \@@_setup:n
15926 { V }
15927 \ExplSyntaxOff
15928 \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*.

```

15929 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
15930 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
15931 |gdef|markdownReadAndConvert@#1<%
15932 |markdownReadAndConvert<\end{#1}>%
15933 <|end<#1>>>%
15934 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\text{\TeX}$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\text{\LaTeX}$  themes provided with the Markdown package.

```

15935 \ExplSyntaxOn
15936 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
15937 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
15938 \cs_gset:Nn
15939 \@@_load_theme:nnn
15940 {

```

If the Markdown package has not yet been loaded, determine whether either this is a built-in theme according to the prop `\g_@@_latex_built_in_themes_prop` or a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```
15941 \ifmarkdownLaTeXLoaded
15942 \ifx\@onlypreamble\@notprerr
```

If both conditions are true, end with an error, since we cannot load L<sup>A</sup>T<sub>E</sub>X themes after the preamble.

```
15943 \bool_if:nTF
15944 {
15945 \bool_lazy_or_p:nn
15946 {
15947 \prop_if_in_p:Nn
15948 \g_@@_latex_built_in_themes_prop
15949 { #1 }
15950 }
15951 {
15952 \file_if_exist_p:n
15953 { markdown theme #3.sty }
15954 }
15955 }
15956 {
15957 \msg_error:nn
15958 { markdown }
15959 { latex-theme-after-preamble }
15960 }
```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```
15961 {
15962 \@@_plain_tex_load_theme:nnn
15963 { #1 }
15964 { #2 }
15965 { #3 }
15966 }
15967 \else
```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```
15968 \bool_if:nTF
15969 {
15970 \bool_lazy_or_p:nn
15971 {
15972 \prop_if_in_p:Nn
15973 \g_@@_latex_built_in_themes_prop
15974 { #1 }
15975 }
```

```

15976 {
15977 \file_if_exist_p:n
15978 { markdown theme #3.sty }
15979 }
15980 }
15981 {
15982 \prop_get:NnNTF
15983 \g_@@_latex_loaded_themes_linenos_prop
15984 { #1 }
15985 \l_tmpa_tl
15986 {
15987 \prop_get:NnN
15988 \g_@@_latex_loaded_themes_versions_prop
15989 { #1 }
15990 \l_tmpb_tl
15991 \str_if_eq:nVTF
15992 { #2 }
15993 \l_tmpb_tl
15994 {
15995 \msg_warning:nnnVn
15996 { markdown }
15997 { repeatedly-loaded-latex-theme }
15998 { #1 }
15999 \l_tmpa_tl
16000 { #2 }
16001 }
16002 }
16003 \msg_error:nnnnVV
16004 { markdown }
16005 { different-versions-of-latex-theme }
16006 { #1 }
16007 { #2 }
16008 \l_tmpb_tl
16009 \l_tmpa_tl
16010 }
16011 }
16012 {
16013 \prop_gput:Nnx
16014 \g_@@_latex_loaded_themes_linenos_prop
16015 { #1 }
16016 { \tex_the:D \tex_inputlineno:D } % noqa: W200
16017 \prop_gput:Nnn
16018 \g_@@_latex_loaded_themes_versions_prop
16019 { #1 }
16020 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

16021 \prop_if_in:NnTF
16022 \g_@@_latex_built_in_themes_prop
16023 { #1 }
16024 {
16025 \msg_info:nnnn
16026 { markdown }
16027 { loading-built-in-latex-theme }
16028 { #1 }
16029 { #2 }
16030 \prop_item:Nn
16031 \g_@@_latex_built_in_themes_prop
16032 { #1 }
16033 }
16034 {
16035 \msg_info:nnnn
16036 { markdown }
16037 { loading-latex-theme }
16038 { #1 }
16039 { #2 }
16040 \RequirePackage
16041 { markdown theme #3 }
16042 }
16043 }
16044 }
16045 {
16046 \@@_plain_tex_load_theme:nnn
16047 { #1 }
16048 { #2 }
16049 { #3 }
16050 }
16051 \fi
16052 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

16053 \msg_info:nnnn
16054 { markdown }
16055 { theme-loading-postponed }
16056 { #1 }
16057 { #2 }
16058 \AtEndOfPackage
16059 {
16060 \@@_set_theme:n
16061 { #1 @ #2 }
16062 }

```

```

16063 \fi
16064 }
16065 \msg_new:nnn
16066 { markdown }
16067 { theme-loading-postponed }
16068 {
16069 Postponing~loading~version~#2~of~Markdown~theme~#1~until~
16070 Markdown~package~has~finished~loading
16071 }
16072 \msg_new:nnn
16073 { markdown }
16074 { loading-built-in-latex-theme }
16075 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
16076 \msg_new:nnn
16077 { markdown }
16078 { loading-latex-theme }
16079 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
16080 \msg_new:nnn
16081 { markdown }
16082 { repeatedly-loaded-latex-theme }
16083 {
16084 Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
16085 loaded~on~line~#2,~not~loading~it~again
16086 }
16087 \msg_new:nnn
16088 { markdown }
16089 { different-versions-of-latex-theme }
16090 {
16091 Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
16092 but~version~#3~has~already~been~loaded~on~line~#4
16093 }
16094 \cs_generate_variant:Nn
16095 \msg_new:nnnn
16096 { nnVV }
16097 \tl_set:Nn
16098 \l_tmpa_tl
16099 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
16100 \tl_put_right:NV
16101 \l_tmpa_tl
16102 \c_backslash_str
16103 \tl_put_right:Nn
16104 \l_tmpa_tl
16105 { begin { document } } }
16106 \tl_set:Nn
16107 \l_tmpb_tl
16108 { Load~Markdown~theme~#1~before~ }
16109 \tl_put_right:NV

```

```

16110 \l_tmpb_tl
16111 \c_backslash_str
16112 \tl_put_right:Nn
16113 \l_tmpb_tl
16114 { begin { document } }
16115 \msg_new:nnVV
16116 { markdown }
16117 { latex-theme-after-preamble }
16118 \l_tmpa_tl
16119 \l_tmpb_tl

```

The [witiko/dot](#) and [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X themes load the package `graphicx`, see also Section 1.1.3. Then, they load the corresponding plain T<sub>E</sub>X themes.

```

16120 \tl_set:Nn
16121 \l_tmpa_tl
16122 {
16123 \RequirePackage
16124 { graphicx }
16125 \markdownLoadPlainTeXTheme
16126 }
16127 \prop_gput:NnV
16128 \g_@@_latex_built_in_themes_prop
16129 { witiko / dot }
16130 \l_tmpa_tl
16131 \prop_gput:NnV
16132 \g_@@_latex_built_in_themes_prop
16133 { witiko / graphicx / http }
16134 \l_tmpa_tl
16135 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) L<sup>A</sup>T<sub>E</sub>X theme also loads the corresponding plain T<sub>E</sub>X theme.

```

16136 \markdownLoadPlainTeXTheme

```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.3 Options

The supplied package options are processed using the [\markdownSetup](#) macro.

```

16137 \DeclareOption*{%
16138 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
16139 \ProcessOptions\relax

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option [plain](#) has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
16140 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

### 3.3.4.1 Lists

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package `paralist` or the package `enumitem` have already been loaded, use them. Otherwise, if the option `experimental` or the command `\DocumentMetadata` have been used, use the package `enumitem`. Otherwise, use the package `paralist`.

```
16141 \ExplSyntaxOn
16142 \bool_new:N
16143 \g_@@_tight_or_fancy_lists_bool
16144 \bool_gset_false:N
16145 \g_@@_tight_or_fancy_lists_bool
16146 \@@_if_option:nTF
16147 { tightLists }
16148 {
16149 \bool_gset_true:N
16150 \g_@@_tight_or_fancy_lists_bool
16151 }
16152 {
16153 \@@_if_option:nT
16154 { fancyLists }
16155 {
16156 \bool_gset_true:N
16157 \g_@@_tight_or_fancy_lists_bool
16158 }
16159 }
16160 \bool_new:N
16161 \g_@@_beamer_paralist_or_enumitem_bool
16162 \bool_gset_true:N
16163 \g_@@_beamer_paralist_or_enumitem_bool
16164 \@ifclassloaded
16165 { beamer }
16166 { }
16167 {
16168 \@ifpackageloaded
16169 { paralist }
16170 { }
16171 {
16172 \@ifpackageloaded
16173 { enumitem }
16174 { }
16175 {
16176 \bool_gset_false:N
16177 \g_@@_beamer_paralist_or_enumitem_bool
```



```

16178 }
16179 }
16180 }
16181 \prg_generate_conditional_variant:Nnn
16182 \str_if_eq:nn
16183 { en }
16184 { TF }
16185 \bool_if:nT
16186 {
16187 \g_@@_tight_or_fancy_lists_bool &&
16188 ! \g_@@_beamer_paralist_or_enumitem_bool
16189 }
16190 {
16191 \str_if_eq:enTF
16192 { \markdownThemeVersion }
16193 { experimental }
16194 {
16195 \RequirePackage
16196 { enumitem }
16197 }
16198 {
16199 \IfDocumentMetadataTF
16200 {
16201 \RequirePackage
16202 { enumitem }
16203 }
16204 {
16205 \RequirePackage
16206 { paralist }
16207 }
16208 }
16209 }
16210 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16211 \ExplSyntaxOn
16212 \cs_new:Nn
16213 \@@_latex_fancy_list_item_label_number:nn
16214 {
16215 \str_case:nn
16216 { #1 }
16217 {
16218 { Decimal } { #2 }
16219 { LowerRoman } { \int_to_roman:n { #2 } }
16220 { UpperRoman } { \int_to_Roman:n { #2 } }
16221 { LowerAlpha } { \int_to_alph:n { #2 } }

```

```

16222 { UpperAlpha } { \int_to_Alph:n { #2 } }
16223 }
16224 }
16225 \cs_new:Nn
16226 \@@_latex_fancy_list_item_label_delimiter:n
16227 {
16228 \str_case:nn
16229 { #1 }
16230 {
16231 { Default } { . }
16232 { OneParen } {) }
16233 { Period } { . }
16234 }
16235 }
16236 \cs_new:Nn
16237 \@@_latex_fancy_list_item_label:nnn
16238 {
16239 \@@_latex_fancy_list_item_label_number:nn
16240 { #1 }
16241 { #3 }
16242 \@@_latex_fancy_list_item_label_delimiter:n
16243 { #2 }
16244 }
16245 \cs_generate_variant:Nn
16246 \@@_latex_fancy_list_item_label:nnn
16247 { VVn }
16248 \tl_new:N
16249 \l_@@_latex_fancy_list_item_label_number_style_tl
16250 \tl_new:N
16251 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16252 \ifpackageloaded { enumitem } {
16253 \markdownSetup { rendererPrototypes = {
First, let's define the tight list item renderer prototypes.
16254 ulBeginTight = {
16255 \begin
16256 { itemize }
16257 [noitemsep]
16258 },
16259 ulEndTight = {
16260 \end
16261 { itemize }
16262 },
16263 olBeginTight = {
16264 \begin
16265 { enumerate }
16266 [noitemsep]
16267 },

```

```

16268 olEndTight = {
16269 \end
16270 { enumerate }
16271 },
16272 dlBeginTight = {
16273 \begin
16274 { description }
16275 [noitemsep]
16276 },
16277 dlEndTight = {
16278 \end
16279 { description }
16280 },

```

Second, let's define the fancy list item renderer prototypes.

```

16281 fancyOlBegin = {
16282 \group_begin:
16283 \tl_set:Nn
16284 \l_@@_latex_fancy_list_item_label_number_style_tl
16285 { #1 }
16286 \tl_set:Nn
16287 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16288 { #2 }
16289 \begin
16290 { enumerate }
16291 },
16292 fancyOlBeginTight = {
16293 \group_begin:
16294 \tl_set:Nn
16295 \l_@@_latex_fancy_list_item_label_number_style_tl
16296 { #1 }
16297 \tl_set:Nn
16298 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16299 { #2 }
16300 \begin
16301 { enumerate }
16302 [noitemsep]
16303 },
16304 fancyOlEnd(|Tight) = {
16305 \end { enumerate }
16306 \group_end:
16307 },
16308 fancyOlItemWithNumber = {
16309 \item
16310 [
16311 \@@_latex_fancy_list_item_label:VVn
16312 \l_@@_latex_fancy_list_item_label_number_style_tl
16313 \l_@@_latex_fancy_list_item_label_delimiter_style_tl

```

```

16314 { #1 }
16315]
16316 },
16317 } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16318 }
16319 { \@ifpackageloaded { paralist } {
16320 \markdownSetup { rendererPrototypes = {

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

16321 ulBeginTight = {
16322 \group_begin:
16323 \pltopsep=\topsep
16324 \plpartopsep=\partopsep
16325 \begin { compactitem }
16326 },
16327 ulEndTight = {
16328 \end { compactitem }
16329 \group_end:
16330 },
16331 fancyOlBegin = {
16332 \group_begin:
16333 \tl_set:Nn
16334 \l_@@_latex_fancy_list_item_label_number_style_tl
16335 { #1 }
16336 \tl_set:Nn
16337 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16338 { #2 }
16339 \begin { enumerate }
16340 },
16341 fancyOlEnd = {
16342 \end { enumerate }
16343 \group_end:
16344 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

16345 olBeginTight = {
16346 \group_begin:
16347 \plpartopsep=\partopsep
16348 \pltopsep=\topsep
16349 \begin { compactenum }
16350 },
16351 olEndTight = {
16352 \end { compactenum }

```

```

16353 \group_end:
16354 },
16355 fancyOlBeginTight = {
16356 \group_begin:
16357 \tl_set:Nn
16358 \l_@@_latex_fancy_list_item_label_number_style_tl
16359 { #1 }
16360 \tl_set:Nn
16361 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16362 { #2 }
16363 \plpartopsep=\partopsep
16364 \pltopsep=\topsep
16365 \begin { compactenum }
16366 },
16367 fancyOlEndTight = {
16368 \end { compactenum }
16369 \group_end:
16370 },
16371 fancyOlItemWithNumber = {
16372 \item
16373 [
16374 \@@_latex_fancy_list_item_label:VVn
16375 \l_@@_latex_fancy_list_item_label_number_style_tl
16376 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16377 { #1 }
16378]
16379 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

16380 dlBeginTight = {
16381 \group_begin:
16382 \plpartopsep=\partopsep
16383 \pltopsep=\topsep
16384 \begin { compactdesc }
16385 },
16386 dlEndTight = {
16387 \end { compactdesc }
16388 \group_end:
16389 }
16390 } }
16391 }
16392 {

```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

16393 \markdownSetup

```

```

16394 {
16395 rendererPrototypes = {
16396 ulBeginTight = \markdownRendererUlBegin,
16397 ulEndTight = \markdownRendererUlEnd,
16398 fancyOlBegin = \markdownRendererOlBegin,
16399 fancyOlEnd = \markdownRendererOlEnd,
16400 olBeginTight = \markdownRendererOlBegin,
16401 olEndTight = \markdownRendererOlEnd,
16402 fancyOlBeginTight = \markdownRendererOlBegin,
16403 fancyOlEndTight = \markdownRendererOlEnd,
16404 dlBeginTight = \markdownRendererDlBegin,
16405 dlEndTight = \markdownRendererDlEnd,
16406 },
16407 }
16408 } }
16409 \ExplSyntaxOff
16410 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

16411 \@ifpackageloaded{unicode-math}{
16412 \markdownSetup{rendererPrototypes={
16413 untickedBox = {\mdlgwhtsquare$},
16414 }}
16415 }{
16416 \RequirePackage{amssymb}
16417 \markdownSetup{rendererPrototypes={
16418 untickedBox = {\square$},
16419 }}
16420 }
16421 \RequirePackage{csvsimple}
16422 \RequirePackage{fancyvrb}
16423 \RequirePackage{graphicx}
16424 \markdownSetup{rendererPrototypes={
16425 hardLineBreak = {\},
16426 leftBrace = {\textbraceleft},
16427 rightBrace = {\textbraceright},
16428 dollarSign = {\textdollar},
16429 underscore = {\textunderscore},
16430 circumflex = {\textasciicircum},
16431 backslash = {\textbackslash},
16432 tilde = {\textasciitilde},
16433 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\text{\TeX}$  during the typesetting. Therefore, even if we don't know whether a span of text is

part of math formula or not when we are parsing markdown,<sup>37</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.

```
16434 codeSpan = {%
16435 \ifmmode
16436 \text{#1}%
16437 \else
16438 \texttt{#1}%
16439 \fi
16440 }}}
```

### 3.3.4.2 Content Blocks

In content block renderer prototypes, display the content as a table using the package `csvsimple` when the raw attribute is `csv`, display the content using the default templates of the package `luaxml` when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```
16441 \ExplSyntaxOn
16442 \markdownSetup{
16443 rendererPrototypes = {
16444 contentBlock = {
16445 \str_case:nnF
16446 { #1 }
16447 {
16448 { csv }
16449 {
16450 \begin { table }
16451 \begin { center }
16452 \csvautotabular { #3 }
16453 \end { center }
16454 \tl_if_empty:nF
16455 { #4 }
16456 { \caption { #4 } }
16457 \end { table }
16458 }
16459 { html }
16460 {
```

If we are using `TeX4ht`<sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

---

<sup>37</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.

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

16461 \cs_if_exist:NTF
16462 \HCode
16463 {
16464 \if_mode_vertical:
16465 \IgnorePar
16466 \fi:
16467 \EndP
16468 \special
16469 { t4ht* < #3 }
16470 \par
16471 \ShowPar
16472 }
16473 {
16474 \@@_luaxml_print_html:n
16475 { #3 }
16476 }
16477 }
16478 { tex }
16479 {
16480 \markdownEscape
16481 { #3 }
16482 }
16483 }
16484 {
16485 \markdownInput
16486 { #3 }
16487 }
16488 },
16489 },
16490 }
16491 \ExplSyntaxOff
16492 \markdownSetup{rendererPrototypes={
16493 ulBegin = {\begin{itemize}},
16494 ulEnd = {\end{itemize}},
16495 olBegin = {\begin{enumerate}},
16496 olItem = {\item{}},
16497 olItemWithNumber = {\item[#1.]},
16498 olEnd = {\end{enumerate}},
16499 dlBegin = {\begin{description}},
16500 dlItem = {\item[#1]},
16501 dlEnd = {\end{description}},
16502 emphasis = {\emph{#1}},
16503 tickedTextBox = {\\boxtimes},
16504 halfTickedTextBox = {\\boxdot}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

16505 \ExplSyntaxOn
16506 \seq_new:N

```



```

16507 \g_@@_header_identifiers_seq
16508 \markdownSetup
16509 {
16510 rendererPrototypes = {
16511 headerAttributeContextBegin = {
16512 \markdownSetup
16513 {
16514 rendererPrototypes = {
16515 attributeIdentifier = {
16516 \seq_gput_right:Nn
16517 \g_@@_header_identifiers_seq
16518 { ##1 }
16519 },
16520 },
16521 }
16522 },
16523 headerAttributeContextEnd = {
16524 \seq_map_inline:Nn
16525 \g_@@_header_identifiers_seq
16526 { \label { ##1 } }
16527 \seq_gclear:N
16528 \g_@@_header_identifiers_seq
16529 },
16530 },
16531 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

16532 \bool_new:N
16533 \l_@@_header_unnumbered_bool
16534 \markdownSetup
16535 {
16536 rendererPrototypes = {
16537 headerAttributeContextBegin += {
16538 \markdownSetup
16539 {
16540 rendererPrototypes = {
16541 attributeClassName = {
16542 \bool_if:nT
16543 {
16544 \str_if_eq_p:nn
16545 { ##1 }
16546 { unnumbered } &&
16547 ! \l_@@_header_unnumbered_bool
16548 }
16549 {
16550 \group_begin:

```

```

16551 \bool_set_true:N
16552 \l_@@_header_unnumbered_bool
16553 \c@secnumdepth = -2
16554 \markdownSetup
16555 {
16556 rendererPrototypes = {
16557 sectionBegin = {
16558 \group_begin:
16559 },
16560 sectionEnd = {
16561 \group_end:
16562 },
16563 },
16564 }
16565 },
16566 },
16567 },
16568 }
16569 },
16570 },
16571 }
16572 \ExplSyntaxOff
16573 \markdownSetup{rendererPrototypes={
16574 superscript = {#1},
16575 subscript = {\textsubscript{#1}},
16576 blockQuoteBegin = {\begin{quotation}},
16577 blockQuoteEnd = {\end{quotation}},
16578 inputVerbatim = {\VerbatimInput{#1}},
16579 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
16580 note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16581 \RequirePackage{ltxcmds}
16582 \ExplSyntaxOn
16583 \cs_gset_protected:Npn
16584 \markdownRendererInputFencedCodePrototype#1#2#3
16585 {
16586 \tl_if_empty:nTF
16587 { #2 }
16588 { \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.

```

16589 {
16590 \regex_extract_once:nnN
16591 { \w* }

```

```

16592 { #2 }
16593 \l_tmpa_seq
16594 \seq_pop_left:NN
16595 \l_tmpa_seq
16596 \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

16597 \ltx@ifpackageloaded
16598 { minted }
16599 {
16600 \catcode`\%=14\relax
16601 \catcode`\#=6\relax
16602 \exp_args:NV
16603 \inputminted
16604 \l_tmpa_tl
16605 { #1 }
16606 \catcode`\%=12\relax
16607 \catcode`\#=12\relax
16608 }
16609 {

```

When the listings package is loaded, use it for syntax highlighting.

```

16610 \ltx@ifpackageloaded
16611 { listings }
16612 { \lstinputlisting [language = \l_tmpa_tl] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

16613 { \markdownRendererInputFencedCode { #1 } { } { } }
16614 }
16615 }
16616 }
16617 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

16618 \ExplSyntaxOn
16619 \def\markdownLATEXStrongEmphasis#1{
16620 \str_if_in:NnTF
16621 \f@series
16622 { b }
16623 { \textnormal{#1} }
16624 { \textbf{#1} }
16625 }
16626 \ExplSyntaxOff
16627 \markdownSetup{rendererPrototypes={strongEmphasis={%
16628 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

16629 \@ifundefined{chapter}{%

```

```

16630 \markdownSetup{rendererPrototypes = {
16631 headingOne = {\section{#1}},
16632 headingTwo = {\subsection{#1}},
16633 headingThree = {\subsubsection{#1}},
16634 headingFour = {\paragraph{#1}},
16635 headingFive = {\subparagraph{#1}}}}
16636 }{%
16637 \markdownSetup{rendererPrototypes = {
16638 headingOne = {\chapter{#1}},
16639 headingTwo = {\section{#1}},
16640 headingThree = {\subsection{#1}},
16641 headingFour = {\subsubsection{#1}},
16642 headingFive = {\paragraph{#1}},
16643 headingSix = {\subparagraph{#1}}}}
16644 }%

```

### 3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

16645 \markdownSetup{
16646 rendererPrototypes = {
16647 ulItem = {%
16648 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
16649 },
16650 },
16651 }
16652 \def\markdownLaTeXUListItem{%
16653 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
16654 \item[\markdownLaTeXCheckbox]%
16655 \expandafter\@gobble
16656 \else
16657 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
16658 \item[\markdownLaTeXCheckbox]%
16659 \expandafter\expandafter\expandafter\@gobble
16660 \else
16661 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
16662 \item[\markdownLaTeXCheckbox]%
16663 \expandafter\expandafter\expandafter\expandafter
16664 \expandafter\expandafter\expandafter\@gobble
16665 \else
16666 \item{}%
16667 \fi
16668 \fi
16669 \fi
16670 }

```

### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using  $\text{\TeX}4\text{ht}$ <sup>39</sup>, we will pass HTML elements to the output HTML document unchanged.

```
16671 \@ifundefined{HCode}{\}{
16672 \markdownSetup{
16673 rendererPrototypes = {
16674 inlineHtmlTag = {%
16675 \ifvmode
16676 \IgnorePar
16677 \EndP
16678 \fi
16679 \HCode{#1}%
16680 },
16681 inputBlockHtmlElement = {%
16682 \ifvmode
16683 \IgnorePar
16684 \fi
16685 \EndP
16686 \special{t4ht*<#1}%
16687 \par
16688 \ShowPar
16689 },
16690 },
16691 }
16692 }
```

### 3.3.4.6 Citations

Here is a basic implementation for citations that uses the  $\text{\LaTeX}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `Bib $\text{\LaTeX}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
16693 \newcount\markdownLaTeXCitationsCounter
16694
16695 % Basic implementation
16696 \long\def@gobblethree#1#2#3{%
16697 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
16698 \advance\markdownLaTeXCitationsCounter by 1\relax
16699 \ifx\relax#4\relax
16700 \ifx\relax#5\relax
16701 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16702 \relax
16703 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
16704 \expandafter\expandafter\expandafter
16705 \expandafter\expandafter\expandafter\expandafter
```

---

<sup>39</sup>See <https://tug.org/tex4ht/>.

```

16706 \@gobblethree
16707 \fi
16708 \else% Before a postnote (#5), dump the accumulator
16709 \ifx\relax#1\relax\else
16710 \cite{#1}%
16711 \fi
16712 \cite[#5]{#6}%
16713 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16714 \relax
16715 \else
16716 \expandafter\expandafter\expandafter
16717 \expandafter\expandafter\expandafter\expandafter
16718 \expandafter\expandafter\expandafter
16719 \expandafter\expandafter\expandafter\expandafter
16720 \markdownLaTeXBasicCitations
16721 \fi
16722 \expandafter\expandafter\expandafter
16723 \expandafter\expandafter\expandafter\expandafter{%
16724 \expandafter\expandafter\expandafter
16725 \expandafter\expandafter\expandafter\expandafter}%
16726 \expandafter\expandafter\expandafter
16727 \expandafter\expandafter\expandafter\expandafter{%
16728 \expandafter\expandafter\expandafter
16729 \expandafter\expandafter\expandafter\expandafter}%
16730 \expandafter\expandafter\expandafter
16731 \@gobblethree
16732 \fi
16733 \else% Before a prenote (#4), dump the accumulator
16734 \ifx\relax#1\relax\else
16735 \cite{#1}%
16736 \fi
16737 \ifnum\markdownLaTeXCitationsCounter>1\relax
16738 \space % Insert a space before the prenote in later citations
16739 \fi
16740 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
16741 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16742 \relax
16743 \else
16744 \expandafter\expandafter\expandafter
16745 \expandafter\expandafter\expandafter\expandafter
16746 \markdownLaTeXBasicCitations
16747 \fi
16748 \expandafter\expandafter\expandafter{%
16749 \expandafter\expandafter\expandafter}%
16750 \expandafter\expandafter\expandafter{%
16751 \expandafter\expandafter\expandafter}%
16752 \expandafter

```

```

16753 \gobblethree
16754 \fi\markdownLaTeXBasicCitations{#1#2#6},}
16755 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
16756
16757 % Natbib implementation
16758 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
16759 \advance\markdownLaTeXCitationsCounter by 1\relax
16760 \ifx\relax#3\relax
16761 \ifx\relax#4\relax
16762 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16763 \relax
16764 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
16765 \expandafter\expandafter\expandafter
16766 \expandafter\expandafter\expandafter\expandafter
16767 \@gobbletwo
16768 \fi
16769 \else% Before a postnote (#4), dump the accumulator
16770 \ifx\relax#1\relax\else
16771 \citep{#1}%
16772 \fi
16773 \citep[] [#4]{#5}%
16774 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16775 \relax
16776 \else
16777 \expandafter\expandafter\expandafter
16778 \expandafter\expandafter\expandafter\expandafter
16779 \expandafter\expandafter\expandafter
16780 \expandafter\expandafter\expandafter\expandafter
16781 \markdownLaTeXNatbibCitations
16782 \fi
16783 \expandafter\expandafter\expandafter
16784 \expandafter\expandafter\expandafter\expandafter{%
16785 \expandafter\expandafter\expandafter
16786 \expandafter\expandafter\expandafter\expandafter}%
16787 \expandafter\expandafter\expandafter
16788 \@gobbletwo
16789 \fi
16790 \else% Before a prenote (#3), dump the accumulator
16791 \ifx\relax#1\relax\relax\else
16792 \citep{#1}%
16793 \fi
16794 \citep[#3] [#4]{#5}%
16795 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16796 \relax
16797 \else
16798 \expandafter\expandafter\expandafter
16799 \expandafter\expandafter\expandafter\expandafter

```

```

16800 \markdownLaTeXNatbibCitations
16801 \fi
16802 \expandafter\expandafter\expandafter{%
16803 \expandafter\expandafter\expandafter}%
16804 \expandafter
16805 \@gobbletwo
16806 \fi\markdownLaTeXNatbibCitations{#1,#5}}
16807 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
16808 \advance\markdownLaTeXCitationsCounter by 1\relax
16809 \ifx\relax#3\relax
16810 \ifx\relax#4\relax
16811 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16812 \relax
16813 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
16814 \expandafter\expandafter\expandafter
16815 \expandafter\expandafter\expandafter\expandafter
16816 \@gobbletwo
16817 \fi
16818 \else% After a prenote or a postnote, dump the accumulator
16819 \ifx\relax#1\relax\else
16820 \citet{#1}%
16821 \fi
16822 , \citet[#3][#4]{#5}%
16823 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
16824 \relax
16825 ,
16826 \else
16827 \ifnum
16828 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
16829 \relax
16830 ,
16831 \fi
16832 \fi
16833 \expandafter\expandafter\expandafter
16834 \expandafter\expandafter\expandafter\expandafter
16835 \markdownLaTeXNatbibTextCitations
16836 \expandafter\expandafter\expandafter
16837 \expandafter\expandafter\expandafter\expandafter{%
16838 \expandafter\expandafter\expandafter
16839 \expandafter\expandafter\expandafter\expandafter}%
16840 \expandafter\expandafter\expandafter
16841 \@gobbletwo
16842 \fi
16843 \else% After a prenote or a postnote, dump the accumulator
16844 \ifx\relax#1\relax\relax\else
16845 \citet{#1}%
16846 \fi

```



```

16847 , \citet[#3][#4]{#5}%
16848 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
16849 \relax
16850 ,
16851 \else
16852 \ifnum
16853 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
16854 \relax
16855 ,
16856 \fi
16857 \fi
16858 \expandafter\expandafter\expandafter
16859 \markdownLaTeXNatbibTextCitations
16860 \expandafter\expandafter\expandafter{%
16861 \expandafter\expandafter\expandafter}%
16862 \expandafter
16863 \@gobbletwo
16864 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
16865
16866 % BibLaTeX implementation
16867 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
16868 \advance\markdownLaTeXCitationsCounter by 1\relax
16869 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16870 \relax
16871 \autocites#1[#3][#4]{#5}%
16872 \expandafter\@gobbletwo
16873 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
16874 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
16875 \advance\markdownLaTeXCitationsCounter by 1\relax
16876 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16877 \relax
16878 \textcites#1[#3][#4]{#5}%
16879 \expandafter\@gobbletwo
16880 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
16881
16882 \markdownSetup{rendererPrototypes = {
16883 cite = {%
16884 \markdownLaTeXCitationsCounter=1%
16885 \def\markdownLaTeXCitationsTotal{#1}%
16886 \@ifundefined{autocites}{%
16887 \ifundefined{citep}{%
16888 \expandafter\expandafter\expandafter
16889 \markdownLaTeXBasicCitations
16890 \expandafter\expandafter\expandafter{%
16891 \expandafter\expandafter\expandafter}%
16892 \expandafter\expandafter\expandafter{%
16893 \expandafter\expandafter\expandafter}%

```

```

16894 }{%
16895 \expandafter\expandafter\expandafter
16896 \markdownLaTeXNatbibCitations
16897 \expandafter\expandafter\expandafter{%
16898 \expandafter\expandafter\expandafter}%
16899 }%
16900 }{%
16901 \expandafter\expandafter\expandafter
16902 \markdownLaTeXBibLaTeXCitations
16903 \expandafter{\expandafter}%
16904 },
16905 textCite = {%
16906 \markdownLaTeXCitationsCounter=1%
16907 \def\markdownLaTeXCitationsTotal{#1}%
16908 \@ifundefined{autocites}{%
16909 \@ifundefined{citep}{%
16910 \expandafter\expandafter\expandafter
16911 \markdownLaTeXBasicTextCitations
16912 \expandafter\expandafter\expandafter{%
16913 \expandafter\expandafter\expandafter}%
16914 \expandafter\expandafter\expandafter{%
16915 \expandafter\expandafter\expandafter}%
16916 }{%
16917 \expandafter\expandafter\expandafter
16918 \markdownLaTeXNatbibTextCitations
16919 \expandafter\expandafter\expandafter{%
16920 \expandafter\expandafter\expandafter}%
16921 }%
16922 }{%
16923 \expandafter\expandafter\expandafter
16924 \markdownLaTeXBibLaTeXTextCitations
16925 \expandafter{\expandafter}%
16926 }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

16927 \RequirePackage{url}
16928 \RequirePackage{expl3}
16929 \ExplSyntaxOn
16930 \cs_gset_protected:Npn
16931 \markdownRendererLinkPrototype
16932 #1#2#3#4
16933 {
16934 \tl_set:Nn \l_tmpa_tl { #1 }
16935 \tl_set:Nn \l_tmpb_tl { #2 }
16936 \bool_set:Nn

```

```

16937 \l_tmpa_bool
16938 {
16939 \tl_if_eq_p:NN
16940 \l_tmpa_tl
16941 \l_tmpb_tl
16942 }
16943 \tl_set:Nn \l_tmpa_tl { #4 }
16944 \bool_set:Nn
16945 \l_tmpb_bool
16946 {
16947 \tl_if_empty_p:N
16948 \l_tmpa_tl
16949 }

```

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.

```

16950 \bool_if:nTF
16951 {
16952 \l_tmpa_bool && \l_tmpb_bool
16953 }
16954 {
16955 \markdownLaTeXRendererAutolink { #2 } { #3 }
16956 }
16957 {
16958 \markdownLaTeXRendererDirectOrIndirectLink
16959 { #1 } { #2 } { #3 } { #4 }
16960 }
16961 }
16962 \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.

```

16963 \tl_set:Nn
16964 \l_tmpa_tl
16965 { #2 }
16966 \tl_trim_spaces:N
16967 \l_tmpa_tl
16968 \tl_set:Nx
16969 \l_tmpb_tl
16970 {
16971 \tl_range:Nnn
16972 \l_tmpa_tl
16973 { 1 }
16974 { 1 }
16975 }
16976 \str_if_eq:NNTF
16977 \l_tmpb_tl

```

```

16978 \c_hash_str
16979 {
16980 \tl_set:Nx
16981 \l_tmpb_tl
16982 {
16983 \tl_range:Nnn
16984 \l_tmpa_tl
16985 { 2 }
16986 { -1 }
16987 }
16988 \exp_args:NV
16989 \ref
16990 \l_tmpb_tl
16991 }
16992 {
16993 \url { #2 }
16994 }
16995 }
16996 \ExplSyntaxOff
16997 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
16998 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}%

```

### 3.3.4.8 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

16999 \newcount\markdownLaTeXRowCount
17000 \newcount\markdownLaTeXRowTotal
17001 \newcount\markdownLaTeXColumnCounter
17002 \newcount\markdownLaTeXColumnTotal
17003 \newtoks\markdownLaTeXTable
17004 \newtoks\markdownLaTeXTableAlignment
17005 \newtoks\markdownLaTeXTableEnd
17006 \AtBeginDocument{%
17007 \ifpackageloaded{booktabs}{%
17008 \def\markdownLaTeXTopRule{\toprule}%
17009 \def\markdownLaTeXMidRule{\midrule}%
17010 \def\markdownLaTeXBottomRule{\bottomrule}%
17011 }{%
17012 \def\markdownLaTeXTopRule{\hline}%
17013 \def\markdownLaTeXMidRule{\hline}%
17014 \def\markdownLaTeXBottomRule{\hline}%
17015 }%
17016 }
17017 \markdownSetup{rendererPrototypes={
17018 table = {%
17019 \markdownLaTeXTable={}%

```

```

17020 \markdownLaTeXTableAlignment={}%
17021 \markdownLaTeXTableEnd={%
17022 \markdownLaTeXBottomRule
17023 \end{tabular}}}%
17024 \ifx\empty#1\empty\else
17025 \addto@hook\markdownLaTeXTable{%
17026 \begin{table}
17027 \centering}%
17028 \addto@hook\markdownLaTeXTableEnd{%
17029 \caption{#1}}}%
17030 \fi
17031 }
17032 }}

```

If the `tableAttributes` option is enabled, we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing tables.

```

17033 \ExplSyntaxOn
17034 \seq_new:N
17035 \l_@@_table_identifiers_seq
17036 \markdownSetup {
17037 rendererPrototypes = {
17038 table += {
17039 \seq_map_inline:Nn
17040 \l_@@_table_identifiers_seq
17041 {
17042 \addto@hook
17043 \markdownLaTeXTableEnd
17044 { \label { ##1 } }
17045 }
17046 },
17047 }
17048 }
17049 \markdownSetup {
17050 rendererPrototypes = {
17051 tableAttributeContextBegin = {
17052 \group_begin:
17053 \markdownSetup {
17054 rendererPrototypes = {
17055 attributeIdentifier = {
17056 \seq_put_right:Nn
17057 \l_@@_table_identifiers_seq
17058 { ##1 }
17059 },
17060 },
17061 }
17062 },
17063 tableAttributeContextEnd = {

```

```

17064 \group_end:
17065 },
17066 },
17067 }
17068 \ExplSyntaxOff
17069 \markdownSetup{rendererPrototypes={
17070 table += {%
17071 \ifx\empty#1\empty\else
17072 \addto@hook\markdownLaTeXTableEnd{%
17073 \end{table}}}%
17074 \fi
17075 \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
17076 \markdownLaTeXRowCount=0%
17077 \markdownLaTeXRowTotal=#2%
17078 \markdownLaTeXColumnTotal=#3%
17079 \markdownLaTeXRenderTableRow
17080 }
17081 }}
17082 \def\markdownLaTeXRenderTableRow#1{%
17083 \markdownLaTeXColumnCounter=0%
17084 \ifnum\markdownLaTeXRowCount=0\relax
17085 \markdownLaTeXReadAlignments#1%
17086 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
17087 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
17088 \the\markdownLaTeXTableAlignment}}}%
17089 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
17090 \else
17091 \markdownLaTeXRenderTableCell#1%
17092 \fi
17093 \ifnum\markdownLaTeXRowCount=1\relax
17094 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
17095 \fi
17096 \advance\markdownLaTeXRowCount by 1\relax
17097 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
17098 \the\markdownLaTeXTable
17099 \the\markdownLaTeXTableEnd
17100 \expandafter\@gobble
17101 \fi\markdownLaTeXRenderTableRow}
17102 \def\markdownLaTeXReadAlignments#1{%
17103 \advance\markdownLaTeXColumnCounter by 1\relax
17104 \if#1d%
17105 \addto@hook\markdownLaTeXTableAlignment{1}%
17106 \else
17107 \addto@hook\markdownLaTeXTableAlignment{#1}%
17108 \fi
17109 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
17110 \expandafter\@gobble

```

```

17111 \fi\markdownLaTeXReadAlignments}
17112 \def\markdownLaTeXRenderTableCell#1{%
17113 \advance\markdownLaTeXColumnCounter by 1\relax
17114 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
17115 \addto@hook\markdownLaTeXTable{#1&}%
17116 \else
17117 \addto@hook\markdownLaTeXTable{#1\\}%
17118 \expandafter\@gobble
17119 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.9 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

17120
17121 \markdownIfOption{lineBlocks}{%
17122 \RequirePackage{verse}
17123 \markdownSetup{rendererPrototypes={
17124 lineBlockBegin = {%
17125 \begingroup
17126 \def\markdownRendererHardLineBreak{\\}%
17127 \begin{verse}%
17128 },
17129 lineBlockEnd = {%
17130 \end{verse}%
17131 \endgroup
17132 },
17133 }}
17134 }{}
17135

```

### 3.3.4.10 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

17136 \ExplSyntaxOn
17137 \keys_define:nn
17138 { markdown / jekyllData }
17139 {
17140 author .code:n = {
17141 \author
17142 { #1 }
17143 },
17144 date .code:n = {
17145 \date
17146 { #1 }

```

```

17147 },
17148 title .code:n = {
17149 \title
17150 { #1 }

```

To complement the default setup of our key-values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away, temporarily resetting the category codes of the percent sign and the hash sign back to comment and parameter, respectively.

```

17151 \char_set_catcode_comment:N \%
17152 \char_set_catcode_parameter:N \#
17153 \AddToHook
17154 { begindocument / end }
17155 { \maketitle }
17156 \char_set_catcode_other:N \%
17157 \char_set_catcode_other:N \#
17158 },
17159 }

```

### 3.3.4.11 Marked Text

If the `mark` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement marked text.

```

17160 \@@_if_option:nT
17161 { mark }
17162 {
17163 \sys_if_engine luatex:TF
17164 {
17165 \RequirePackage
17166 { luacolor }
17167 \RequirePackage
17168 { lua-ul }
17169 \markdownSetup
17170 {
17171 rendererPrototypes = {
17172 mark = {
17173 \highLight
17174 { #1 }
17175 },
17176 }
17177 }
17178 }
17179 {
17180 \RequirePackage
17181 { xcolor }

```



```

17182 \RequirePackage
17183 { soul }
17184 \markdownSetup
17185 {
17186 rendererPrototypes = {
17187 mark = {
17188 \hl
17189 { #1 }
17190 },
17191 }
17192 }
17193 }
17194 }

```

### 3.3.4.12 Strike-Through

If the `strikeThrough` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement strike-throughs.

```

17195 \@@_if_option:nT
17196 { strikeThrough }
17197 {
17198 \sys_if_engine luatex:TF
17199 {
17200 \RequirePackage
17201 { lua-ul }
17202 \markdownSetup
17203 {
17204 rendererPrototypes = {
17205 strikeThrough = {
17206 \strikeThrough
17207 { #1 }
17208 },
17209 }
17210 }
17211 }
17212 {
17213 \RequirePackage
17214 { soul }
17215 \markdownSetup
17216 {
17217 rendererPrototypes = {
17218 strikeThrough = {
17219 \st
17220 { #1 }
17221 },
17222 }
17223 }
17224 }
17225 }

```

```

17224 }
17225 }

```

### 3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding values and we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing figures.

```

17226 \seq_new:N
17227 \l_@@_image_identifiers_seq
17228 \markdownSetup {
17229 rendererPrototypes = {
17230 image = {
17231 \tl_if_empty:nTF
17232 { #4 }
17233 {
17234 \begin { center }
17235 \includegraphics
17236 [alt = { #1 }]
17237 { #3 }
17238 \end { center }
17239 }
17240 {
17241 \begin { figure }
17242 \begin { center }
17243 \includegraphics
17244 [alt = { #1 }]
17245 { #3 }
17246 \caption { #4 }
17247 \seq_map_inline:Nn
17248 \l_@@_image_identifiers_seq
17249 { \label { ##1 } }
17250 \end { center }
17251 \end { figure }
17252 }
17253 },
17254 }
17255 }
17256 \@@_if_option:nT
17257 { linkAttributes }
17258 {
17259 \RequirePackage { graphicx }
17260 }

```

```

17261 \markdownSetup {
17262 rendererPrototypes = {
17263 imageAttributeContextBegin = {
17264 \group_begin:
17265 \markdownSetup {
17266 rendererPrototypes = {
17267 attributeIdentifier = {
17268 \seq_put_right:Nn
17269 \l_@@_image_identifiers_seq
17270 { ##1 }
17271 },
17272 attributeKeyValue = {
17273 \setkeys
17274 { Gin }
17275 { { ##1 } = { ##2 } }
17276 },
17277 },
17278 }
17279 },
17280 imageAttributeContextEnd = {
17281 \group_end:
17282 },
17283 },
17284 }
17285 \ExplSyntaxOff

```

### 3.3.4.14 Raw Attributes

In the raw block and inline raw span renderer prototypes, display the content using the default templates of the package `luaxml` when the raw attribute is `html` and default to the plain TeX renderer prototypes otherwise, translating raw attribute `latex` to `tex`.

```

17286 \ExplSyntaxOn
17287 \cs_new:Nn
17288 \@@_luaxml_print_html:n
17289 {
17290 \luabridge_now:n
17291 {
17292 local~input_file = assert(io.open(" #1 ", "r"))
17293 local~input = assert(input_file:read("*a"))
17294 assert(input_file:close())
17295 input = "<body>" .. input .. "</body>"
17296 local~dom = require("luaxml-domobject").html_parse(input)
17297 local~output = require("luaxml-htmltemplates"):process_dom(dom)
17298 print(output)
17299 }
17300 }

```

```

17301 \cs_gset_protected:Npn
17302 \markdownRendererInputRawInlinePrototype#1#2
17303 {
17304 \str_case:nnF
17305 { #2 }
17306 {
17307 { latex }
17308 {
17309 \@@_plain_tex_default_input_raw_inline:nn
17310 { #1 }
17311 { tex }
17312 }
17313 { html }
17314 {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>40</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17315 \cs_if_exist:NTF
17316 \HCode
17317 {
17318 \if_mode_vertical:
17319 \IgnorePar
17320 \EndP
17321 \fi:
17322 \special
17323 { t4ht* < #1 }
17324 }
17325 {
17326 \@@_luaxml_print_html:n
17327 { #1 }
17328 }
17329 }
17330 }
17331 {
17332 \@@_plain_tex_default_input_raw_inline:nn
17333 { #1 }
17334 { #2 }
17335 }
17336 }
17337 \cs_gset_protected:Npn
17338 \markdownRendererInputRawBlockPrototype#1#2
17339 {
17340 \str_case:nnF
17341 { #2 }
17342 {
17343 { latex }

```

---

<sup>40</sup>See <https://tug.org/tex4ht/>.

```

17344 {
17345 \@@_plain_tex_default_input_raw_block:nn
17346 { #1 }
17347 { tex }
17348 }
17349 { html }
17350 {

```

If we are using  $\text{T}_{\text{E}}\text{X}4\text{ht}$ <sup>41</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17351 \cs_if_exist:NTF
17352 \HCode
17353 {
17354 \if_mode_vertical:
17355 \IgnorePar
17356 \fi:
17357 \EndP
17358 \special
17359 { t4ht* < #1 }
17360 \par
17361 \ShowPar
17362 }
17363 {
17364 \@@_luaxml_print_html:n
17365 { #1 }
17366 }
17367 }
17368 }
17369 {
17370 \@@_plain_tex_default_input_raw_block:nn
17371 { #1 }
17372 { #2 }
17373 }
17374 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  labels for referencing the last  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  counter that has been incremented in e.g. ordered lists.

```

17375 \seq_new:N
17376 \l_@@_bracketed_span_identifiers_seq
17377 \markdownSetup {
17378 rendererPrototypes = {
17379 bracketedSpanAttributeContextBegin = {
17380 \group_begin:

```

---

<sup>41</sup>See <https://tug.org/tex4ht/>.

```

17381 \markdownSetup {
17382 rendererPrototypes = {
17383 attributeIdentifier = {
17384 \seq_put_right:Nn
17385 \l_@@_bracketed_span_identifiers_seq
17386 { ##1 }
17387 },
17388 },
17389 }
17390 },
17391 bracketedSpanAttributeContextEnd = {
17392 \seq_map_inline:Nn
17393 \l_@@_bracketed_span_identifiers_seq
17394 { \label { ##1 } }
17395 \group_end:
17396 },
17397 },
17398 }
17399 \ExplSyntaxOff
17400 \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.

```

17401 \newcommand\markdownMakeOther{%
17402 \count0=128\relax
17403 \loop
17404 \catcode\count0=11\relax
17405 \advance\count0 by 1\relax
17406 \ifnum\count0<256\repeat}%

```

## 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L<sup>A</sup>T<sub>E</sub>X package.

```

17407 \def\markdownMakeOther{%
17408 \count0=128\relax
17409 \loop
17410 \catcode\count0=11\relax
17411 \advance\count0 by 1\relax
17412 \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConT<sub>E</sub>Xt.

```

17413 \catcode`\|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```

17414 \long\def\inputmarkdown{%
17415 \dosingleempty
17416 \doinputmarkdown}%
17417 \long\def\doinputmarkdown[#1]#2{%
17418 \begingroup
17419 \iffirstargument
17420 \setupmarkdown[#1]%
17421 \fi
17422 \markdownInput{#2}%
17423 \endgroup}%
17424 \long\def\inputyaml{%
17425 \dosingleempty
17426 \doinputyaml}%
17427 \long\def\doinputyaml[#1]#2{%
17428 \doinputmarkdown
17429 [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s T<sub>E</sub>X, trailing spaces are removed very early on when a line is being put to the input buffer. [20, sec. 31]. According to Eijkhout [21, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)T<sub>E</sub>X, but ConT<sub>E</sub>Xt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConT<sub>E</sub>Xt MkIV and therefore to insert hard line breaks into markdown text.

```

17430 \startluacode
17431 document.markdown_buffering = false
17432 local function preserve_trailing_spaces(line)
17433 if document.markdown_buffering then
17434 line = line:gsub("[\t][\t]$", "\t\t")
17435 end

```

```

17436 return line
17437 end
17438 resolvers.installinputlinehandler(preserve_trailing_spaces)
17439 \stopluacode
17440 \begingroup
17441 \catcode`\|=0%
17442 \catcode`\|=12%
17443 |gdef|startmarkdown{%
17444 |ctxlua{document.markdown_buffering = true}%
17445 |markdownReadAndConvert{\stopmarkdown}%
17446 {|stopmarkdown}}%
17447 |gdef|stopmarkdown{%
17448 |ctxlua{document.markdown_buffering = false}%
17449 |markdownEnd}%
17450 |gdef|startyaml{%
17451 |begingroup
17452 |ctxlua{document.markdown_buffering = true}%
17453 |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
17454 |markdownReadAndConvert{\stopyaml}%
17455 {|stopyaml}}%
17456 |gdef|stopyaml{%
17457 |ctxlua{document.markdown_buffering = false}%
17458 |yamlEnd}%
17459 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\text{\TeX}$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\text{\ConTeXt}$  themes provided with the Markdown package.

```

17460 \ExplSyntaxOn
17461 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
17462 \prop_new:N \g_@@_context_loaded_themes_versions_prop
17463 \cs_gset:Nn
17464 \@@_load_theme:nnn
17465 {

```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_themes_prop` or a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain  $\text{\TeX}$  theme instead.

```

17466 \bool_if:nTF
17467 {
17468 \bool_lazy_or_p:nn
17469 {
17470 \prop_if_in_p:Nn

```



```

17471 \g_@@_context_built_in_themes_prop
17472 { #1 }
17473 }
17474 {
17475 \file_if_exist_p:n
17476 { t - markdown theme #3.tex }
17477 }
17478 }
17479 {
17480 \prop_get:NnNTF
17481 \g_@@_context_loaded_themes_linenos_prop
17482 { #1 }
17483 \l_tmpa_tl
17484 {
17485 \prop_get:NnN
17486 \g_@@_context_loaded_themes_versions_prop
17487 { #1 }
17488 \l_tmpb_tl
17489 \str_if_eq:nVTF
17490 { #2 }
17491 \l_tmpb_tl
17492 {
17493 \msg_warning:nnnVn
17494 { markdown }
17495 { repeatedly-loaded-context-theme }
17496 { #1 }
17497 \l_tmpa_tl
17498 { #2 }
17499 }
17500 }
17501 {
17502 \msg_error:nnnnVV
17503 { markdown }
17504 { different-versions-of-context-theme }
17505 { #1 }
17506 { #2 }
17507 \l_tmpb_tl
17508 \l_tmpa_tl
17509 }
17510 }
17511 {
17512 \prop_gput:Nnx
17513 \g_@@_context_loaded_themes_linenos_prop
17514 { #1 }
17515 { \tex_the:D \tex_inputlineno:D } % noqa: W200
17516 \prop_gput:Nnn
17517 \g_@@_context_loaded_themes_versions_prop
17518 { #1 }

```

```

17518 { #2 }
Load built-in plain TeX themes from the prop \g_@@_context_built_in_themes_prop
and from the filesystem otherwise.
17519 \prop_if_in:NnTF
17520 \g_@@_context_built_in_themes_prop
17521 { #1 }
17522 {
17523 \msg_info:nnnn
17524 { markdown }
17525 { loading-built-in-context-theme }
17526 { #1 }
17527 { #2 }
17528 \prop_item:Nn
17529 \g_@@_context_built_in_themes_prop
17530 { #1 }
17531 }
17532 {
17533 \msg_info:nnnn
17534 { markdown }
17535 { loading-context-theme }
17536 { #1 }
17537 { #2 }
17538 \usemodule
17539 [t]
17540 [markdown theme #3]
17541 }
17542 }
17543 }
17544 {
17545 \@@_plain_tex_load_theme:nnn
17546 { #1 }
17547 { #2 }
17548 { #3 }
17549 }
17550 }
17551 \msg_new:nnn
17552 { markdown }
17553 { loading-built-in-context-theme }
17554 { Loading-version~#2~of~built-in-ConTeXt-Markdown-theme~#1 }
17555 \msg_new:nnn
17556 { markdown }
17557 { loading-context-theme }
17558 { Loading-version~#2~of~ConTeXt-Markdown-theme~#1 }
17559 \msg_new:nnn
17560 { markdown }
17561 { repeatedly-loaded-context-theme }

```

```

17562 {
17563 Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
17564 loaded~on~line~#2,~not~loading~it~again
17565 }
17566 \msg_new:nnn
17567 { markdown }
17568 { different-versions-of-context-theme }
17569 {
17570 Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
17571 but~version~#3~has~already~been~loaded~on~line~#4
17572 }
17573 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```

17574 \markdownLoadPlainTeXTheme

```

Next, the ConTeXt theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option [plain](#) has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

17575 \markdownIfOption{plain}{\iffalse}{\iftrue}
17576 \def\markdownRendererHardLineBreakPrototype{\blank}%
17577 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
17578 \def\markdownRendererRightBracePrototype{\textbraceright}%
17579 \def\markdownRendererDollarSignPrototype{\textdollar}%
17580 \def\markdownRendererPercentSignPrototype{\percent}%
17581 \def\markdownRendererUnderscorePrototype{\textunderscore}%
17582 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
17583 \def\markdownRendererBackslashPrototype{\textbackslash}%
17584 \def\markdownRendererTildePrototype{\textasciitilde}%
17585 \def\markdownRendererPipePrototype{\char`|}%
17586 \def\markdownRendererLinkPrototype#1#2#3#4{%
17587 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
17588 \fi\texttt<\hyphenatedurl{#3}>}}%
17589 \usemodule[database]
17590 \defineseparatedlist
17591 [MarkdownConTeXtCSV]
17592 [separator={,},
17593 before=\bTABLE,after=\eTABLE,
17594 first=\bTR,last=\eTR,
17595 left=\bTD,right=\eTD]
17596 \def\markdownConTeXtCSV{csv}

```

```

17597 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
17598 \def\markdownConTeXtCSV@arg{#1}%
17599 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
17600 \placetable[] [tab:#1]{#4}{%
17601 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
17602 \else
17603 \markdownInput{#3}%
17604 \fi}%
17605 \def\markdownRendererImagePrototype#1#2#3#4{%
17606 \placefigure[] []{#4}{\externalfigure[#3]}%
17607 \def\markdownRendererUlBeginPrototype{\startitemize}%
17608 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
17609 \def\markdownRendererUlItemPrototype{\item}%
17610 \def\markdownRendererUlEndPrototype{\stopitemize}%
17611 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
17612 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
17613 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
17614 \def\markdownRendererOlItemPrototype{\item}%
17615 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
17616 \def\markdownRendererOlEndPrototype{\stopitemize}%
17617 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
17618 \definedescription
17619 [MarkdownConTeXtDlItemPrototype]
17620 [location=hanging,
17621 margin=standard,
17622 headstyle=bold]%
17623 \definestartstop
17624 [MarkdownConTeXtDlPrototype]
17625 [before=\blank,
17626 after=\blank]%
17627 \definestartstop
17628 [MarkdownConTeXtDlTightPrototype]
17629 [before=\blank\startpacked,
17630 after=\stoppacked\blank]%
17631 \def\markdownRendererDlBeginPrototype{%
17632 \startMarkdownConTeXtDlPrototype}%
17633 \def\markdownRendererDlBeginTightPrototype{%
17634 \startMarkdownConTeXtDlTightPrototype}%
17635 \def\markdownRendererDlItemPrototype#1{%
17636 \startMarkdownConTeXtDlItemPrototype{#1}}%
17637 \def\markdownRendererDlItemEndPrototype{%
17638 \stopMarkdownConTeXtDlItemPrototype}%
17639 \def\markdownRendererDlEndPrototype{%
17640 \stopMarkdownConTeXtDlPrototype}%
17641 \def\markdownRendererDlEndTightPrototype{%
17642 \stopMarkdownConTeXtDlTightPrototype}%
17643 \def\markdownRendererEmphasisPrototype#1{\em{#1}}%

```

```

17644 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}}%
17645 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
17646 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
17647 \def\markdownRendererLineBlockBeginPrototype{%
17648 \begingroup
17649 \def\markdownRendererHardLineBreak{
17650 }%
17651 \startlines
17652 }%
17653 \def\markdownRendererLineBlockEndPrototype{%
17654 \stoplines
17655 \endgroup
17656 }%
17657 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

17658 \ExplSyntaxOn
17659 \cs_gset:Npn
17660 \markdownRendererInputFencedCodePrototype#1#2#3
17661 {
17662 \tl_if_empty:nTF
17663 { #2 }
17664 { \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

```

```

17665 {

```

```

17666 \regex_extract_once:nnN
17667 { \w* }
17668 { #2 }
17669 \l_tmpa_seq
17670 \seq_pop_left:NN
17671 \l_tmpa_seq
17672 \l_tmpa_tl
17673 \typefile[\l_tmpa_tl][] {#1}
17674 }
17675 }
17676 \ExplSyntaxOff
17677 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
17678 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
17679 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
17680 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
17681 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
17682 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
17683 \def\markdownRendererThematicBreakPrototype{%
17684 \blackrule[height=1pt, width=\hsize]}%
17685 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
17686 \def\markdownRendererTickedBoxPrototype{\boxtimes}%
17687 \def\markdownRendererHalfTickedBoxPrototype{\boxdot}%
17688 \def\markdownRendererUntickedBoxPrototype{\square}%
17689 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
17690 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
17691 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
17692 \def\markdownRendererDisplayMathPrototype#1{%
17693 \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

17694 \newcount\markdownConTeXtRowCounter
17695 \newcount\markdownConTeXtRowTotal
17696 \newcount\markdownConTeXtColumnCounter
17697 \newcount\markdownConTeXtColumnTotal
17698 \newtoks\markdownConTeXtTable
17699 \newtoks\markdownConTeXtTableFloat
17700 \def\markdownRendererTablePrototype#1#2#3{%
17701 \markdownConTeXtTable={}%
17702 \ifx\empty#1\empty
17703 \markdownConTeXtTableFloat={%
17704 \the\markdownConTeXtTable}%
17705 \else
17706 \markdownConTeXtTableFloat={%
17707 \placetable{#1}{\the\markdownConTeXtTable}}%
17708 \fi

```

```

17709 \beginngroup
17710 \setupTABLE[r][each][topframe=off, bottomframe=off,
17711 leftframe=off, rightframe=off]
17712 \setupTABLE[c][each][topframe=off, bottomframe=off,
17713 leftframe=off, rightframe=off]
17714 \setupTABLE[r][1][topframe=on, bottomframe=on]
17715 \setupTABLE[r][#1][bottomframe=on]
17716 \markdownConTeXtRowCounter=0%
17717 \markdownConTeXtRowTotal=#2%
17718 \markdownConTeXtColumnTotal=#3%
17719 \markdownConTeXtRenderTableRow}
17720 \def\markdownConTeXtRenderTableRow#1{%
17721 \markdownConTeXtColumnCounter=0%
17722 \ifnum\markdownConTeXtRowCounter=0\relax
17723 \markdownConTeXtReadAlignments#1%
17724 \markdownConTeXtTable={\bTABLE}%
17725 \else
17726 \markdownConTeXtTable=\expandafter{%
17727 \the\markdownConTeXtTable\bTR}%
17728 \markdownConTeXtRenderTableCell#1%
17729 \markdownConTeXtTable=\expandafter{%
17730 \the\markdownConTeXtTable\eTR}%
17731 \fi
17732 \advance\markdownConTeXtRowCounter by 1\relax
17733 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
17734 \markdownConTeXtTable=\expandafter{%
17735 \the\markdownConTeXtTable\eTABLE}%
17736 \the\markdownConTeXtTableFloat
17737 \endgroup
17738 \expandafter\gobbleoneargument
17739 \fi\markdownConTeXtRenderTableRow}
17740 \def\markdownConTeXtReadAlignments#1{%
17741 \advance\markdownConTeXtColumnCounter by 1\relax
17742 \if#1d%
17743 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
17744 \fi\if#1l%
17745 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
17746 \fi\if#1c%
17747 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
17748 \fi\if#1r%
17749 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
17750 \fi
17751 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
17752 \else
17753 \expandafter\gobbleoneargument
17754 \fi\markdownConTeXtReadAlignments}
17755 \def\markdownConTeXtRenderTableCell#1{%

```

```

17756 \advance\markdownConTeXtColumnCounter by 1\relax
17757 \markdownConTeXtTable=\expandafter{%
17758 \the\markdownConTeXtTable\bTD#1\eTD}%
17759 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
17760 \else
17761 \expandafter\gobbleoneargument
17762 \fi\markdownConTeXtRenderTableCell}

```

### 3.4.3.3 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```

17763 \ExplSyntaxOn
17764 \cs_gset:Npn
17765 \markdownRendererInputRawInlinePrototype#1#2
17766 {
17767 \str_case:nnF
17768 { #2 }
17769 {
17770 { latex }
17771 {
17772 \@@_plain_tex_default_input_raw_inline:nn
17773 { #1 }
17774 { context }
17775 }
17776 }
17777 {
17778 \@@_plain_tex_default_input_raw_inline:nn
17779 { #1 }
17780 { #2 }
17781 }
17782 }
17783 \cs_gset:Npn
17784 \markdownRendererInputRawBlockPrototype#1#2
17785 {
17786 \str_case:nnF
17787 { #2 }
17788 {
17789 { context }
17790 {
17791 \@@_plain_tex_default_input_raw_block:nn
17792 { #1 }
17793 { tex }
17794 }
17795 }
17796 {
17797 \@@_plain_tex_default_input_raw_block:nn

```



```

17798 { #1 }
17799 { #2 }
17800 }
17801 }
17802 \cs_gset_eq:NN
17803 \markdownRendererInputRawBlockPrototype
17804 \markdownRendererInputRawInlinePrototype
17805 \fi % Closes ``\markdownIfOption{plain}{\iffalse}{\iftrue}`
17806 \ExplSyntaxOff
17807 \stopmodule
17808 \protect

```

At the end of the ConT<sub>E</sub>Xt module, we load the [witiko/markdown/defaults](#) ConT<sub>E</sub>Xt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

17809 \ExplSyntaxOn
17810 \str_if_eq:VVT
17811 \c_@@_top_layer_tl
17812 \c_@@_option_layer_context_tl
17813 {
17814 \use:c
17815 { ExplSyntaxOff }
17816 \@@_if_option:nF
17817 { noDefaults }
17818 {
17819 \@@_if_option:nTF
17820 { experimental }
17821 {
17822 \@@_setup:n
17823 { theme = witiko/markdown/defaults@experimental }
17824 }
17825 {
17826 \@@_setup:n
17827 { theme = witiko/markdown/defaults }
17828 }
17829 }
17830 \use:c
17831 { ExplSyntaxOn }
17832 }
17833 \ExplSyntaxOff
17834 \stopmodule
17835 \protect

```

## References

- [1] Hans Hagen. *ConT<sub>E</sub>Xt Lua Documents*. July 8, 2023. URL: <https://www.pragma-ade.nl/general/manuals/cld-mkiv.pdf> (visited on 09/22/2025).
- [2] LuaT<sub>E</sub>X development team. *LuaT<sub>E</sub>X reference manual*. Version 1.21. Feb. 1, 2025. URL: <http://mirrors.ctan.org/systems/doc/luatex/luatex.pdf> (visited on 05/12/2025).
- [3] L<sup>A</sup>T<sub>E</sub>X Project. *l3kernel. L<sup>A</sup>T<sub>E</sub>X3 programming conventions*. Dec. 25, 2024. URL: <https://ctan.org/pkg/l3kernel> (visited on 01/06/2025).
- [4] Frank Mittelbach, Ulrike Fischer, and L<sup>A</sup>T<sub>E</sub>X Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).
- [5] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [6] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [7] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [8] 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).
- [9] 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.
- [10] Frank Mittelbach. *The doc and shortverb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [11] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [12] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [13] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).

- [14] Vít Starý Novotný. *Routing YAML metadata to expl3 key-values*. Markdown Enhancement Proposal. Oct. 14, 2024. URL: <https://github.com/witiko/markdown/discussions/517> (visited on 01/06/2025).
- [15] Frank Mittelbach. *L<sup>A</sup>T<sub>E</sub>X's hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [16] 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).
- [17] Unicode Consortium. *The Unicode Standard. Version 16.0 – Core Specification*. Sept. 10, 2024. URL: <https://www.unicode.org/versions/Unicode16.0.0/UnicodeStandard-16.0.pdf> (visited on 05/07/2025).
- [18] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [19] 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).
- [20] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [21] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

<code>autoIdentifiers</code>	22, 35, 89, 105
<code>blankBeforeBlockquote</code>	23
<code>blankBeforeCodeFence</code>	23
<code>blankBeforeDivFence</code>	23
<code>blankBeforeHeading</code>	24
<code>blankBeforeHtmlBlock</code>	24
<code>blankBeforeList</code>	24
<code>bracketedSpans</code>	25, 91, 493
<code>breakableBlockquotes</code>	25
<code>cacheDir</code>	4, 17, 20, 61, 62, 163, 194, 404, 426, 445
<code>citationNbsps</code>	25
<code>citations</code>	26, 94, 95
<code>codeSpans</code>	26
<code>contentBlocks</code>	21, 27, 37

contentBlocksLanguageMap	21
contentLevel	27
debugExtensions	9, 21, 28, 347
debugExtensionsFileName	21, 28
defaultOptions	10, 54, 401, 403
definitionLists	28, 99
depth_first_search	171
\DocumentMetadata	464
eagerCache	17, 401
ensureJekyllData	29
entities.char_entity	248
entities.dec_entity	247
entities.hex_entity	248
entities.hex_entity_with_x_char	248
escape_minimal	252
escape_programmatic_text	252
escape_typographic_text	252
expandtabs	308
expectJekyllData	29, 29
experimental	5, 18, 36, 464
experimentalOptions	10, 401, 403
extensions	30, 170, 352
extensions.bracketed_spans	352
extensions.citations	353
extensions.content_blocks	358
extensions.definition_lists	361
extensions.fancy_lists	363
extensions.fenced_code	369
extensions.fenced_divs	374
extensions.header_attributes	378
extensions.inline_code_attributes	380
extensions.jekyll_data	397
extensions.line_blocks	380
extensions.link_attributes	382
extensions.mark	381
extensions.notes	384
extensions.pipe_table	386
extensions.raw_inline	391
extensions.strike_through	391
extensions.subscripts	392

<code>extensions.superscripts</code>	393
<code>extensions.tex_math</code>	393
<code>fancyLists</code>	31, 115–121, 464
<code>fencedCode</code>	32, 42, 95, 103, 122, 415, 418
<code>fencedCodeAttributes</code>	33, 89, 103, 418
<code>fencedDiv</code>	104
<code>fencedDivs</code>	33, 44
<code>finalizeCache</code>	18, 22, 34, 34, 61, 62, 162, 401, 402
<code>frozenCache</code>	22, 34, 62, 79, 162, 415, 425
<code>frozenCacheCounter</code>	34, 402, 453, 454
<code>frozenCacheFileName</code>	22, 34, 61, 402
<code>\g_markdown_diagrams_infostrings_prop</code>	420
<code>gfmAutoIdentifiers</code>	22, 34, 89, 105
<code>hashEnumerators</code>	35
<code>headerAttributes</code>	35, 44, 89, 105
<code>HTML</code>	36
<code>html</code>	36, 107, 108, 477
<code>htmlOverLinks</code>	18, 36
<code>hybrid</code>	36, 37, 42, 48, 50, 65, 79, 123, 163, 253, 309, 453
<code>inlineCodeAttributes</code>	38, 89, 96
<code>inlineNotes</code>	38
<code>\input</code>	58, 403
<code>\inputmarkdown</code>	165, 167, 168, 495
<code>inputTempFileName</code>	63, 65, 447, 448, 450, 451
<code>\inputyaml</code>	165, 167, 495
<code>iterlines</code>	308
<code>jeekyllData</code>	3, 29, 30, 39, 132–135, 137
<code>\l_file_search_path_seq</code>	452
<code>languages_json</code>	358, 358
<code>lineBlocks</code>	40, 111
<code>linkAttributes</code>	39, 89, 109, 113, 328, 490
<code>mark</code>	40, 113, 488
<code>\markdown</code>	158, 159, 457
<code>markdown</code>	157, 157, 158, 455, 456
<code>markdown*</code>	157, 157, 162, 455
<code>\markdownBegin</code>	56, 56, 57, 58, 155, 157, 158, 166
<code>\markdownCleanup</code>	446

<code>\markdownConvert</code>	446
<code>\markdownEnd</code>	56, 56, 57, 58, 155, 157–159, 166
<code>\markdownError</code>	155, 155
<code>\markdownEscape</code>	56, 59, 454
<code>\markdownIfOption</code>	60
<code>\markdownIfSnippetExists</code>	84
<code>\markdownInfo</code>	155, 155
<code>\markdownInput</code>	56, 58, 157, 159, 160, 162, 167, 452, 455
<code>\markdownInputFilename</code>	445
<code>\markdownInputFileStream</code>	446
<code>\markdownInputPlainTeX</code>	455
<code>\markdownLoadPlainTeXTheme</code>	163, 169, 414
<code>\markdownLuaExecute</code>	449, 452
<code>\markdownLuaOptions</code>	442, 446
<code>\markdownMakeOther</code>	155, 494
<code>\markdownOptionExperimental</code>	65
<code>\markdownOptionFinalizeCache</code>	61
<code>\markdownOptionFrozenCache</code>	61
<code>\markdownOptionHybrid</code>	65
<code>\markdownOptionInputTempFileName</code>	62
<code>\markdownOptionNoDefaults</code>	64
<code>\markdownOptionOutputDir</code>	62, 63, 66, 67
<code>\markdownOptionPlain</code>	63
<code>\markdownOptionStripPercentSigns</code>	64
<code>\markdownOutputFileStream</code>	446
<code>\markdownPrepare</code>	445
<code>\markdownPrepareInputFilename</code>	445
<code>\markdownPrepareLuaOptions</code>	442
<code>\markdownReadAndConvert</code>	155, 447, 455–457, 495
<code>\markdownReadAndConvertProcessLine</code>	448, 449
<code>\markdownReadAndConvertStripPercentSigns</code>	447
<code>\markdownReadAndConvertTab</code>	446
<code>\markdownRendererAttributeClassName</code>	89
<code>\markdownRendererAttributeIdentifier</code>	89
<code>\markdownRendererAttributeKeyValue</code>	89
<code>\markdownRendererBlockQuoteBegin</code>	90
<code>\markdownRendererBlockQuoteEnd</code>	91
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	91
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	91
<code>\markdownRendererCite</code>	94, 95
<code>\markdownRendererCodeSpan</code>	96
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	96

<code>\markdownRendererCodeSpanAttributeContextEnd</code>	96
<code>\markdownRendererContentBlock</code>	97, 98
<code>\markdownRendererContentBlockCode</code>	98
<code>\markdownRendererContentBlockOnlineImage</code>	98
<code>\markdownRendererDisplayMath</code>	129
<code>\markdownRendererDlBegin</code>	99
<code>\markdownRendererDlBeginTight</code>	99
<code>\markdownRendererDlDefinitionBegin</code>	100
<code>\markdownRendererDlDefinitionEnd</code>	101
<code>\markdownRendererDlEnd</code>	101
<code>\markdownRendererDlEndTight</code>	101
<code>\markdownRendererDlItem</code>	100
<code>\markdownRendererDlItemEnd</code>	100
<code>\markdownRendererDocumentBegin</code>	114
<code>\markdownRendererDocumentEnd</code>	114
<code>\markdownRendererEllipsis</code>	44, 102
<code>\markdownRendererEmphasis</code>	102, 142
<code>\markdownRendererError</code>	131
<code>\markdownRendererFancyOlBegin</code>	116, 117
<code>\markdownRendererFancyOlBeginTight</code>	117
<code>\markdownRendererFancyOlEnd</code>	120
<code>\markdownRendererFancyOlEndTight</code>	121
<code>\markdownRendererFancyOlItem</code>	118
<code>\markdownRendererFancyOlItemEnd</code>	119
<code>\markdownRendererFancyOlItemWithNumber</code>	119
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	103
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	103
<code>\markdownRendererFencedDivAttributeContextBegin</code>	104
<code>\markdownRendererFencedDivAttributeContextEnd</code>	104
<code>\markdownRendererHalfTickedBox</code>	130
<code>\markdownRendererHardLineBreak</code>	112
<code>\markdownRendererHeaderAttributeContextBegin</code>	105
<code>\markdownRendererHeaderAttributeContextEnd</code>	105
<code>\markdownRendererHeadingFive</code>	107
<code>\markdownRendererHeadingFour</code>	106
<code>\markdownRendererHeadingOne</code>	105
<code>\markdownRendererHeadingSix</code>	107
<code>\markdownRendererHeadingThree</code>	106
<code>\markdownRendererHeadingTwo</code>	106
<code>\markdownRendererImage</code>	109
<code>\markdownRendererImageAttributeContextBegin</code>	109
<code>\markdownRendererImageAttributeContextEnd</code>	109

<code>\markdownRendererInlineHtmlComment</code>	107
<code>\markdownRendererInlineHtmlTag</code>	108
<code>\markdownRendererInlineMath</code>	129
<code>\markdownRendererInputBlockHtmlElement</code>	108
<code>\markdownRendererInputFencedCode</code>	95
<code>\markdownRendererInputRawBlock</code>	122
<code>\markdownRendererInputRawInline</code>	121
<code>\markdownRendererInputVerbatim</code>	95
<code>\markdownRendererInterblockSeparator</code>	110
<code>\markdownRendererJekyllDataBegin</code>	132
<code>\markdownRendererJekyllDataBoolean</code>	134
<code>\markdownRendererJekyllDataEmpty</code>	137
<code>\markdownRendererJekyllDataEnd</code>	132
<code>\markdownRendererJekyllDataMappingBegin</code>	133
<code>\markdownRendererJekyllDataMappingEnd</code>	133
<code>\markdownRendererJekyllDataNumber</code>	135
<code>\markdownRendererJekyllDataProgrammaticString</code>	135, 135, 136
<code>\markdownRendererJekyllDataSequenceBegin</code>	133
<code>\markdownRendererJekyllDataSequenceEnd</code>	134
<code>\markdownRendererJekyllDataString</code>	136, 140
<code>\markdownRendererJekyllDataStringPrototype</code>	150
<code>\markdownRendererJekyllDataTypographicString</code>	135, 135, 136, 397
<code>\markdownRendererLineBlockBegin</code>	111
<code>\markdownRendererLineBlockEnd</code>	111
<code>\markdownRendererLink</code>	112, 142
<code>\markdownRendererLinkAttributeContextBegin</code>	113
<code>\markdownRendererLinkAttributeContextEnd</code>	113
<code>\markdownRendererMark</code>	113
<code>\markdownRendererNbsp</code>	115
<code>\markdownRendererNote</code>	115
<code>\markdownRendererOlBegin</code>	115
<code>\markdownRendererOlBeginTight</code>	116
<code>\markdownRendererOlEnd</code>	120
<code>\markdownRendererOlEndTight</code>	120
<code>\markdownRendererOlItem</code>	44, 117
<code>\markdownRendererOlItemEnd</code>	118
<code>\markdownRendererOlItemWithNumber</code>	44, 118
<code>\markdownRendererParagraphSeparator</code>	110
<code>\markdownRendererReplacementCharacter</code>	123
<code>\markdownRendererSectionBegin</code>	122
<code>\markdownRendererSectionEnd</code>	122
<code>\markdownRendererSoftLineBreak</code>	111



<code>\markdownRendererStrikeThrough</code>	126
<code>\markdownRendererStrongEmphasis</code>	103
<code>\markdownRendererSubscript</code>	127
<code>\markdownRendererSuperscript</code>	127
<code>\markdownRendererTable</code>	128
<code>\markdownRendererTableAttributeContextBegin</code>	127
<code>\markdownRendererTableAttributeContextEnd</code>	127
<code>\markdownRendererTextCite</code>	95
<code>\markdownRendererThematicBreak</code>	129
<code>\markdownRendererTickedBox</code>	130
<code>\markdownRendererULBegin</code>	92
<code>\markdownRendererULBeginTight</code>	92
<code>\markdownRendererULEnd</code>	93
<code>\markdownRendererULEndTight</code>	94
<code>\markdownRendererULItem</code>	93
<code>\markdownRendererULItemEnd</code>	93
<code>\markdownRendererUntickedBox</code>	130
<code>\markdownRendererWarning</code>	131
<code>\markdownSetup</code>	60, 60, 65, 161, 162, 168, 457, 463
<code>\markdownSetupSnippet</code>	82, 82
<code>\markdownThemeVersion</code>	72, 73
<code>\markdownWarning</code>	155, 155
<code>\markinline</code>	56, 57, 58, 157, 159, 450, 455
<code>\markinlinePlainTeX</code>	455
<code>\mmdcCommand</code>	422
 new	7, 19, 401, 403
notes	40, 115
 parsers	269, 307
<code>parsers.commented_line</code>	289
<code>parsers.unicode</code>	272
<code>pipeTables</code>	7, 41, 47, 128
<code>preserveTabs</code>	42, 45, 308
 rawAttribute	37, 42, 42, 121, 122
<code>read_decompositions</code>	173
reader	8, 31, 170, 269, 306, 352
<code>reader-&gt;add_special_character</code>	8, 9, 31, 346
<code>reader-&gt;auto_link_email</code>	336
<code>reader-&gt;auto_link_url</code>	336
<code>reader-&gt;create_parser</code>	308
<code>reader-&gt;finalize_grammar</code>	342, 408

reader->initialize_named_group	347
reader->insert_pattern	8, 9, 31, 342, 348
reader->lookup_note_reference	320
reader->lookup_reference	320
reader->normalize_tag	307
reader->options	307
reader->parser_functions	308
reader->parser_functions.name	308
reader->parsers	307, 307
reader->register_link	320
reader->update_rule	342, 345, 348
reader->writer	307
reader.new	307, 307, 408
relativeReferences	36, 43
serialize_byte_parser	171
serialize_byte_range_parser	172
serialize_replacement	172
\setupmarkdown	168, 168
\setupyaml	168
shiftHeadings	7, 43
singletonCache	19
slice	7, 43, 249, 262, 263
smartEllipses	44, 102, 163
\startmarkdown	165, 166, 495
startNumber	44, 117–119
\startyaml	165, 166, 495
\stopmarkdown	165, 166, 495
\stopyaml	165, 166, 495
strikeThrough	45, 126, 489
stripIndent	45, 308
stripPercentSigns	447
subscripts	46, 127
superscripts	46, 127
syntax	344, 348
tableAttributes	46, 127, 485
tableCaptions	7, 46, 47, 127
taskLists	47, 130, 476
texComments	48, 309
texMathDollars	37, 48, 129
texMathDoubleBackslash	37, 49, 129

<code>texMathSingleBackslash</code>	37, 49, 129
<code>tightLists</code>	49, 92, 94, 99, 101, 116, 117, 120, 121, 464
<code>underscores</code>	50
<code>unicode_data.casefold_mapping</code>	183, 194
<code>unicode_data.categories</code>	185, 272
<code>unicode_data.ccc</code>	187, 195
<code>unicode_data.composition_mapping</code>	179, 199
<code>unicode_data.decomposition_mapping</code>	173, 175, 197
<code>unicodeNormalization</code>	19, 20
<code>unicodeNormalizationForm</code>	19, 20
<code>util.cache</code>	189, 190
<code>util.cache_verbatim</code>	190
<code>util.canonically_order</code>	195
<code>util.casefold</code>	194
<code>util.compose</code>	199
<code>util.decompose</code>	197
<code>util.encode_json_string</code>	190
<code>util.err</code>	189
<code>util.escaper</code>	192
<code>util.expand_tabs_in_line</code>	190
<code>util.find_file</code>	202
<code>util.find_files</code>	202
<code>util.flatten</code>	191
<code>util.intersperse</code>	192
<code>util.map</code>	192
<code>util.normalize</code>	201
<code>util.pathname</code>	193
<code>util.rope_last</code>	191
<code>util.rope_to_string</code>	191
<code>util.salt</code>	193
<code>util.table_copy</code>	190
<code>util.walk</code>	190, 191
<code>util.warning</code>	194
<code>walkable_syntax</code>	8, 21, 28, 342, 345–348
<code>writer</code>	170, 170, 248, 249, 352
<code>writer-&gt;active_attributes</code>	261, 261, 262, 263
<code>writer-&gt;attribute_type_levels</code>	261
<code>writer-&gt;attributes</code>	259
<code>writer-&gt;block_html_element</code>	257
<code>writer-&gt;blockquote</code>	258

writer->bulletitem	255
writer->bulletlist	255
writer->citations	353
writer->code	253
writer->contentblock	358
writer->defer_call	269, 269
writer->definitionlist	361
writer->display_math	393
writer->div_begin	374
writer->div_end	374
writer->document	258
writer->ellipsis	251
writer->emphasis	257
writer->error	253
writer->escape	253
writer->escaped_chars	252, 252
writer->escaped_minimal_strings	251, 252
writer->escaped_strings	252
writer->escaped_uri_chars	251, 252
writer->fancyitem	364
writer->fancylist	363
writer->fencedCode	369
writer->flatten_inlines	249, 249
writer->get_state	269
writer->hard_line_break	251
writer->heading	267
writer->identifier	253
writer->image	254
writer->infostring	253
writer->inline_html_comment	256
writer->inline_html_tag	257
writer->inline_math	394
writer->interblocksep	250
writer->is_writing	249, 249
writer->jekyllData	397
writer->lineblock	381
writer->link	254
writer->mark	381
writer->math	253
writer->nbsp	250
writer->note	384
writer->options	249

writer->ordereditem	256
writer->orderedlist	256
writer->paragraph	250
writer->paragraphsep	250
writer->plain	250
writer->pop_attributes	261, 262, 263
writer->push_attributes	261, 262, 263
writer->rawBlock	370
writer->rawInline	391
writer->set_state	269
writer->slice_begin	249
writer->slice_end	249
writer->soft_line_break	251
writer->space	250
writer->span	352
writer->strike_through	391
writer->string	253
writer->strong	257
writer->subscript	392
writer->superscript	393
writer->table	388
writer->thematic_break	251
writer->checkbox	257
writer->undosep	251, 351
writer->uri	253
writer->verbatim	258
writer->warning	194, 253
writer.new	249, 249, 408
\yaml	159
yaml	157, 158, 159, 455
\yamlBegin	56, 57, 155, 158, 166
\yamlEnd	56, 57, 155, 158, 159, 166
\yamlInput	56, 58, 157, 160, 167, 455
\yamlSetup	60