

A Markdown Interpreter for T_EX

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

Version 3.16.0-0-g4d0fb773
2026-06-17

Contents

1	Introduction	1	3	Implementation	182
1.1	Requirements	2	3.1	Lua Implementation . . .	182
1.2	Feedback	6	3.2	Plain T _E X Implementation	429
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation . .	476
2	Interfaces	7	3.4	ConT _E Xt Implementation	523
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	57	References		534
2.3	L ^A T _E X Interface	168	Index		536
2.4	ConT _E Xt Interface	177			

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 . .	53
3	A sequence diagram of typesetting a document using the Lua CLI	54
4	An example directed graph	83
5	An example mindmap	84
6	An example UML sequence diagram	85
7	The banner of the Markdown package	86
8	A pushdown automaton that recognizes T _E X comments	303

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

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

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

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

Kpathsea A package that implements the loading of third-party Lua libraries and looking up files in the 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 3.3.5) 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 UTF8 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`

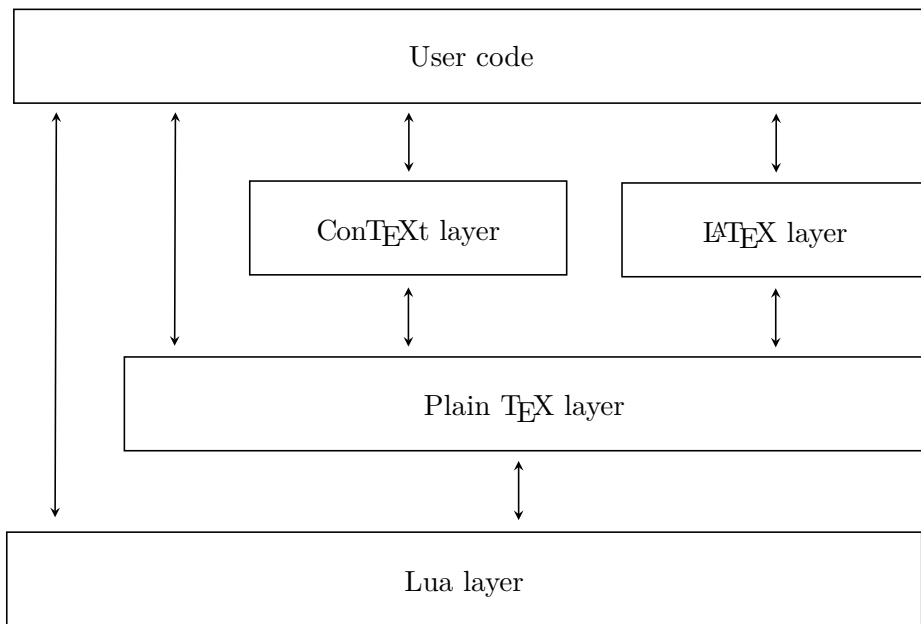


Figure 1: A block diagram of the Markdown package

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 T_EX output using the default options and prints the T_EX output:

```

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

```

2.1.2 User-Defined Syntax Extensions

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

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

```

57 local walkable_syntax = {

```



```

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

```

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

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

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

2.1.3 Options

The Lua interface recognizes the following options. When unspecified, the value of a key is taken from the `defaultOptions` 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_protected: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_protected: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_protected: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_protected: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_protected: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_protected: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_protected: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_protected: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_protected: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     }
395 \regex_const:Nn
396     \c_@@_option_regex
397     { ^markdownOption }
398 \prg_new_protected_conditional:Nnn
399     \@@_csname_to_option_str:nN
400     { T }
401     {
402         \tl_if_regex_match:nNTF
403             { #1 }
404             \c_@@_option_regex
405             {
406                 \tl_set:Nn

```



```

407         \l_tmpa_tl
408         { #1 }
409     \tl_regex_replace_once:NNn
410     \l_tmpa_tl
411     \c_@@_option_regex
412     { }
413     \tl_set:Nn
414     \l_tmpb_tl
415     {
416         \str_lowercase:f
417         { \l_tmpa_tl }
418     }
419     \str_set:Nx
420     #2
421     {
422         \tl_head:f { \l_tmpb_tl }
423         \tl_tail:V \l_tmpa_tl
424     }
425     \prg_return_true:
426 }
427 { \prg_return_false: }
428 }

```

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.

```

429 \cs_new_protected:Nn \@@_with_various_cases:nn
430 {
431     \seq_clear:N
432     \l_tmpa_seq
433     \seq_map_inline:Nn
434     \g_@@_cases_seq
435     {
436         \tl_set:Nn
437         \l_tmpa_tl
438         { #1 }
439         \use:c { ##1 }
440         \l_tmpa_tl
441         \seq_put_right:NV
442         \l_tmpa_seq
443         \l_tmpa_tl
444     }
445     \seq_map_inline:Nn
446     \l_tmpa_seq
447     { #2 }
448 }

```

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

```

449 \seq_new:N \g_@@_cases_seq
450 \cs_new_protected:Nn \@@_camel_case:N
451 {
452   \regex_replace_all:nnN
453     { _ ([a-z]) }
454     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
455     #1
456   \tl_set:Nx
457     #1
458     { #1 }
459 }
460 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
461 \cs_new_protected:Nn \@@_snake_case:N
462 {
463   \regex_replace_all:nnN
464     { ([a-z])([A-Z]) }
465     { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
466     #1
467   \tl_set:Nx
468     #1
469     { #1 }
470 }
471 \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.

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.

```
472 \@@_add_lua_option:nnn
473   { eagerCache }
474   { boolean }
475   { true }

476 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.

```
477 \@@_add_lua_option:nnn
478   { experimental }
479   { boolean }
480   { false }

481 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.

```
482 \@@_add_lua_option:nnn
483 { singletonCache }
484 { boolean }
485 { true }

486 defaultOptions.singletonCache = true

487 local singletonCache = {
488   convert = nil,
489   options = nil,
490 }
```

`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.

```
491 \@@_add_lua_option:nnn
492 { unicodeNormalization }
493 { boolean }
494 { true }

495 defaultOptions.unicodeNormalization = true
```

`unicodeNormalizationForm=nfc, nfd, nfkc, nfkd`

default: `nfc`

nfc When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form C (NFC) before conversion.

nfd When option `unicodeNormalization` has been enabled, markdown documents will be normalized using Unicode Normalization Form D (NFD) before conversion.

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

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

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.

```

496 \@@_add_lua_option:nnn
497   { unicodeNormalizationForm }
498   { string }
499   { nfc }

500 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.

```

501 \str_new:N
502   \g_@@_unquoted_jobname_str
503 \str_gset:NV
504   \g_@@_unquoted_jobname_str
505   \c_sys_jobname_str
506 \bool_new:N
507   \g_@@_jobname_quoted_bool
508 \regex_replace_all:nnNTF
509   { \A ("|') ( .* ) ("|') \Z }
510   { \2 }
511   \g_@@_unquoted_jobname_str
512   {
513     \bool_gset_true:N
514     \g_@@_jobname_quoted_bool
515   }
516   {
517     \bool_gset_false:N
518     \g_@@_jobname_quoted_bool

```

```

519     }
520     \@@_add_lua_option:nnn
521     { cacheDir }
522     { path }
523     {
524         \markdownOptionOutputDir
525         / _markdown_
526         \str_use:N
527         \g_@@_unquoted_jobname_str
528     }
529     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 2.2.5.14 for more information.

```

530     \@@_add_lua_option:nnn
531     { contentBlocksLanguageMap }
532     { path }
533     { markdown-languages.json }
534     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.

```

535     \@@_add_lua_option:nnn
536     { debugExtensionsFileName }
537     { path }
538     {
539         \markdownOptionOutputDir
540         /
541         \str_use:N
542         \g_@@_unquoted_jobname_str
543         .debug-extensions.json
544     }
545     defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

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

```
546 \@@_add_lua_option:nnn
547   { frozenCacheFileName }
548   { path }
549   { \markdownOptionCacheDir / frozenCache.tex }
550 defaultOptions.frozenCacheFileName = "frozenCache.tex"
```

2.1.6 Parser Options

`acronyms`= $\langle acronyms \rangle$

Acronyms, initialisms, and other all-caps sequences that will be recognized in markdown text and formatted accordingly.

```
551 \cs_generate_variant:Nn
552   \@@_add_lua_option:nnn
553   { nnV }
554 \@@_add_lua_option:nnV
555   { acronyms }
556   { clist }
557   \c_empty_clist
558 defaultOptions.acronyms = {}
```

`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.

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

See also the option [gfmAutoIdentifiers](#).

```
559 \@@_add_lua_option:nnn
560   { autoIdentifiers }
561   { boolean }
562   { false }

563 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.

```
564 \@@_add_lua_option:nnn
565   { blankBeforeBlockquote }
566   { boolean }
567   { false }

568 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.

```
569 \@@_add_lua_option:nnn
570   { blankBeforeCodeFence }
571   { boolean }
572   { false }

573 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.

```
574 \@@_add_lua_option:nnn
575   { blankBeforeDivFence }
576   { boolean }
577   { false }

578 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.

```
579 \@@_add_lua_option:nnn
580 { blankBeforeHeading }
581 { boolean }
582 { false }

583 defaultOptions.blankBeforeHeading = false
```

`blankBeforeHtmlBlock=true, false` default: false

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

```
584 \@@_add_lua_option:nnn
585 { blankBeforeHtmlBlock }
586 { boolean }
587 { false }

588 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.

```
589 \@@_add_lua_option:nnn
590 { blankBeforeList }
591 { boolean }
592 { false }

593 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.

```
594 \@@_add_lua_option:nnn
595   { bracketedSpans }
596   { boolean }
597   { false }

598 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.

```
599 \@@_add_lua_option:nnn
600   { breakableBlockquotes }
601   { boolean }
602   { true }

603 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.

```
604 \@@_add_lua_option:nnn
605   { citationNbsps }
606   { boolean }
607   { true }

608 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.

```
609 \@@_add_lua_option:nnn
610   { citations }
611   { boolean }
612   { false }
613 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.``

```
614 \@@_add_lua_option:nnn
615   { codeSpans }
616   { boolean }
617   { true }
618 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]:

```
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.

```
619 \@@_add_lua_option:nnn
620 { contentBlocks }
621 { boolean }
622 { false }

623 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
```

```
624 \@@_add_lua_option:nnn
625 { contentLevel }
626 { string }
627 { block }

628 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.

```
629 \@@_add_lua_option:nnn
630 { debugExtensions }
631 { boolean }
632 { false }

633 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.

```
634 \@@_add_lua_option:nnn
635 { definitionLists }
636 { boolean }
637 { false }

638 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: false

- 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.
- 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.

See also the plain $\text{T}_{\text{E}}\text{X}$ macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ environment `yaml` in Section 2.3.1, and the $\text{C}^{\text{o}}\text{nT}_{\text{E}}\text{Xt}$ macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jekyllData`, `expectJekyllData`, and `ensureJekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```
639 \@@_add_lua_option:nnn
640   { ensureJekyllData }
641   { boolean }
642   { false }

643 defaultOptions.ensureJekyllData = false
```

`expectJekyllData=true, false`

default: false

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

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

```

- is
- YAML
\end{markdown}
\end{document}

```

`false`

When the `jeekyllData` 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[jeekyllData]{markdown}
\begin{document}
\begin{markdown}
---
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- Markdown
\end{markdown}
\end{document}

```

See also the plain T_EX macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the L^AT_EX environment `yaml` in Section 2.3.1, and the ConT_EXt macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jeekyllData`, `expectJekyllData`, and `ensureJekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```

644 \@@_add_lua_option:nnn
645   { expectJekyllData }
646   { boolean }
647   { false }

648 defaultOptions.expectJekyllData = false

```

`extensions`= $\langle filenames \rangle$

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:

```
649 metadata.user_extension_api_version = 2
650 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.

```
651 \@@_add_lua_option:nnV
652   { extensions }
653   { clist }
654   \c_empty_clist
655 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: false

`true` Enable the Pandoc fancy list syntax extension¹³:

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

`false` Disable the Pandoc fancy list syntax extension.

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

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

```
</code>
</pre>
...
```

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

```
661 \@@_add_lua_option:nnn
662 { fencedCode }
663 { boolean }
664 { true }

665 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.

```
666 \@@_add_lua_option:nnn
667 { fencedCodeAttributes }
668 { boolean }
669 { false }

670 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.

<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).

```

671 \@@_add_lua_option:nnn
672 { fencedDivs }
673 { boolean }
674 { false }

675 defaultOptions.fencedDivs = false

```

**frozenCache**=true, false

default: false

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

The frozen cache makes it possible to later typeset a plain T<sub>E</sub>X document that contains markdown documents without invoking Lua using the **frozenCache** plain T<sub>E</sub>X option. 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.

```

676 \@@_add_lua_option:nnn
677 { finalizeCache }
678 { boolean }
679 { false }

680 defaultOptions.finalizeCache = false

```

**frozenCacheCounter**=*<number>*

default: 0

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

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

```

681 \@@_add_lua_option:nnn
682 { frozenCacheCounter }
683 { counter }
684 { 0 }

685 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.

See also the option `autoIdentifiers`.

```
686 \@@_add_lua_option:nnn
687 { gfmAutoIdentifiers }
688 { boolean }
689 { false }
690 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.

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

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

`false`      Disable the assignment of HTML attributes to headings.

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

700 defaultOptions.headerAttributes = false
```

`html=true, false`      default: `true`

`true`      Enable the recognition of HTML entities, inline HTML tags, and block HTML elements, comments, declarations, processing instructions, and CDATA sections in the input.

HTML entities will be replaced with the corresponding Unicode code-points and other HTML content will be ignored by default.

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

```
701 \@@_add_lua_option:nnn
702 { html }
703 { boolean }
704 { true }

705 defaultOptions.html = true
```

`htmlOutput=basic, commonmark`      default: `basic`

`basic`

: When the `html` option is enabled, produce basic coarse-grained renderers for all HTML, see Section 2.2.5.3.

`commonmark`

: When the `html` option is enabled, produce fine-grained renderers for each top-level non-terminal element of CommonMark's grammar for HTML blocks<sup>17</sup> and raw HTML<sup>18</sup>, see sections 2.2.5.12 and 2.2.5.13, respectively.

```
706 \@@_add_lua_option:nnn
707 { htmlOutput }
708 { string }
709 { basic }

710 defaultOptions.htmlOutput = "basic"
```

---

<sup>17</sup>See <https://spec.commonmark.org/0.31.2/#html-blocks>.

<sup>18</sup>See <https://spec.commonmark.org/0.31.2/#raw-html>.

`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.

```
711 \@@_add_experimental_lua_option:n
712 { htmlOverLinks }

713 defaultOptions.htmlOverLinks = false
714 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 TeX characters outside verbatim environments, so that they are not interpreted by TeX. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

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

Consider one of the following better alternatives for mixing TeX and markdown:

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

Here is a mathematical formula:

```
/math-formula.tex
```

- With the `rawAttribute` option, authors can denote raw text spans and code blocks that will be interpreted as  $\TeX$  code:

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

Here is a mathematical formula:

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

- With options `texMathDollars`, `texMathSingleBackslash`, and `texMathDoubleBackslash`, authors can freely type  $\TeX$  commands between dollar signs or backslash-escaped brackets:

```
$H_2 O$ is a liquid.
```

Here is a mathematical formula:

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

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

`inlineCodeAttributes=true, false` default: false

**true** Enable the Pandoc inline code span attribute extension<sup>19</sup>:

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

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

```
720 \@@_add_lua_option:nnn
721 { inlineCodeAttributes }
722 { boolean }
723 { false }
724 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

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

```
Here is an inline note.^[Inline 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.

```
725 \@@_add_lua_option:nnn
726 { inlineNotes }
727 { boolean }
728 { false }
729 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

**true** Enable the Pandoc YAML metadata block syntax extension<sup>21</sup> for entering metadata in YAML:

```

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

<sup>19</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-inline_code_attributes).

<sup>20</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>21</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).



```
abstract: |
 This is the abstract.

 It consists of two paragraphs.

```

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

See also the plain  $\text{\TeX}$  macros `\yamlBegin`, `\yamlEnd`, and `\yamlInput` in Section 2.2.1, the  $\text{\LaTeX}$  environment `yaml` in Section 2.3.1, and the  $\text{\ConTeXt}$  macros `\startyaml`, `\stopyaml`, and `\inputyaml` in Section 2.4.1 to see how the options `jeekyllData`, `expectJeekyllData`, and `ensureJeekyllData` can be used together to implement high-level interfaces for processing YAML documents.

```
730 \@@_add_lua_option:nnn
731 { jeekyllData }
732 { boolean }
733 { false }

734 defaultOptions.jekyllData = false
```

`linkAttributes=true, false`

default: `false`

`true` Enable the Pandoc link and image attribute syntax extension<sup>22</sup>:

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

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

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

```
735 \@@_add_lua_option:nnn
736 { linkAttributes }
737 { boolean }
738 { false }

739 defaultOptions.linkAttributes = false
```

<sup>22</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>23</sup>:

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

`false` Disable the Pandoc line block syntax extension.

```
740 \@@_add_lua_option:nnn
741 { lineBlocks }
742 { boolean }
743 { false }

744 defaultOptions.lineBlocks = false
```

`mark=true, false`

default: false

`true` Enable the Pandoc mark syntax extension<sup>24</sup>:

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

`false` Disable the Pandoc mark syntax extension.

```
745 \@@_add_lua_option:nnn
746 { mark }
747 { boolean }
748 { false }

749 defaultOptions.mark = false
```

`notes=true, false`

default: false

`true` Enable the Pandoc note syntax extension<sup>25</sup>:

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

[^1]: Here is the note.

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

<sup>23</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

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

<sup>25</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.

```
750 \@@_add_lua_option:nnn
751 { notes }
752 { boolean }
753 { false }
754 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.

```
755 \@@_add_lua_option:nnn
756 { pipeTables }
757 { boolean }
758 { false }
759 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.

```
760 \@@_add_lua_option:nnn
761 { preserveTabs }
762 { boolean }
763 { true }

764 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: `false`

- `true` Enable the Pandoc raw attribute syntax extension<sup>26</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 T_EX.

- `false` Disable the Pandoc raw attribute syntax extension.

```
765 \@@_add_lua_option:nnn
766   { rawAttribute }
767   { boolean }
768   { false }

769 defaultOptions.rawAttribute = false
```

²⁶See https://pandoc.org/MANUAL.html#extension-raw_attribute.

`relativeReferences=true, false`

default: false

`true` Enable relative references²⁷ 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.

```
770 \@@_add_lua_option:nnn
771 { relativeReferences }
772 { boolean }
773 { false }

774 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.

```
775 \@@_add_lua_option:nnn
776 { shiftHeadings }
777 { number }
778 { 0 }

779 defaultOptions.shiftHeadings = 0
```

`slice=<the beginning and the end of a slice>`

default: ^ \$

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

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

²⁷See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

- $\$ \langle identifier \rangle$ 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.

```

780 \@@_add_lua_option:nnn
781   { slice }
782   { slice }
783   { ^~$ }
784 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.

```

785 \@@_add_lua_option:nnn
786   { smartEllipses }
787   { boolean }
788   { false }
789 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 `\markdownRendererOliItemWithNumber` \TeX macro.
- false** Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRenderOliItem` \TeX macro.

```

790 \@@_add_lua_option:nnn
791   { startNumber }
792   { boolean }
793   { true }
794 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

true Enable the Pandoc strike-through syntax extension²⁸:

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

false Disable the Pandoc strike-through syntax extension.

```
795 \@@_add_lua_option:nnn
796 { strikeThrough }
797 { boolean }
798 { false }
799 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.

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

²⁸See <https://pandoc.org/MANUAL.html#extension-strikeout>.

`subscripts=true, false` default: false

`true` Enable the Pandoc subscript syntax extension²⁹:

H~2~0 is a liquid.

`false` Disable the Pandoc subscript syntax extension.

```
805 \@@_add_lua_option:nnn
806 { subscripts }
807 { boolean }
808 { false }
809 defaultOptions.subscripts = false
```

`superscripts=true, false` default: false

`true` Enable the Pandoc superscript syntax extension³⁰:

2¹⁰ is 1024.

`false` Disable the Pandoc superscript syntax extension.

```
810 \@@_add_lua_option:nnn
811 { superscripts }
812 { boolean }
813 { false }
814 defaultOptions.superscripts = false
```

`tableAttributes=true, false` default: false

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

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

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

²⁹See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

³⁰See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

false Disable the assignment of HTML attributes to table captions.

```
815 \@@_add_lua_option:nnn
816 { tableAttributes }
817 { boolean }
818 { false }

819 defaultOptions.tableAttributes = false
```

tableCaptions=true, false

default: false

true Enable the Pandoc table caption syntax extension³¹ for pipe tables (see the **pipeTables** option):

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.

```
820 \@@_add_lua_option:nnn
821 { tableCaptions }
822 { boolean }
823 { false }

824 defaultOptions.tableCaptions = false
```

taskLists=true, false

default: false

true Enable the Pandoc task list syntax extension³²:

- [] an unticked task list item
- [X] a ticked task list item

Our implementation also supports half-ticked task list items:

- [/] a half-checked task list item
- [.] another half-checked task list item

false Disable the Pandoc task list syntax extension.

³¹See https://pandoc.org/MANUAL.html#extension-table_captions.

³²See https://pandoc.org/MANUAL.html#extension-task_lists.

```

825 \@@_add_lua_option:nnn
826   { taskLists }
827   { boolean }
828   { false }

829 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

true Strip T_EX-style comments.

```

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

```

Always enabled when `hybrid` is enabled.

false Do not strip T_EX-style comments.

```

830 \@@_add_lua_option:nnn
831   { texComments }
832   { boolean }
833   { false }

834 defaultOptions.texComments = false

```

`texMathDollars=true, false`

default: false

true Enable the Pandoc dollar math syntax extension³³:

```

inline math: $E=mc^2$

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

```

false Disable the Pandoc dollar math syntax extension.

```

835 \@@_add_lua_option:nnn
836   { texMathDollars }
837   { boolean }
838   { false }

839 defaultOptions.texMathDollars = false

```

³³See https://pandoc.org/MANUAL.html#extension-tex_math_dollars.

`texMathDoubleBackslash=true, false` default: false

true Enable the Pandoc double backslash math syntax extension³⁴:

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

false Disable the Pandoc double backslash math syntax extension.

```
840 \\@@_add_lua_option:nnn
841   { texMathDoubleBackslash }
842   { boolean }
843   { false }

844 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

true Enable the Pandoc single backslash math syntax extension³⁵:

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

false Disable the Pandoc single backslash math syntax extension.

```
845 \\@@_add_lua_option:nnn
846   { texMathSingleBackslash }
847   { boolean }
848   { false }

849 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:

³⁴See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.

³⁵See 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.

```

850 \@@_add_lua_option:nnn
851   { tightLists }
852   { boolean }
853   { true }

854 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.

```

855 \@@_add_lua_option:nnn
856   { underscores }
857   { boolean }
858   { true }
859 \ExplSyntaxOff

860 defaultOptions.underscores = true

```

2.1.7 Command-Line Interface

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

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

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

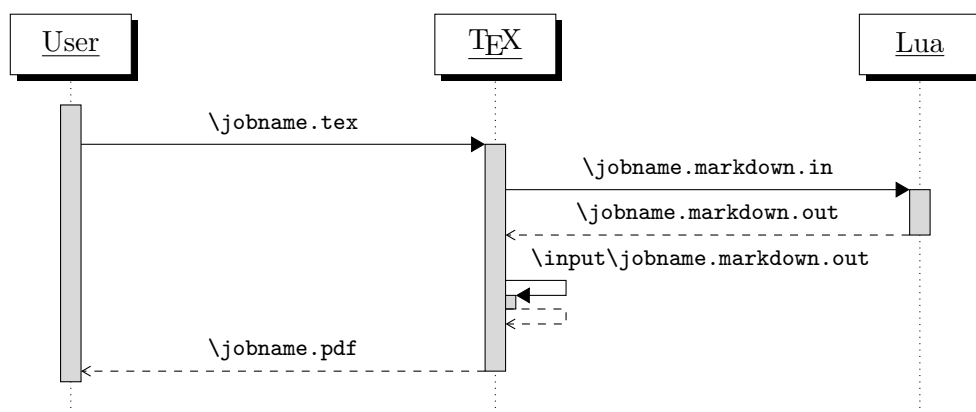


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T_EX interface

```

861 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
862 .SH NAME
863 markdown2tex \- convert .md files to .tex
864 .SH SYNOPSIS
865 local HELP_STRING = "Usage: " .. [[
866 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
867
868 .SH DESCRIPTION
869 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
870 Manual (https://ctan.org/pkg/markdown).
871

```

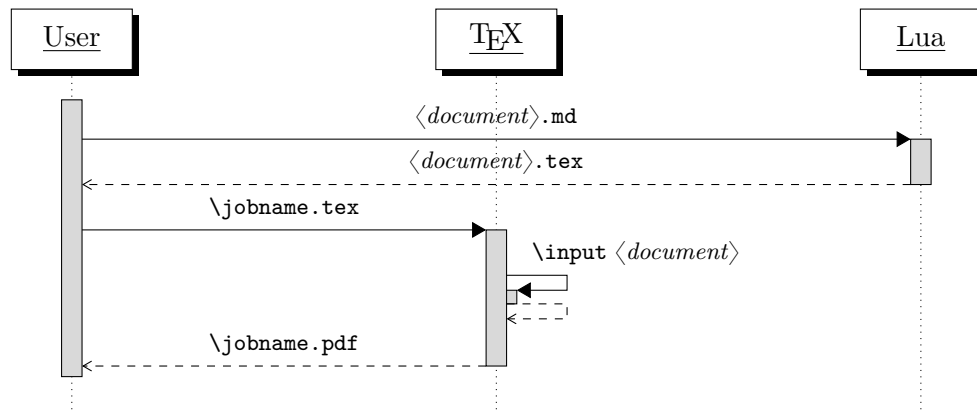


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

872 When `OUTPUT_FILE` is unspecified, the result of the conversion will be
 873 written to the standard output. When `INPUT_FILE` is also unspecified, the
 874 result of the conversion will be read from the standard input.

```

875
876 Report bugs to: witiko@mail.muni.cz
877 Markdown package home page: <https://github.com/witiko/markdown>]]
878
879 local VERSION_STRING = [[
880 markdown2tex (Markdown) ]] .. metadata.version .. [[
881
882 Copyright (C) ]] .. table.concat(metadata.copyright,
883                                   "\nCopyright (C) ") .. [[
884
885 License: ]] .. metadata.license
886
887 local function warn(s)
888   io.stderr:write("Warning: " .. s .. "\n")
889 end
890
891 local function error(s)
892   io.stderr:write("Error: " .. s .. "\n")
893   os.exit(1)
894 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.

```

895 local function camel_case(option_name)
  
```

```

896     local cased_option_name = option_name:gsub("_(%l)", function(match)
897         return match:sub(2, 2):upper()
898     end)
899     return cased_option_name
900 end
901
902 local function snake_case(option_name)
903     local cased_option_name = option_name:gsub("%l%u", function(match)
904         return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
905     end)
906     return cased_option_name
907 end
908
909 local cases = {camel_case, snake_case}
910 local various_case_options = {}
911 for option_name, _ in pairs(defaultOptions) do
912     for _, case in ipairs(cases) do
913         various_case_options[case(option_name)] = option_name
914     end
915 end
916
917 local process_options = true
918 local options = {}
919 local input_filename
920 local output_filename
921 for i = 1, #arg do
922     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.

```

923         if arg[i] == "--" then
924             process_options = false
925             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.

```

926         elseif arg[i]:match("=") then
927             local key, value = arg[i]:match("(.)=(.*)")
928             if defaultOptions[key] == nil and
929                 various_case_options[key] ~= nil then
930                 key = various_case_options[key]
931             end

```

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

```

932     local default_type = type(defaultOptions[key])
933     if default_type == "boolean" then
934         options[key] = (value == "true")
935     elseif default_type == "number" then
936         options[key] = tonumber(value)
937     elseif default_type == "table" then
938         options[key] = {}
939         for item in value:gmatch("[^,]+") do
940             table.insert(options[key], item)
941         end
942     else
943         if default_type ~= "string" then
944             if default_type == "nil" then
945                 warn('Option "' .. key .. '" not recognized.')
946             else
947                 warn('Option "' .. key .. '" type not recognized, ' ..
948                     'please file a report to the package maintainer.')
949             end
950             warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
951                 key .. '" as a string.')
952         end
953         options[key] = value
954     end
955     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.

```

956     elseif arg[i] == "--help" or arg[i] == "-h" then
957         print(HELP_STRING)
958         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.

```

959     elseif arg[i] == "--version" or arg[i] == "-v" then
960         print(VERSION_STRING)
961         os.exit()
962     end
963 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 \TeX document.

```

964     if input_filename == nil then
965         input_filename = arg[i]

```


The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the T_EX document that will result from the conversion.

```

966 elseif output_filename == nil then
967     output_filename = arg[i]
968 else
969     error('Unexpected argument: "' .. arg[i] .. '".')
970 end
971 ::continue::
972 end

```

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

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

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

```
\input hello
```

2.2 Plain T_EX Interface

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

```

973 \def\markdownLastModified{((LASTMODIFIED))}%
974 \def\markdownVersion{((VERSION))}%

```

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

```
\input markdown
```

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

2.2.1 Typesetting Markdown and YAML

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

2.2.1.1 Typesetting Markdown and yaml directly

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

```
975 \let\markdownBegin\relax
976 \let\markdownEnd\relax
```

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

There are several limitations to the macros you need to be aware of:

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T_EX [9, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```

977 \let\yamlBegin\relax
978 \def\yamlEnd{\markdownEnd\endgroup}

```

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

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

```

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

```

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

```

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

```

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

```

979 \let\markinline\relax

```

The following example plain T_EX code showcases the usage of the `\markinline` macro:

```

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

```

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

```

\input markdown
\markdownSetup{contentLevel=inline}
\markdownBegin
_Hello_ **world**
\markdownEnd
\bye

```

Besides `\markinline{<markdown text>}`, any other delimiter such as `|<markdown text>|` can also be used. This allows potentially unbalanced braces to appear in `<markdown text>`.

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.

```
980 \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.

```
981 \def\yamlInput#1{%
982   \begingroup
983   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
984   \markdownInput{#1}%
985   \endgroup
986 }%
```

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

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

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

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

```

\input markdown
\yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}
\markdownInput{hello.yml}
\bye

```

2.2.1.3 Typesetting TeX from inside Markdown and yaml documents

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

```
987 \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.

```

988 \ExplSyntaxOn
989 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
990 \cs_generate_variant:Nn
991   \tl_const:Nn
992   { NV }
993 \tl_if_exist:NF
994   \c_@@_top_layer_tl
995   {
996     \tl_const:NV
997       \c_@@_top_layer_tl
998       \c_@@_option_layer_plain_tex_tl
999   }

```

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

```
1000 \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.

```

1001 \prop_new:N \g_@@_plain_tex_option_types_prop
1002 \prop_new:N \g_@@_default_plain_tex_options_prop

```

```

1003 \seq_new:N \g_@@_experimental_plain_tex_options_seq
1004 \seq_gput_right:NV
1005   \g_@@_option_layers_seq
1006   \c_@@_option_layer_plain_tex_tl
1007 \cs_new_protected:Nn
1008   \@@_add_plain_tex_option:nnn
1009   {
1010     \@@_add_option:Vnnn
1011     \c_@@_option_layer_plain_tex_tl
1012     { #1 }
1013     { #2 }
1014     { #3 }
1015   }

```

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 $\langle key \rangle = \langle value \rangle$ 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. The `\markdownSetup` macro receives the options to set up as its only argument.

```

1016 \cs_new_protected:Nn
1017   \@@_setup:n
1018   {
1019     \keys_set:nn
1020       { markdown/options }
1021       { #1 }
1022   }
1023 \cs_gset_eq:NN
1024   \markdownSetup
1025   \@@_setup:n

```

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

```

1026 \cs_gset_eq:NN
1027   \yamlSetup
1028   \markdownSetup

```

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

```

1029 \prg_new_protected_conditional:Nnn
1030   \@@_if_option:n
1031   { TF, T, F }
1032   {
1033     \@@_get_option_type:nN
1034     { #1 }
1035     \l_tmpa_tl
1036     \str_if_eq:NNF

```

```

1037     \l_tmpa_tl
1038     \c_@@_option_type_boolean_tl
1039     {
1040         \msg_error:nxxx
1041         { markdown }
1042         { expected-boolean-option }
1043         { #1 }
1044         { \l_tmpa_tl }
1045     }
1046     \@@_get_option_value:nN
1047     { #1 }
1048     \l_tmpa_tl
1049     \str_if_eq:NNTF
1050     \l_tmpa_tl
1051     \c_@@_option_value_true_tl
1052     { \prg_return_true: }
1053     { \prg_return_false: }
1054 }
1055 \msg_new:nnn
1056 { markdown }
1057 { expected-boolean-option }
1058 {
1059     Option~#1~has~type~#2,~
1060     but~a~boolean~was~expected.
1061 }
1062 \let
1063 \markdownIfOption
1064 \@@_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_EX 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_EX document without invoking Lua. 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. It defaults to `false`.

```

1065 \@@_add_plain_tex_option:nnn
1066 { frozenCache }
1067 { boolean }
1068 { 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 (\).

```

1069 \tl_set:Nn
1070   \l_tmpa_tl
1071   {
1072     \str_use:N
1073       \g_@@_unquoted_jobname_str
1074       .markdown.in
1075   }
1076 \bool_if:NT
1077   \g_@@_jobname_quoted_bool
1078   {
1079     \tl_put_left:Nn
1080       \l_tmpa_tl
1081       { " }
1082     \tl_put_right:Nn
1083       \l_tmpa_tl
1084       { " }
1085   }
1086 \cs_generate_variant:Nn
1087   \@@_add_plain_tex_option:nnn
1088   { nnV }
1089   \@@_add_plain_tex_option:nnV
1090   { inputTempFileName }
1091   { path }
1092   \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 \TeX Live 2024, to the value of the `-output-directory` option of your \TeX engine.

In \MikTeX , this automatic detection is currently only supported with \LuaTeX ³⁶. If you need to use \MikTeX and cannot use \LuaTeX , you can either a) fix the automatic

³⁶See <https://github.com/MiKTeX/miktex/issues/1630>.

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

```
1093 \@@_add_plain_tex_option:nnn
1094   { outputDir }
1095   { path }
1096   { . }
```

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 $\text{T}_{\text{E}}\text{X}$ formats such as $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ and $\text{ConT}_{\text{E}}\text{Xt}$. 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 $\text{T}_{\text{E}}\text{X}$ formats should only use the plain $\text{T}_{\text{E}}\text{X}$ default definitions or whether they should also use the format-specific default definitions. Whereas plain $\text{T}_{\text{E}}\text{X}$ 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 $\text{T}_{\text{E}}\text{X}$ 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 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ document:

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

Here is how you would enable the macro in a $\text{ConT}_{\text{E}}\text{Xt}$ document:

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

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

```
1097 \@@_add_plain_tex_option:nnn
1098   { plain }
1099   { boolean }
1100   { 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 \LaTeX document:

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

Here is how you would enable the macro in a 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.

```
1101 \@@_add_plain_tex_option:nnn
1102   { noDefaults }
1103   { boolean }
1104   { 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 \TeX package documentation using the Doc \LaTeX package [10] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
1105 \seq_gput_right:Nn
1106   \g_@@_plain_tex_options_seq
1107   { stripPercentSigns }
1108 \prop_gput:Nnn
1109   \g_@@_plain_tex_option_types_prop
1110   { stripPercentSigns }
1111   { boolean }
1112 \prop_gput:Nnx
1113   \g_@@_default_plain_tex_options_prop
1114   { stripPercentSigns }
1115   { false }
```

2.2.2.5 Generating Plain \TeX Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain \TeX macros and the key-value interface of the `\markdownSetup` macro for the above plain \TeX options.

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

Furthermore, the command also defines options and key-values for subsequently loaded layers that correspond to higher-level T_EX formats such as L^AT_EX and ConT_EXt.

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

```

1116 \cs_new_protected:Nn
1117   \@@_define_option_commands_and_keyvals:
1118   {
1119     \seq_map_inline:Nn
1120       \g_@@_option_layers_seq
1121       {
1122         \seq_map_inline:cn
1123           { g_@@_ ##1 _options_seq }
1124           {
1125             \@@_define_option_command:n
1126             { ####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.

```

1127         \@@_with_various_cases:nn
1128         { ####1 }
1129         {
1130           \@@_define_option_keyval:nnn
1131           { ##1 }
1132           { ####1 }
1133           { #####1 }
1134         }
1135       }
1136     }
1137   }
1138 \cs_new_protected:Nn
1139   \@@_define_option_command:n
1140   {

```

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

```

1141   \bool_set_false:N
1142     \l_tmpa_bool
1143   \seq_map_inline:Nn
1144     \g_@@_option_layers_seq

```

```

1145     {
1146       \seq_if_in:cnT
1147       { g_@@_experimental_ ##1 _options_seq }
1148       { #1 }
1149       {
1150         \bool_set_true:N
1151         \l_tmpa_bool
1152         \seq_map_break:
1153       }
1154     }
1155   \bool_if:NTF
1156   \l_tmpa_bool
1157   {
1158     \_@@_option_tl_to_csname:nN
1159     { #1 }
1160     \l_tmpa_tl
1161     \cs_if_exist:cF
1162     { \l_tmpa_tl }
1163     {
1164       \cs_gset:cpn
1165       { \l_tmpa_tl }
1166       { \markdownOptionExperimental }
1167     }
1168   }
1169   {

```

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

```

1170     \str_if_eq:nnTF
1171     { #1 }
1172     { outputDir }
1173     { \@@_define_option_command_output_dir: }
1174     {

```

Do not override options defined before loading the package.

```

1175     \_@@_option_tl_to_csname:nN
1176     { #1 }
1177     \l_tmpa_tl
1178     \cs_if_exist:cF
1179     { \l_tmpa_tl }
1180     {
1181       \_@@_get_default_option_value:nN
1182       { #1 }
1183       \l_tmpa_tl
1184       \_@@_set_option_value:nV
1185       { #1 }
1186       \l_tmpa_tl
1187     }

```

```

1188         }
1189     }
1190 }
1191 \ExplSyntaxOff
1192 \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.

```

1193 \ExplSyntaxOn
1194 \cs_new:Nn
1195   \@@_define_option_command_output_dir:
1196 {
1197   \cs_if_free:NT
1198     \markdownOptionOutputDir
1199   {
1200     \bool_if:nTF
1201     {
1202       \cs_if_exist_p:N
1203         \luabridge_tl_set:Nn &&
1204       (
1205         \int_compare_p:nNn
1206           { \g_luabridge_method_int }
1207           =
1208           { \c_luabridge_method_directlua_int } ||
1209         \sys_if_shell_unrestricted_p:
1210       )
1211     }
1212   {

```

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.

```

1213       \group_begin:
1214       \cctab_select:N
1215       \c_str_cctab
1216       \luabridge_tl_set:Nn
1217       \l_tmpa_tl
1218       {

```

```

1219             print(
1220                 (status.output_directory)
1221                 or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1222                 or~"."
1223             )
1224         }
1225         \tl_gset:NV
1226         \markdownOptionOutputDir
1227         \l_tmpa_tl
1228         \group_end:
1229     }
1230     {
1231         \tl_gset:Nn
1232         \markdownOptionOutputDir
1233         { . }
1234     }
1235 }
1236 }
1237 \cs_new_protected:Nn
1238 \@@_set_option_value:nn
1239 {
1240     \@@_define_option:n
1241     { #1 }
1242     \@@_get_option_type:nN
1243     { #1 }
1244     \l_tmpa_tl
1245     \str_if_eq:NNTF
1246     \c_@@_option_type_counter_tl
1247     \l_tmpa_tl
1248     {
1249         \@@_option_tl_to_csname:nN
1250         { #1 }
1251         \l_tmpa_tl
1252         \int_gset:cn
1253         { \l_tmpa_tl }
1254         { #2 }
1255     }
1256     {
1257         \@@_option_tl_to_csname:nN
1258         { #1 }
1259         \l_tmpa_tl
1260         \cs_set:cpn
1261         { \l_tmpa_tl }
1262         { #2 }
1263     }
1264 }
1265 \cs_new_protected:Nn

```

```

1266 \@@_prepend_option_value:nn
1267 {
1268     \@@_get_option_value:nN
1269     { #1 }
1270     \l_tmpa_clist
1271     \clist_put_left:Nn
1272     \l_tmpa_clist
1273     { #2 }
1274     \@@_set_option_value:nV
1275     { #1 }
1276     \l_tmpa_clist
1277 }
1278 \cs_new_protected:Nn
1279 \@@_append_option_value:nn
1280 {
1281     \@@_get_option_value:nN
1282     { #1 }
1283     \l_tmpa_clist
1284     \clist_put_right:Nn
1285     \l_tmpa_clist
1286     { #2 }
1287     \@@_set_option_value:nV
1288     { #1 }
1289     \l_tmpa_clist
1290 }
1291 \cs_generate_variant:Nn
1292 \@@_set_option_value:nn
1293 { nV }
1294 \cs_new_protected:Nn
1295 \@@_define_option:n
1296 {
1297     \@@_option_tl_to_csname:nN
1298     { #1 }
1299     \l_tmpa_tl
1300     \cs_if_free:cT
1301     { \l_tmpa_tl }
1302     {
1303         \@@_get_option_type:nN
1304         { #1 }
1305         \l_tmpb_tl
1306         \str_if_eq:NNT
1307         \c_@@_option_type_counter_tl
1308         \l_tmpb_tl
1309         {
1310             \@@_option_tl_to_csname:nN
1311             { #1 }
1312             \l_tmpa_tl

```

```

1313         \int_new:c
1314         { \l_tmpa_tl }
1315     }
1316 }
1317 }
1318 \cs_new_protected:Nn
1319 \@@_define_option_keyval:nnn
1320 {
1321     \prop_get:cnN
1322     { g_@@_ #1 _option_types_prop }
1323     { #2 }
1324     \l_tmpa_tl
1325     \str_if_eq:VVTF
1326     \l_tmpa_tl
1327     \c_@@_option_type_boolean_tl
1328     {
1329         \keys_define:nn
1330         { markdown/options }
1331         {

```

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

```

1332         #3 .code:n = {
1333             \tl_set:Nx
1334             \l_tmpa_tl
1335             {
1336                 \str_case:nnF
1337                 { ##1 }
1338                 {
1339                     { yes } { true }
1340                     { no } { false }
1341                 }
1342                 { ##1 }
1343             }
1344             \@@_set_option_value:nV
1345             { #2 }
1346             \l_tmpa_tl
1347         },
1348         #3 .value_required:n = { false },
1349         #3 .default:n = { true },
1350     }
1351 }
1352 {
1353     \keys_define:nn
1354     { markdown/options }
1355     {
1356         #3 .value_required:n = { true },

```



```

1357         #3 .code:n = {
1358             \@@_set_option_value:nn
1359             { #2 }
1360             { ##1 }
1361         },
1362     }
1363 }

```

For comma-list options, 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.

```

1364     \str_if_eq:VVT
1365     \l_tmpa_tl
1366     \c_@@_option_type_clist_tl
1367     {
1368         \tl_set:Nn
1369         \l_tmpa_tl
1370         { #3 }
1371         \tl_reverse:N
1372         \l_tmpa_tl
1373         \str_if_eq:enF
1374         {
1375             \tl_head:V
1376             \l_tmpa_tl
1377         }
1378         { s }
1379         {
1380             \msg_error:nnn
1381             { markdown }
1382             { malformed-name-for-clist-option }
1383             { #3 }
1384         }
1385         \tl_set:Nx
1386         \l_tmpa_tl
1387         {
1388             \tl_tail:V
1389             \l_tmpa_tl
1390         }
1391         \tl_reverse:N
1392         \l_tmpa_tl
1393         \tl_set_eq:NN
1394         \l_tmpb_tl
1395         \l_tmpa_tl
1396         \tl_put_right:Nn
1397         \l_tmpa_tl

```

```

1398     { .value_required:n = { true } }
1399 \tl_put_right:Nn
1400   \l_tmpb_tl
1401   {
1402     .code:n = {
1403       \@@_append_option_value:nn
1404       { #2 }
1405       { ##1 }
1406     }
1407   }
1408 \keys_define:nV
1409   { markdown/options }
1410   \l_tmpa_tl
1411 \keys_define:nV
1412   { markdown/options }
1413   \l_tmpb_tl

```

To achieve parity with token renderers and renderer prototypes (see sections 2.2.5.48 and 2.2.6.2, respectively) the syntax $\langle key \rangle += \langle value \rangle$ and $\langle key \rangle \$ = \langle value \rangle$ with the same appending semantics as $\langle singular\ key \rangle = \langle value \rangle$ is also supported.

```

1414 \keys_define:nn
1415   { markdown/options }
1416   {
1417     #3~+ .value_required:n = { true },
1418     #3~+ .code:n = {
1419       \@@_append_option_value:nn
1420       { #2 }
1421       { ##1 }
1422     },
1423     #3 + .value_required:n = { true },
1424     #3 + .code:n = {
1425       \@@_append_option_value:nn
1426       { #2 }
1427       { ##1 }
1428     },
1429     #3~$ .value_required:n = { true },
1430     #3~$ .code:n = {
1431       \@@_append_option_value:nn
1432       { #2 }
1433       { ##1 }
1434     },
1435     #3 $ .value_required:n = { true },
1436     #3 $ .code:n = {
1437       \@@_append_option_value:nn
1438       { #2 }
1439       { ##1 }
1440     },

```

Furthermore, similar to token renderers and renderer prototypes, the syntax $\langle key \rangle^{\langle value \rangle}$ is also supported. This syntax prepends $\langle value \rangle$ to the current value as the leftmost item in the list.

```

1441         #3~^ .value_required:n = { true },
1442         #3~^ .code:n = {
1443             \@@_prepend_option_value:nn
1444             { #2 }
1445             { ##1 }
1446         },
1447         #3 ^ .value_required:n = { true },
1448         #3 ^ .code:n = {
1449             \@@_prepend_option_value:nn
1450             { #2 }
1451             { ##1 }
1452         },
1453     }
1454 }
1455 }
1456 \cs_generate_variant:Nn
1457   \clist_set:Nn
1458   { NV }
1459 \cs_generate_variant:Nn
1460   \keys_define:nn
1461   { nV }
1462 \prg_generate_conditional_variant:Nnn
1463   \str_if_eq:nn
1464   { en }
1465   { p, F }
1466 \msg_new:nnn
1467   { markdown }
1468   { malformed-name-for-clist-option }
1469   {
1470     Clist-option-name~#1~does~not~end~with~-s.
1471   }

```

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

```

1472 \str_if_eq:VVT
1473   \c_@@_top_layer_tl
1474   \c_@@_option_layer_plain_tex_tl
1475   {
1476     \@@_define_option_commands_and_keyvals:
1477   }
1478 \ExplSyntaxOff

```

2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a T_EX document (further referred to as a *theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer L^AT_EX package, which provides similar functionality with its `\usetheme` macro [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 T_EX directory structure. For example, loading a theme named `witiko/beamer/MU` would load a T_EX document package named `markdownthemewitiko_beamer_MU.tex`.

If `@<theme version>` is specified after `<theme name>`, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If `@<theme version>` is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [12].

```
1479 \ExplSyntaxOn
1480 \keys_define:nn
1481   { markdown/options }
1482   {
1483     theme .value_required:n = { true },
1484     theme .code:n = {
1485       \@@_set_theme:n
1486       { #1 }
1487     },
1488     import .value_required:n = { true },
1489     import .code:n = {
1490       \tl_set:Nn
1491       \l_tmpa_tl
1492       { #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.

```

1493     \tl_replace_all:NnV
1494     \l_tmpa_tl
1495     { / }
1496     \c_backslash_str
1497     \keys_set:nV
1498     { markdown/options/import }
1499     \l_tmpa_tl
1500 },
1501 }

```

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.

```

1502 \seq_new:N
1503   \g_@@_theme_names_seq
1504 \seq_new:N
1505   \g_@@_theme_versions_seq
1506 \tl_new:N
1507   \g_@@_current_theme_tl
1508 \tl_gset:Nn
1509   \g_@@_current_theme_tl
1510   { }
1511 \seq_gput_right:NV
1512   \g_@@_theme_names_seq
1513   \g_@@_current_theme_tl
1514 \cs_new:Npn
1515   \markdownThemeVersion
1516   { }
1517 \seq_gput_right:NV
1518   \g_@@_theme_versions_seq
1519   \g_@@_current_theme_tl
1520 \cs_new_protected:Nn
1521   \@@_set_theme:n
1522   {

```

First, we validate the theme name.

```

1523   \str_if_in:nnF
1524     { #1 }
1525     { / }
1526     {
1527       \msg_error:nnn
1528         { markdown }
1529         { unqualified-theme-name }

```

```

1530         { #1 }
1531     }
1532     \str_if_in:nnT
1533     { #1 }
1534     { _ }
1535     {
1536         \msg_error:nnn
1537         { markdown }
1538         { underscores-in-theme-name }
1539         { #1 }
1540     }

```

Next, we extract the theme version.

```

1541     \str_if_in:nnTF
1542     { #1 }
1543     { @ }
1544     {
1545         \regex_extract_once:nnN
1546         { (.*?) @ (.*?) }
1547         { #1 }
1548         \l_tmpa_seq
1549         \seq_gpop_left:NN
1550         \l_tmpa_seq
1551         \l_tmpa_tl
1552         \seq_gpop_left:NN
1553         \l_tmpa_seq
1554         \l_tmpa_tl
1555         \tl_gset:Nv
1556         \g_@@_current_theme_tl
1557         \l_tmpa_tl
1558         \seq_gpop_left:NN
1559         \l_tmpa_seq
1560         \l_tmpa_tl
1561         \cs_gset:Npe
1562         \markdownThemeVersion
1563         {
1564             \tl_use:N
1565             \l_tmpa_tl
1566         }
1567     }
1568     {
1569         \tl_gset:Nn
1570         \g_@@_current_theme_tl
1571         { #1 }
1572         \cs_gset:Npn
1573         \markdownThemeVersion
1574         { latest }
1575     }

```

Next, we strip a leading slash, if present, for consistency with snippets, where a leading slash distinguishes absolute snippet names from relative ones. Theme names, however, are currently always absolute, so stripping the slash is only a syntactic normalization and has no semantic effect.

```

1576     \str_if_eq:enT
1577     {
1578         \tl_head:V
1579         \g_@@_current_theme_tl
1580     }
1581     { / }
1582     {
1583         \tl_gset:Ne
1584         \g_@@_current_theme_tl
1585         {
1586             \tl_tail:V
1587             \g_@@_current_theme_tl
1588         }
1589     }

```

Next, we munge the theme name.

```

1590     \str_set:NV
1591     \l_tmpa_str
1592     \g_@@_current_theme_tl
1593     \str_replace_all:Nnn
1594     \l_tmpa_str
1595     { / }
1596     { _ }

```

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

```

1597     \tl_set:NV
1598     \l_tmpa_tl
1599     \g_@@_current_theme_tl
1600     \tl_put_right:Nn
1601     \g_@@_current_theme_tl
1602     { / }
1603     \seq_gput_right:NV
1604     \g_@@_theme_names_seq
1605     \g_@@_current_theme_tl
1606     \seq_gput_right:NV
1607     \g_@@_theme_versions_seq
1608     \markdownThemeVersion
1609     \@@_load_theme:VeV
1610     \l_tmpa_tl
1611     { \markdownThemeVersion }
1612     \l_tmpa_str

```

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

```

1613 \seq_gpop_right:NN
1614 \g_@@_theme_names_seq
1615 \l_tmpa_tl
1616 \seq_get_right:NN
1617 \g_@@_theme_names_seq
1618 \l_tmpa_tl
1619 \tl_gset:Nv
1620 \g_@@_current_theme_tl
1621 \l_tmpa_tl
1622 \seq_gpop_right:NN
1623 \g_@@_theme_versions_seq
1624 \l_tmpa_tl
1625 \seq_get_right:NN
1626 \g_@@_theme_versions_seq
1627 \l_tmpa_tl
1628 \cs_gset:Npe
1629 \markdownThemeVersion
1630 {
1631   \tl_use:N
1632   \l_tmpa_tl
1633 }
1634 }
1635 \msg_new:nnnn
1636 { markdown }
1637 { unqualified-theme-name }
1638 { Won't~load~theme~with~unqualified~name~#1 }
1639 { Theme~names~must~contain~at~least~one~forward~slash }
1640 \msg_new:nnnn
1641 { markdown }
1642 { underscores-in-theme-name }
1643 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1644 { Theme~names~must~not~contain~underscores~in~their~names }
1645 \cs_generate_variant:Nn
1646 \tl_replace_all:Nnn
1647 { NnV }
1648 \cs_generate_variant:Nn
1649 \cs_gset:Npn
1650 { Npe }
1651 \prg_generate_conditional_variant:Nnn
1652 \str_if_eq:nn
1653 { en }
1654 { T }

```

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.

```
1655 \prop_new:N
1656 \g_@@_plain_tex_built_in_themes_prop
```

Built-in plain T_EX themes provided with the Markdown package include:

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

The key-value attribute `caption` can be used to specify the caption of the figure 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[relativeReferences, texComments]{markdown}
\markdownSetup{import=witiko/diagrams@v2}
\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
idea 1
 ((?))
 sub
idea 2
 ((?))
 sub
idea 3
 ((?))
 sub
idea 4
 ((?))
 ...

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

' Diagram title
title Simple Request-Response Flow

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

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

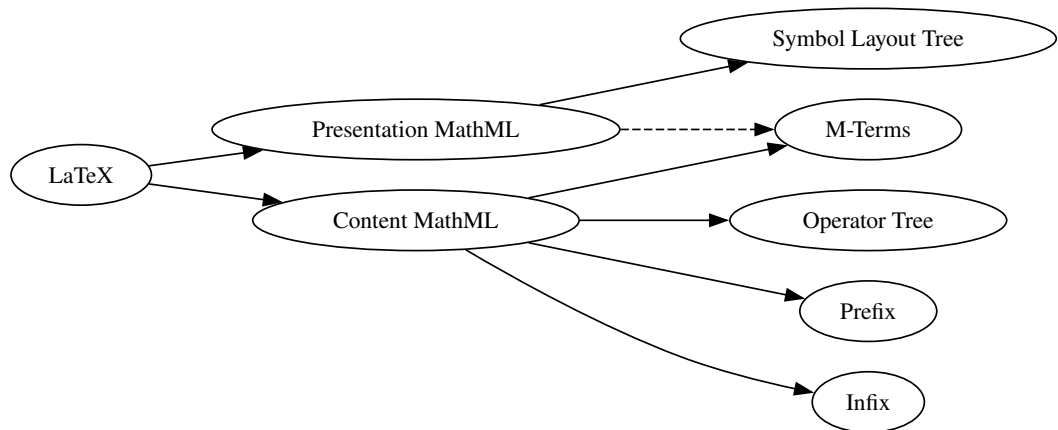
```

```

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

```

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



**Figure 4: An example directed graph**

The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package [@mermaid-js/mermaid-cli](#), and PlantUML installed. All these packages are already included in the Docker image [witiko/markdown](#); consult [Dockerfile](#) to see how they are installed. The theme also requires shell access unless the [frozenCache](#) plain  $\text{\TeX}$  option is enabled.

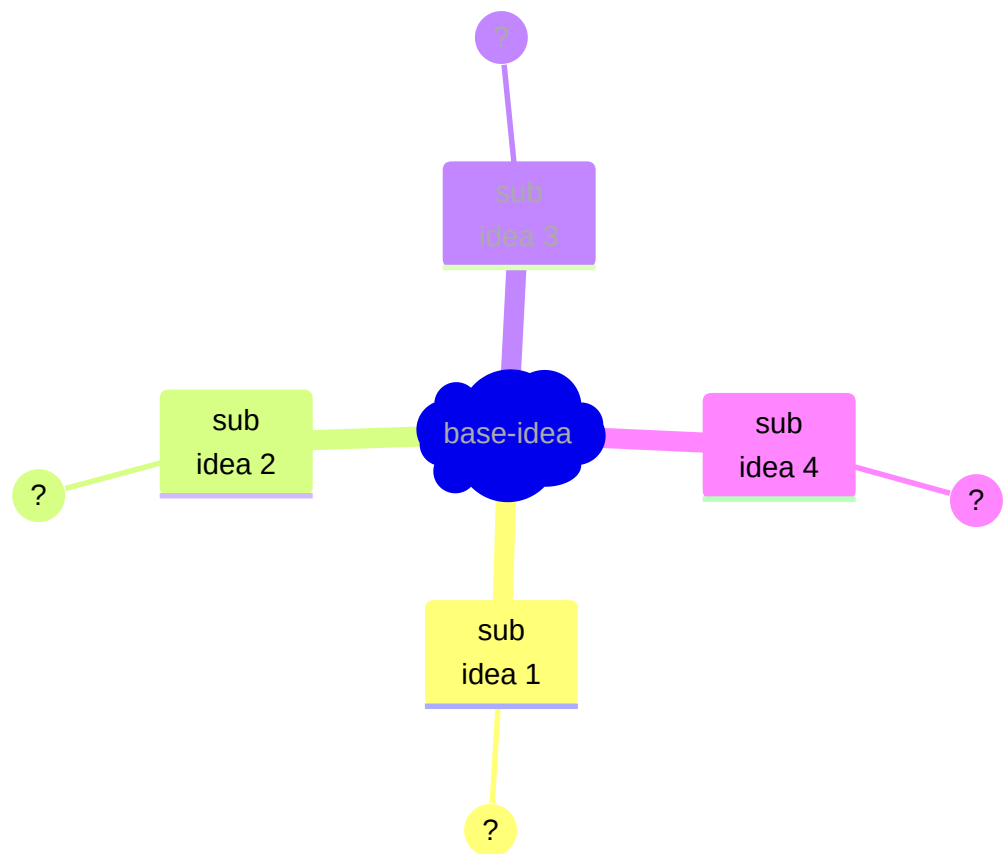
**witiko/glossaries** A theme that defines snippets with integrations for the  $\text{\LaTeX}$  package glossaries. See Section 2.3.5 for more details.

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

```

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

```



**Figure 5: An example mindmap**

## Simple Request-Response Flow

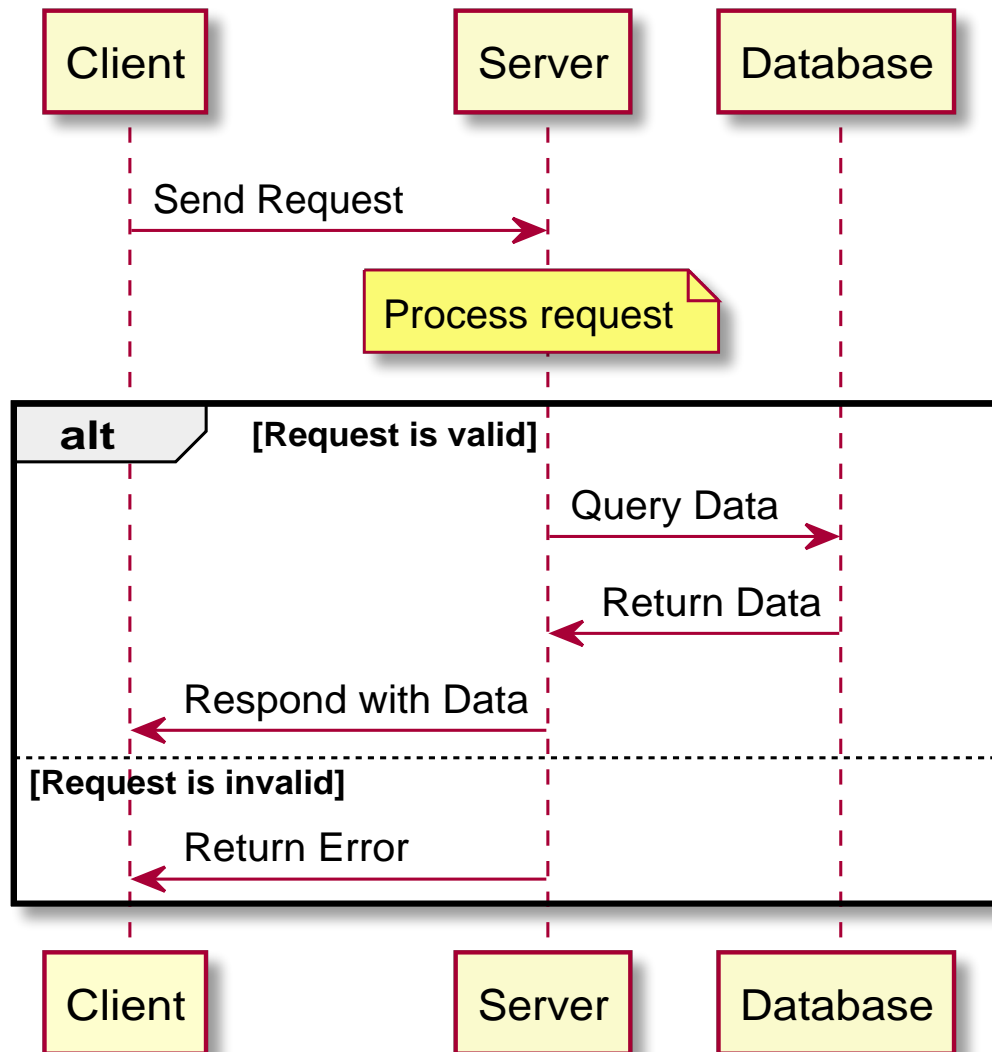


Figure 6: An example UML sequence diagram

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

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

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

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

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



## Chapter 1

# Introduction

### 1.1 Section

#### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

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

theme requires the catchfile  $\text{\LaTeX}$  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
\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.

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.

```

1657 \prop_new:N
1658 \g_@@_snippets_prop
1659 \cs_new_protected:Nn
1660 \@@_setup_snippet:nn
1661 {
1662 \tl_if_empty:nT
1663 { #1 }
1664 {
1665 \msg_error:nnn
1666 { markdown }
1667 { empty-snippet-name }
1668 { #1 }
1669 }
1670 \@@_resolve_snippet_name:nN
1671 { #1 }
1672 \l_tmpa_tl
1673 \@@_if_snippet_exists:nT
1674 { #1 }
1675 {
1676 \msg_warning:nnV
1677 { markdown }
1678 { redefined-snippet }
1679 \l_tmpa_tl
1680 }
1681 \keys_precompile:nnN
1682 { markdown/options }
1683 { #2 }
1684 \l_tmpb_tl
1685 \prop_gput:NVV
1686 \g_@@_snippets_prop
1687 \l_tmpa_tl
1688 \l_tmpb_tl
1689 }
1690 \cs_gset_eq:NN
1691 \markdownSetupSnippet
1692 \@@_setup_snippet:nn

```

```

1693 \msg_new:nnnn
1694 { markdown }
1695 { empty-snippet-name }
1696 { Empty-snippet-name~#1 }
1697 { Pick~a~non-empty~name~for~your~snippet }
1698 \msg_new:nnn
1699 { markdown }
1700 { redefined-snippet }
1701 { Redefined~snippet~#1 }

```

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

```

1702 \tl_new:N
1703 \l_@@_current_snippet_tl
1704 \prg_new_protected_conditional:Nnn
1705 \@@_if_snippet_exists:n
1706 { TF, T }
1707 {
1708 \@@_resolve_snippet_name:nN
1709 { #1 }
1710 \l_@@_current_snippet_tl
1711 \prop_if_in:NVTF
1712 \g_@@_snippets_prop
1713 \l_@@_current_snippet_tl
1714 { \prg_return_true: }
1715 { \prg_return_false: }
1716 }
1717 \cs_gset_eq:NN
1718 \markdownIfSnippetExists
1719 \@@_if_snippet_exists:nTF
1720 \cs_new_protected:Nn
1721 \@@_resolve_snippet_name:nN
1722 {

```

If the provided snippet name starts with a slash (/), consider it as *absolute* and take the rest of the name as the full name of the defined snippet, regardless of the currently processed theme.

```

1723 \str_if_eq:enTF
1724 {
1725 \tl_head:n
1726 { #1 }
1727 }
1728 { / }
1729 {
1730 \tl_set:Ne
1731 #2
1732 {
1733 \tl_tail:n

```



```

1734 { #1 }
1735 }
1736 }
1737 {

```

Otherwise, consider the provided snippet name as *relative* and prepend the name of the currently processed theme to it, if any. The result is then the full name of the defined snippet.

```

1738 \tl_set:NV
1739 #2
1740 \g_@@_current_theme_tl
1741 \tl_put_right:Nn
1742 #2
1743 { #1 }
1744 }
1745 }
1746 \prg_generate_conditional_variant:Nnn
1747 \str_if_eq:nn
1748 { en }
1749 { TF }

```

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

```

1750 \keys_define:nn
1751 { markdown/options }
1752 {
1753 snippet .value_required:n = { true },
1754 snippet .code:n = {
1755 \@@_resolve_snippet_name:nN
1756 { #1 }
1757 \l_tmpa_tl
1758 \@@_if_snippet_exists:nTF
1759 { #1 }
1760 {
1761 \prop_get:NVN
1762 \g_@@_snippets_prop
1763 \l_tmpa_tl
1764 \l_tmpb_tl
1765 \tl_use:N
1766 \l_tmpb_tl
1767 }
1768 {
1769 \msg_error:nnV
1770 { markdown }
1771 { undefined-snippet }
1772 \l_tmpa_tl
1773 }
1774 },
1775 }

```

```

1776 \msg_new:nnn
1777 { markdown }
1778 { undefined-snippet }
1779 { Can't~invoke~undefined~snippet~#1 }
1780 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

If the `romanNumerals` snippet were defined in the `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,
},
}

```

```

1781 \ExplSyntaxOn
1782 \tl_new:N
1783 \l_@@_import_current_theme_tl
1784 \keys_define:nn
1785 { markdown/options/import }
1786 {

```

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

```

1787 unknown .value_required:n = { false },
1788 unknown .default:n = {},
1789 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.

```

1790 \tl_set_eq:NN
1791 \l_@@_import_current_theme_tl
1792 \l_keys_key_str
1793 \tl_replace_all:NVn
1794 \l_@@_import_current_theme_tl
1795 \c_backslash_str
1796 { / }

```

Here, we import the snippets.

```

1797 \clist_map_inline:nn
1798 { #1 }
1799 {
1800 \regex_extract_once:nnNTF
1801 { ^(.*)\s+as\s+(.*)$ }
1802 { ##1 }
1803 \l_tmpa_seq

```

```

1804 {
1805 \seq_pop:NN
1806 \l_tmpa_seq
1807 \l_tmpa_tl
1808 \seq_pop:NN
1809 \l_tmpa_seq
1810 \l_tmpa_tl
1811 \seq_pop:NN
1812 \l_tmpa_seq
1813 \l_tmpb_tl
1814 }
1815 {
1816 \tl_set:Nn
1817 \l_tmpa_tl
1818 { ##1 }
1819 \tl_set:Nn
1820 \l_tmpb_tl
1821 { ##1 }
1822 }
1823 \tl_put_left:Nn
1824 \l_tmpa_tl
1825 { / }
1826 \tl_put_left:NV
1827 \l_tmpa_tl
1828 \l_@@_import_current_theme_tl
1829 \@@_setup_snippet:Vx
1830 \l_tmpb_tl
1831 { snippet = { \l_tmpa_tl } }
1832 }

```

Here, we load the theme.

```

1833 \@@_set_theme:V
1834 \l_@@_import_current_theme_tl
1835 },
1836 }
1837 \cs_generate_variant:Nn
1838 \tl_replace_all:Nnn
1839 { NVn }
1840 \cs_generate_variant:Nn
1841 \@@_set_theme:n
1842 { V }
1843 \cs_generate_variant:Nn
1844 \@@_setup_snippet:nn
1845 { Vx }

```

### 2.2.5 Token Renderers

The following  $\TeX$  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.

```
1846 \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.

```
1847 \prop_new:N \g_@@_renderer_arities_prop
```

```
1848 \ExplSyntaxOff
```

#### 2.2.5.1 Acronym Renderers

The following macro is only produced when the option `acronyms` is non-empty.

`\markdownRendererAcronym` represents an acronym, initialism, or another all-caps sequence. The macro receives a single attribute that corresponds to the sequence.

```
1849 \ExplSyntaxOn
1850 \cs_gset_protected:Npn
1851 \markdownRendererAcronym
1852 {
1853 \markdownRendererAcronymPrototype
1854 }
1855 \seq_gput_right:Nn
1856 \g_@@_renderers_seq
1857 { acronym }
1858 \prop_gput:Nnn
1859 \g_@@_renderer_arities_prop
1860 { acronym }
1861 { 1 }
1862 \ExplSyntaxOff
```

#### 2.2.5.2 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 mark-down element (`id=" $\langle identifier \rangle$ "` in HTML and `# $\langle identifier \rangle$`  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 mark-down element (`class=" $\langle class name \rangle$  ..."` in HTML and `. $\langle class name \rangle$`  in markdown attributes). The macro receives a single attribute that corresponds to the  $\langle class name \rangle$ .

`\markdownRendererAttributeKeyValue` represents an 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.

```

1863 \ExplSyntaxOn
1864 \cs_gset_protected:Npn
1865 \markdownRendererAttributeIdentifier
1866 {
1867 \markdownRendererAttributeIdentifierPrototype
1868 }
1869 \seq_gput_right:Nn
1870 \g_@@_renderers_seq
1871 { attributeIdentifier }
1872 \prop_gput:Nnn
1873 \g_@@_renderer_arities_prop
1874 { attributeIdentifier }
1875 { 1 }
1876 \cs_gset_protected:Npn
1877 \markdownRendererAttributeClassName
1878 {
1879 \markdownRendererAttributeClassNamePrototype
1880 }
1881 \seq_gput_right:Nn
1882 \g_@@_renderers_seq
1883 { attributeClassName }
1884 \prop_gput:Nnn
1885 \g_@@_renderer_arities_prop
1886 { attributeClassName }
1887 { 1 }
1888 \cs_gset_protected:Npn
1889 \markdownRendererAttributeKeyValue
1890 {
1891 \markdownRendererAttributeKeyValuePrototype
1892 }
1893 \seq_gput_right:Nn
1894 \g_@@_renderers_seq
1895 { attributeKeyValue }
1896 \prop_gput:Nnn
1897 \g_@@_renderer_arities_prop

```

```

1898 { attributeKeyValue }
1899 { 2 }
1900 \ExplSyntaxOff

```

### 2.2.5.3 Basic html Renderers

The `\markdownRendererInlineHtmlComment` macro represents the contents of an inline HTML comment. The macro receives a single argument that corresponds to the contents of the HTML comment.

The `\markdownRendererInlineHtmlTag` macro represents an opening, closing, or empty inline HTML tag. 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.

These macros will only be produced when the `html` option is enabled and the `htmlOutput` option is set to `basic`.

```

1901 \ExplSyntaxOn
1902 \cs_gset_protected:Npn
1903 \markdownRendererInlineHtmlComment
1904 {
1905 \markdownRendererInlineHtmlCommentPrototype
1906 }
1907 \seq_gput_right:Nn
1908 \g_@@_renderers_seq
1909 { inlineHtmlComment }
1910 \prop_gput:Nnn
1911 \g_@@_renderer_arities_prop
1912 { inlineHtmlComment }
1913 { 1 }
1914 \cs_gset_protected:Npn
1915 \markdownRendererInlineHtmlTag
1916 {
1917 \markdownRendererInlineHtmlTagPrototype
1918 }
1919 \seq_gput_right:Nn
1920 \g_@@_renderers_seq
1921 { inlineHtmlTag }
1922 \prop_gput:Nnn
1923 \g_@@_renderer_arities_prop
1924 { inlineHtmlTag }
1925 { 1 }
1926 \cs_gset_protected:Npn
1927 \markdownRendererInputBlockHtmlElement
1928 {

```



```

1929 \markdownRendererInputBlockHtmlElementPrototype
1930 }
1931 \seq_gput_right:Nn
1932 \g_@@_renderers_seq
1933 { inputBlockHtmlElement }
1934 \prop_gput:Nnn
1935 \g_@@_renderer_arities_prop
1936 { inputBlockHtmlElement }
1937 { 1 }
1938 \ExplSyntaxOff

```

#### 2.2.5.4 Block Quote Renderers

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

```

1939 \ExplSyntaxOn
1940 \cs_gset_protected:Npn
1941 \markdownRendererBlockQuoteBegin
1942 {
1943 \markdownRendererBlockQuoteBeginPrototype
1944 }
1945 \seq_gput_right:Nn
1946 \g_@@_renderers_seq
1947 { blockQuoteBegin }
1948 \prop_gput:Nnn
1949 \g_@@_renderer_arities_prop
1950 { blockQuoteBegin }
1951 { 0 }
1952 \ExplSyntaxOff

```

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

```

1953 \ExplSyntaxOn
1954 \cs_gset_protected:Npn
1955 \markdownRendererBlockQuoteEnd
1956 {
1957 \markdownRendererBlockQuoteEndPrototype
1958 }
1959 \seq_gput_right:Nn
1960 \g_@@_renderers_seq
1961 { blockQuoteEnd }
1962 \prop_gput:Nnn
1963 \g_@@_renderer_arities_prop
1964 { blockQuoteEnd }
1965 { 0 }
1966 \ExplSyntaxOff

```

### 2.2.5.5 Bracketed Spans

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

The `\markdownRendererBracketedSpan` macro represents an inline bracketed span. It receives a single argument that corresponds to the inline bracketed span.

```
1967 \ExplSyntaxOn
1968 \cs_gset_protected:Npn
1969 \markdownRendererBracketedSpan
1970 {
1971 \markdownRendererBracketedSpanPrototype
1972 }
1973 \seq_gput_right:Nn
1974 \g_@@_renderers_seq
1975 { bracketedSpan }
1976 \prop_gput:Nnn
1977 \g_@@_renderer_arities_prop
1978 { bracketedSpan }
1979 { 1 }
1980 \ExplSyntaxOff
```

### 2.2.5.6 Bracketed Span 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.

```
1981 \ExplSyntaxOn
1982 \cs_gset_protected:Npn
1983 \markdownRendererBracketedSpanAttributeContextBegin
1984 {
1985 \markdownRendererBracketedSpanAttributeContextBeginPrototype
1986 }
1987 \seq_gput_right:Nn
1988 \g_@@_renderers_seq
1989 { bracketedSpanAttributeContextBegin }
1990 \prop_gput:Nnn
1991 \g_@@_renderer_arities_prop
1992 { bracketedSpanAttributeContextBegin }
1993 { 0 }
1994 \cs_gset_protected:Npn
1995 \markdownRendererBracketedSpanAttributeContextEnd
1996 {
1997 \markdownRendererBracketedSpanAttributeContextEndPrototype
1998 }
1999 \seq_gput_right:Nn
```

```

2000 \g_@@_renderers_seq
2001 { bracketedSpanAttributeContextEnd }
2002 \prop_gput:Nnn
2003 \g_@@_renderer_arities_prop
2004 { bracketedSpanAttributeContextEnd }
2005 { 0 }
2006 \ExplSyntaxOff

```

### 2.2.5.7 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.

```

2007 \ExplSyntaxOn
2008 \cs_gset_protected:Npn
2009 \markdownRendererUlBegin
2010 {
2011 \markdownRendererUlBeginPrototype
2012 }
2013 \seq_gput_right:Nn
2014 \g_@@_renderers_seq
2015 { ulBegin }
2016 \prop_gput:Nnn
2017 \g_@@_renderer_arities_prop
2018 { ulBegin }
2019 { 0 }
2020 \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.

```

2021 \ExplSyntaxOn
2022 \cs_gset_protected:Npn
2023 \markdownRendererUlBeginTight
2024 {
2025 \markdownRendererUlBeginTightPrototype
2026 }
2027 \seq_gput_right:Nn
2028 \g_@@_renderers_seq
2029 { ulBeginTight }
2030 \prop_gput:Nnn
2031 \g_@@_renderer_arities_prop
2032 { ulBeginTight }
2033 { 0 }
2034 \ExplSyntaxOff

```

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

```

2035 \ExplSyntaxOn
2036 \cs_gset_protected:Npn
2037 \markdownRendererUListItem
2038 {
2039 \markdownRendererUListItemPrototype
2040 }
2041 \seq_gput_right:Nn
2042 \g_@@_renderers_seq
2043 { ulItem }
2044 \prop_gput:Nnn
2045 \g_@@_renderer_arities_prop
2046 { ulItem }
2047 { 0 }
2048 \ExplSyntaxOff

```

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

```

2049 \ExplSyntaxOn
2050 \cs_gset_protected:Npn
2051 \markdownRendererUItemEnd
2052 {
2053 \markdownRendererUItemEndPrototype
2054 }
2055 \seq_gput_right:Nn
2056 \g_@@_renderers_seq
2057 { ulItemEnd }
2058 \prop_gput:Nnn
2059 \g_@@_renderer_arities_prop
2060 { ulItemEnd }
2061 { 0 }
2062 \ExplSyntaxOff

```

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

```

2063 \ExplSyntaxOn
2064 \cs_gset_protected:Npn
2065 \markdownRendererUListEnd
2066 {
2067 \markdownRendererUListEndPrototype
2068 }
2069 \seq_gput_right:Nn
2070 \g_@@_renderers_seq
2071 { ulEnd }

```

```

2072 \prop_gput:Nnn
2073 \g_@@_renderer_arities_prop
2074 { ulEnd }
2075 { 0 }
2076 \ExplSyntaxOff

```

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

```

2077 \ExplSyntaxOn
2078 \cs_gset_protected:Npn
2079 \markdownRendererUEndTight
2080 {
2081 \markdownRendererUEndTightPrototype
2082 }
2083 \seq_gput_right:Nn
2084 \g_@@_renderers_seq
2085 { ulEndTight }
2086 \prop_gput:Nnn
2087 \g_@@_renderer_arities_prop
2088 { ulEndTight }
2089 { 0 }
2090 \ExplSyntaxOff

```

#### 2.2.5.8 Citation Renderers

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

```

2091 \ExplSyntaxOn
2092 \cs_gset_protected:Npn
2093 \markdownRendererCite
2094 {
2095 \markdownRendererCitePrototype
2096 }
2097 \seq_gput_right:Nn
2098 \g_@@_renderers_seq
2099 { cite }
2100 \prop_gput:Nnn
2101 \g_@@_renderer_arities_prop
2102 { cite }
2103 { 1 }

```

2104 \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.

```
2105 \ExplSyntaxOn
2106 \cs_gset_protected:Npn
2107 \markdownRendererTextCite
2108 {
2109 \markdownRendererTextCitePrototype
2110 }
2111 \seq_gput_right:Nn
2112 \g_@@_renderers_seq
2113 { textCite }
2114 \prop_gput:Nnn
2115 \g_@@_renderer_arities_prop
2116 { textCite }
2117 { 1 }
2118 \ExplSyntaxOff
```

#### 2.2.5.9 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.

```
2119 \ExplSyntaxOn
2120 \cs_gset_protected:Npn
2121 \markdownRendererInputVerbatim
2122 {
2123 \markdownRendererInputVerbatimPrototype
2124 }
2125 \seq_gput_right:Nn
2126 \g_@@_renderers_seq
2127 { inputVerbatim }
2128 \prop_gput:Nnn
2129 \g_@@_renderer_arities_prop
2130 { inputVerbatim }
2131 { 1 }
2132 \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.

```

2133 \ExplSyntaxOn
2134 \cs_gset_protected:Npn
2135 \markdownRendererInputFencedCode
2136 {
2137 \markdownRendererInputFencedCodePrototype
2138 }
2139 \seq_gput_right:Nn
2140 \g_@@_renderers_seq
2141 { inputFencedCode }
2142 \prop_gput:Nnn
2143 \g_@@_renderer_arities_prop
2144 { inputFencedCode }
2145 { 3 }
2146 \ExplSyntaxOff

```

#### 2.2.5.10 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.

```

2147 \ExplSyntaxOn
2148 \cs_gset_protected:Npn
2149 \markdownRendererCodeSpan
2150 {
2151 \markdownRendererCodeSpanPrototype
2152 }
2153 \seq_gput_right:Nn
2154 \g_@@_renderers_seq
2155 { codeSpan }
2156 \prop_gput:Nnn
2157 \g_@@_renderer_arities_prop
2158 { codeSpan }
2159 { 1 }
2160 \ExplSyntaxOff

```

#### 2.2.5.11 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.

```

2161 \ExplSyntaxOn
2162 \cs_gset_protected:Npn
2163 \markdownRendererCodeSpanAttributeContextBegin
2164 {
2165 \markdownRendererCodeSpanAttributeContextBeginPrototype

```

```

2166 }
2167 \seq_gput_right:Nn
2168 \g_@@_renderers_seq
2169 { codeSpanAttributeContextBegin }
2170 \prop_gput:Nnn
2171 \g_@@_renderer_arities_prop
2172 { codeSpanAttributeContextBegin }
2173 { 0 }
2174 \cs_gset_protected:Npn
2175 \markdownRendererCodeSpanAttributeContextEnd
2176 {
2177 \markdownRendererCodeSpanAttributeContextEndPrototype
2178 }
2179 \seq_gput_right:Nn
2180 \g_@@_renderers_seq
2181 { codeSpanAttributeContextEnd }
2182 \prop_gput:Nnn
2183 \g_@@_renderer_arities_prop
2184 { codeSpanAttributeContextEnd }
2185 { 0 }
2186 \ExplSyntaxOff

```

#### 2.2.5.12 CommonMark Block html Renderers

When the `html` option is enabled and `htmlOutput` is set to `commonmark`, the following fine-grained renderers are produced instead of `\markdownRendererInputBlockHtmlElement` to distinguish between different top-level non-terminal elements of CommonMark’s grammar for raw HTML<sup>37</sup>:

1. `\markdownRendererInputBlockHtmlCdataElement` – A CDATA block-level HTML element such as `<script>`, `<pre>`, `<style>`, or `<textarea>` that starts on a line of its own
2. `\markdownRendererBlockHtmlComment` – A comment that starts on a line of its own
3. `\markdownRendererInputBlockHtmlProcessingInstruction` – A processing instruction that starts on a line of its own
4. `\markdownRendererInputBlockHtmlDeclaration` – A declaration that starts on a line of its own
5. `\markdownRendererInputBlockHtmlCdataSection` – A CDATA section that starts on a line of its own
6. `\markdownRendererInputBlockHtmlPcdataElement` – A PCDATA block-level HTML element that starts on a line of its own
7. `\markdownRendererBlockHtmlStandaloneTag` – A non-CDATA HTML tag on a line of its own

<sup>37</sup>See <https://spec.commonmark.org/0.31.2/#html-blocks>.



Each of these macros receive a single argument. For all macros except 2 and 7, it is the filename of a file containing the corresponding HTML text. In contrast, macros 2 and 7 receive the text within the HTML comment and the HTML text, respectively.

```

2187 \ExplSyntaxOn
2188 \cs_gset_protected:Npn
2189 \markdownRendererInputBlockHtmlCdataElement
2190 {
2191 \markdownRendererInputBlockHtmlCdataElementPrototype
2192 }
2193 \seq_gput_right:Nn
2194 \g_@@_renderers_seq
2195 { inputBlockHtmlCdataElement }
2196 \prop_gput:Nnn
2197 \g_@@_renderer_arities_prop
2198 { inputBlockHtmlCdataElement }
2199 { 1 }
2200 \cs_gset_protected:Npn
2201 \markdownRendererBlockHtmlComment
2202 {
2203 \markdownRendererBlockHtmlCommentPrototype
2204 }
2205 \seq_gput_right:Nn
2206 \g_@@_renderers_seq
2207 { blockHtmlComment }
2208 \prop_gput:Nnn
2209 \g_@@_renderer_arities_prop
2210 { blockHtmlComment }
2211 { 1 }
2212 \cs_gset_protected:Npn
2213 \markdownRendererInputBlockHtmlProcessingInstruction
2214 {
2215 \markdownRendererInputBlockHtmlProcessingInstructionPrototype
2216 }
2217 \seq_gput_right:Nn
2218 \g_@@_renderers_seq
2219 { inputBlockHtmlProcessingInstruction }
2220 \prop_gput:Nnn
2221 \g_@@_renderer_arities_prop
2222 { inputBlockHtmlProcessingInstruction }
2223 { 1 }
2224 \cs_gset_protected:Npn
2225 \markdownRendererInputBlockHtmlDeclaration
2226 {
2227 \markdownRendererInputBlockHtmlDeclarationPrototype
2228 }
2229 \seq_gput_right:Nn

```

```

2230 \g_@@_renderers_seq
2231 { inputBlockHtmlDeclaration }
2232 \prop_gput:Nnn
2233 \g_@@_renderer_arities_prop
2234 { inputBlockHtmlDeclaration }
2235 { 1 }
2236 \cs_gset_protected:Npn
2237 \markdownRendererInputBlockHtmlCdataSection
2238 {
2239 \markdownRendererInputBlockHtmlCdataSectionPrototype
2240 }
2241 \seq_gput_right:Nn
2242 \g_@@_renderers_seq
2243 { inputBlockHtmlCdataSection }
2244 \prop_gput:Nnn
2245 \g_@@_renderer_arities_prop
2246 { inputBlockHtmlCdataSection }
2247 { 1 }
2248 \cs_gset_protected:Npn
2249 \markdownRendererInputBlockHtmlPcdataElement
2250 {
2251 \markdownRendererInputBlockHtmlPcdataElementPrototype
2252 }
2253 \seq_gput_right:Nn
2254 \g_@@_renderers_seq
2255 { inputBlockHtmlPcdataElement }
2256 \prop_gput:Nnn
2257 \g_@@_renderer_arities_prop
2258 { inputBlockHtmlPcdataElement }
2259 { 1 }
2260 \cs_gset_protected:Npn
2261 \markdownRendererBlockHtmlStandaloneTag
2262 {
2263 \markdownRendererBlockHtmlStandaloneTagPrototype
2264 }
2265 \seq_gput_right:Nn
2266 \g_@@_renderers_seq
2267 { blockHtmlStandaloneTag }
2268 \prop_gput:Nnn
2269 \g_@@_renderer_arities_prop
2270 { blockHtmlStandaloneTag }
2271 { 1 }
2272 \ExplSyntaxOff

```

### 2.2.5.13 CommonMark Raw html Renderers

When the `html` option is enabled and `htmlOutput` is set to `commonmark`, the

following renderers are produced instead of `\markdownRendererInlineHtmlTag` to distinguish between different top-level non-terminal elements of CommonMark’s grammar for raw HTML<sup>38</sup>:

- `\markdownRendererInlineHtmlProcessingInstruction` – A processing instruction
- `\markdownRendererInlineHtmlDeclaration` – A declaration
- `\markdownRendererInlineHtmlCdataSection` – A CDATA section
- `\markdownRendererInlineHtmlOpenTag` – An opening tag
- `\markdownRendererInlineHtmlCloseTag` – A closing tag
- `\markdownRendererInlineHtmlEmptyTag` – An empty tag

Each of these macros receives a single argument with the HTML text.

```

2273 \ExplSyntaxOn
2274 \cs_gset_protected:Npn
2275 \markdownRendererInlineHtmlProcessingInstruction
2276 {
2277 \markdownRendererInlineHtmlProcessingInstructionPrototype
2278 }
2279 \seq_gput_right:Nn
2280 \g_@@_renderers_seq
2281 { inlineHtmlProcessingInstruction }
2282 \prop_gput:Nnn
2283 \g_@@_renderer_arities_prop
2284 { inlineHtmlProcessingInstruction }
2285 { 1 }
2286 \cs_gset_protected:Npn
2287 \markdownRendererInlineHtmlDeclaration
2288 {
2289 \markdownRendererInlineHtmlDeclarationPrototype
2290 }
2291 \seq_gput_right:Nn
2292 \g_@@_renderers_seq
2293 { inlineHtmlDeclaration }
2294 \prop_gput:Nnn
2295 \g_@@_renderer_arities_prop
2296 { inlineHtmlDeclaration }
2297 { 1 }
2298 \cs_gset_protected:Npn
2299 \markdownRendererInlineHtmlCdataSection
2300 {
2301 \markdownRendererInlineHtmlCdataSectionPrototype
2302 }
2303 \seq_gput_right:Nn

```

---

<sup>38</sup>See <https://spec.commonmark.org/0.31.2/#raw-html>.

```

2304 \g_@@_renderers_seq
2305 { inlineHtmlCdataSection }
2306 \prop_gput:Nnn
2307 \g_@@_renderer_arities_prop
2308 { inlineHtmlCdataSection }
2309 { 1 }
2310 \cs_gset_protected:Npn
2311 \markdownRendererInlineHtmlOpenTag
2312 {
2313 \markdownRendererInlineHtmlOpenTagPrototype
2314 }
2315 \seq_gput_right:Nn
2316 \g_@@_renderers_seq
2317 { inlineHtmlOpenTag }
2318 \prop_gput:Nnn
2319 \g_@@_renderer_arities_prop
2320 { inlineHtmlOpenTag }
2321 { 1 }
2322 \cs_gset_protected:Npn
2323 \markdownRendererInlineHtmlCloseTag
2324 {
2325 \markdownRendererInlineHtmlCloseTagPrototype
2326 }
2327 \seq_gput_right:Nn
2328 \g_@@_renderers_seq
2329 { inlineHtmlCloseTag }
2330 \prop_gput:Nnn
2331 \g_@@_renderer_arities_prop
2332 { inlineHtmlCloseTag }
2333 { 1 }
2334 \cs_gset_protected:Npn
2335 \markdownRendererInlineHtmlEmptyTag
2336 {
2337 \markdownRendererInlineHtmlEmptyTagPrototype
2338 }
2339 \seq_gput_right:Nn
2340 \g_@@_renderers_seq
2341 { inlineHtmlEmptyTag }
2342 \prop_gput:Nnn
2343 \g_@@_renderer_arities_prop
2344 { inlineHtmlEmptyTag }
2345 { 1 }
2346 \ExplSyntaxOff

```

#### 2.2.5.14 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.

```

2347 \ExplSyntaxOn
2348 \cs_gset_protected:Npn
2349 \markdownRendererContentBlock
2350 {
2351 \markdownRendererContentBlockPrototype
2352 }
2353 \seq_gput_right:Nn
2354 \g_@@_renderers_seq
2355 { contentBlock }
2356 \prop_gput:Nnn
2357 \g_@@_renderer_arities_prop
2358 { contentBlock }
2359 { 4 }
2360 \ExplSyntaxOff

```

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

```

2361 \ExplSyntaxOn
2362 \cs_gset_protected:Npn
2363 \markdownRendererContentBlockOnlineImage
2364 {
2365 \markdownRendererContentBlockOnlineImagePrototype
2366 }
2367 \seq_gput_right:Nn
2368 \g_@@_renderers_seq
2369 { contentBlockOnlineImage }
2370 \prop_gput:Nnn
2371 \g_@@_renderer_arities_prop
2372 { contentBlockOnlineImage }
2373 { 4 }
2374 \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>39</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

---

<sup>39</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

2375 \ExplSyntaxOn
2376 \cs_gset_protected:Npn
2377 \markdownRendererContentBlockCode
2378 {
2379 \markdownRendererContentBlockCodePrototype
2380 }
2381 \seq_gput_right:Nn
2382 \g_@@_renderers_seq
2383 { contentBlockCode }
2384 \prop_gput:Nnn
2385 \g_@@_renderer_arities_prop
2386 { contentBlockCode }
2387 { 5 }
2388 \ExplSyntaxOff

```

#### 2.2.5.15 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.

```

2389 \ExplSyntaxOn
2390 \cs_gset_protected:Npn
2391 \markdownRendererDlBegin
2392 {
2393 \markdownRendererDlBeginPrototype
2394 }
2395 \seq_gput_right:Nn
2396 \g_@@_renderers_seq
2397 { dlBegin }
2398 \prop_gput:Nnn
2399 \g_@@_renderer_arities_prop
2400 { dlBegin }
2401 { 0 }
2402 \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.

```

2403 \ExplSyntaxOn
2404 \cs_gset_protected:Npn
2405 \markdownRendererDlBeginTight
2406 {
2407 \markdownRendererDlBeginTightPrototype
2408 }
2409 \seq_gput_right:Nn
2410 \g_@@_renderers_seq
2411 { dlBeginTight }
2412 \prop_gput:Nnn
2413 \g_@@_renderer_arities_prop
2414 { dlBeginTight }
2415 { 0 }
2416 \ExplSyntaxOff

```

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

```

2417 \ExplSyntaxOn
2418 \cs_gset_protected:Npn
2419 \markdownRendererDlItem
2420 {
2421 \markdownRendererDlItemPrototype
2422 }
2423 \seq_gput_right:Nn
2424 \g_@@_renderers_seq
2425 { dlItem }
2426 \prop_gput:Nnn
2427 \g_@@_renderer_arities_prop
2428 { dlItem }
2429 { 1 }
2430 \ExplSyntaxOff

```

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

```

2431 \ExplSyntaxOn
2432 \cs_gset_protected:Npn
2433 \markdownRendererDlItemEnd
2434 {
2435 \markdownRendererDlItemEndPrototype
2436 }
2437 \seq_gput_right:Nn
2438 \g_@@_renderers_seq

```

```

2439 { dlItemEnd }
2440 \prop_gput:Nnn
2441 \g_@@_renderer_arities_prop
2442 { dlItemEnd }
2443 { 0 }
2444 \ExplSyntaxOff

```

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

```

2445 \ExplSyntaxOn
2446 \cs_gset_protected:Npn
2447 \markdownRendererDlDefinitionBegin
2448 {
2449 \markdownRendererDlDefinitionBeginPrototype
2450 }
2451 \seq_gput_right:Nn
2452 \g_@@_renderers_seq
2453 { dlDefinitionBegin }
2454 \prop_gput:Nnn
2455 \g_@@_renderer_arities_prop
2456 { dlDefinitionBegin }
2457 { 0 }
2458 \ExplSyntaxOff

```

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

```

2459 \ExplSyntaxOn
2460 \cs_gset_protected:Npn
2461 \markdownRendererDlDefinitionEnd
2462 {
2463 \markdownRendererDlDefinitionEndPrototype
2464 }
2465 \seq_gput_right:Nn
2466 \g_@@_renderers_seq
2467 { dlDefinitionEnd }
2468 \prop_gput:Nnn
2469 \g_@@_renderer_arities_prop
2470 { dlDefinitionEnd }
2471 { 0 }
2472 \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.

```

2473 \ExplSyntaxOn
2474 \cs_gset_protected:Npn

```



```

2475 \markdownRendererDlEnd
2476 {
2477 \markdownRendererDlEndPrototype
2478 }
2479 \seq_gput_right:Nn
2480 \g_@@_renderers_seq
2481 { dlEnd }
2482 \prop_gput:Nnn
2483 \g_@@_renderer_arities_prop
2484 { dlEnd }
2485 { 0 }
2486 \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.

```

2487 \ExplSyntaxOn
2488 \cs_gset_protected:Npn
2489 \markdownRendererDlEndTight
2490 {
2491 \markdownRendererDlEndTightPrototype
2492 }
2493 \seq_gput_right:Nn
2494 \g_@@_renderers_seq
2495 { dlEndTight }
2496 \prop_gput:Nnn
2497 \g_@@_renderer_arities_prop
2498 { dlEndTight }
2499 { 0 }
2500 \ExplSyntaxOff

```

#### 2.2.5.16 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.

```

2501 \ExplSyntaxOn
2502 \cs_gset_protected:Npn
2503 \markdownRendererEllipsis
2504 {
2505 \markdownRendererEllipsisPrototype
2506 }
2507 \seq_gput_right:Nn
2508 \g_@@_renderers_seq
2509 { ellipsis }

```

```

2510 \prop_gput:Nnn
2511 \g_@@_renderer_arities_prop
2512 { ellipsis }
2513 { 0 }
2514 \ExplSyntaxOff

```

### 2.2.5.17 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.

```

2515 \ExplSyntaxOn
2516 \cs_gset_protected:Npn
2517 \markdownRendererEmphasis
2518 {
2519 \markdownRendererEmphasisPrototype
2520 }
2521 \seq_gput_right:Nn
2522 \g_@@_renderers_seq
2523 { emphasis }
2524 \prop_gput:Nnn
2525 \g_@@_renderer_arities_prop
2526 { emphasis }
2527 { 1 }
2528 \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.

```

2529 \ExplSyntaxOn
2530 \cs_gset_protected:Npn
2531 \markdownRendererStrongEmphasis
2532 {
2533 \markdownRendererStrongEmphasisPrototype
2534 }
2535 \seq_gput_right:Nn
2536 \g_@@_renderers_seq
2537 { strongEmphasis }
2538 \prop_gput:Nnn
2539 \g_@@_renderer_arities_prop
2540 { strongEmphasis }
2541 { 1 }
2542 \ExplSyntaxOff

```

### 2.2.5.18 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.

```

2543 \ExplSyntaxOn
2544 \cs_gset_protected:Npn
2545 \markdownRendererFencedCodeAttributeContextBegin
2546 {
2547 \markdownRendererFencedCodeAttributeContextBeginPrototype
2548 }
2549 \seq_gput_right:Nn
2550 \g_@@_renderers_seq
2551 { fencedCodeAttributeContextBegin }
2552 \prop_gput:Nnn
2553 \g_@@_renderer_arities_prop
2554 { fencedCodeAttributeContextBegin }
2555 { 0 }
2556 \cs_gset_protected:Npn
2557 \markdownRendererFencedCodeAttributeContextEnd
2558 {
2559 \markdownRendererFencedCodeAttributeContextEndPrototype
2560 }
2561 \seq_gput_right:Nn
2562 \g_@@_renderers_seq
2563 { fencedCodeAttributeContextEnd }
2564 \prop_gput:Nnn
2565 \g_@@_renderer_arities_prop
2566 { fencedCodeAttributeContextEnd }
2567 { 0 }
2568 \ExplSyntaxOff

```

### 2.2.5.19 Fenced Div Attribute Context Renderers

The following macros are only produced, when the `fencedDivs` 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.

```

2569 \ExplSyntaxOn
2570 \cs_gset_protected:Npn
2571 \markdownRendererFencedDivAttributeContextBegin
2572 {
2573 \markdownRendererFencedDivAttributeContextBeginPrototype
2574 }
2575 \seq_gput_right:Nn
2576 \g_@@_renderers_seq

```

```

2577 { fencedDivAttributeContextBegin }
2578 \prop_gput:Nnn
2579 \g_@@_renderer_arities_prop
2580 { fencedDivAttributeContextBegin }
2581 { 0 }
2582 \cs_gset_protected:Npn
2583 \markdownRendererFencedDivAttributeContextEnd
2584 {
2585 \markdownRendererFencedDivAttributeContextEndPrototype
2586 }
2587 \seq_gput_right:Nn
2588 \g_@@_renderers_seq
2589 { fencedDivAttributeContextEnd }
2590 \prop_gput:Nnn
2591 \g_@@_renderer_arities_prop
2592 { fencedDivAttributeContextEnd }
2593 { 0 }
2594 \ExplSyntaxOff

```

#### 2.2.5.20 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.

```

2595 \ExplSyntaxOn
2596 \cs_gset_protected:Npn
2597 \markdownRendererHeaderAttributeContextBegin
2598 {
2599 \markdownRendererHeaderAttributeContextBeginPrototype
2600 }
2601 \seq_gput_right:Nn
2602 \g_@@_renderers_seq
2603 { headerAttributeContextBegin }
2604 \prop_gput:Nnn
2605 \g_@@_renderer_arities_prop
2606 { headerAttributeContextBegin }
2607 { 0 }
2608 \cs_gset_protected:Npn
2609 \markdownRendererHeaderAttributeContextEnd
2610 {
2611 \markdownRendererHeaderAttributeContextEndPrototype
2612 }
2613 \seq_gput_right:Nn
2614 \g_@@_renderers_seq
2615 { headerAttributeContextEnd }

```

```

2616 \prop_gput:Nnn
2617 \g_@@_renderer_arities_prop
2618 { headerAttributeContextEnd }
2619 { 0 }
2620 \ExplSyntaxOff

```

### 2.2.5.21 Heading Renderers

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

```

2621 \ExplSyntaxOn
2622 \cs_gset_protected:Npn
2623 \markdownRendererHeadingOne
2624 {
2625 \markdownRendererHeadingOnePrototype
2626 }
2627 \seq_gput_right:Nn
2628 \g_@@_renderers_seq
2629 { headingOne }
2630 \prop_gput:Nnn
2631 \g_@@_renderer_arities_prop
2632 { headingOne }
2633 { 1 }
2634 \ExplSyntaxOff

```

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

```

2635 \ExplSyntaxOn
2636 \cs_gset_protected:Npn
2637 \markdownRendererHeadingTwo
2638 {
2639 \markdownRendererHeadingTwoPrototype
2640 }
2641 \seq_gput_right:Nn
2642 \g_@@_renderers_seq
2643 { headingTwo }
2644 \prop_gput:Nnn
2645 \g_@@_renderer_arities_prop
2646 { headingTwo }
2647 { 1 }
2648 \ExplSyntaxOff

```

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

```

2649 \ExplSyntaxOn
2650 \cs_gset_protected:Npn
2651 \markdownRendererHeadingThree

```

```

2652 {
2653 \markdownRendererHeadingThreePrototype
2654 }
2655 \seq_gput_right:Nn
2656 \g_@@_renderers_seq
2657 { headingThree }
2658 \prop_gput:Nnn
2659 \g_@@_renderer_arities_prop
2660 { headingThree }
2661 { 1 }
2662 \ExplSyntaxOff

```

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

```

2663 \ExplSyntaxOn
2664 \cs_gset_protected:Npn
2665 \markdownRendererHeadingFour
2666 {
2667 \markdownRendererHeadingFourPrototype
2668 }
2669 \seq_gput_right:Nn
2670 \g_@@_renderers_seq
2671 { headingFour }
2672 \prop_gput:Nnn
2673 \g_@@_renderer_arities_prop
2674 { headingFour }
2675 { 1 }
2676 \ExplSyntaxOff

```

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

```

2677 \ExplSyntaxOn
2678 \cs_gset_protected:Npn
2679 \markdownRendererHeadingFive
2680 {
2681 \markdownRendererHeadingFivePrototype
2682 }
2683 \seq_gput_right:Nn
2684 \g_@@_renderers_seq
2685 { headingFive }
2686 \prop_gput:Nnn
2687 \g_@@_renderer_arities_prop
2688 { headingFive }
2689 { 1 }
2690 \ExplSyntaxOff

```

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

```

2691 \ExplSyntaxOn
2692 \cs_gset_protected:Npn
2693 \markdownRendererHeadingSix
2694 {
2695 \markdownRendererHeadingSixPrototype
2696 }
2697 \seq_gput_right:Nn
2698 \g_@@_renderers_seq
2699 { headingSix }
2700 \prop_gput:Nnn
2701 \g_@@_renderer_arities_prop
2702 { headingSix }
2703 { 1 }
2704 \ExplSyntaxOff

```

### 2.2.5.22 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.

```

2705 \ExplSyntaxOn
2706 \cs_gset_protected:Npn
2707 \markdownRendererImage
2708 {
2709 \markdownRendererImagePrototype
2710 }
2711 \seq_gput_right:Nn
2712 \g_@@_renderers_seq
2713 { image }
2714 \prop_gput:Nnn
2715 \g_@@_renderer_arities_prop
2716 { image }
2717 { 4 }
2718 \ExplSyntaxOff

```

### 2.2.5.23 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.

```

2719 \ExplSyntaxOn
2720 \cs_gset_protected:Npn

```

```

2721 \markdownRendererImageAttributeContextBegin
2722 {
2723 \markdownRendererImageAttributeContextBeginPrototype
2724 }
2725 \seq_gput_right:Nn
2726 \g_@@_renderers_seq
2727 { imageAttributeContextBegin }
2728 \prop_gput:Nnn
2729 \g_@@_renderer_arities_prop
2730 { imageAttributeContextBegin }
2731 { 0 }
2732 \cs_gset_protected:Npn
2733 \markdownRendererImageAttributeContextEnd
2734 {
2735 \markdownRendererImageAttributeContextEndPrototype
2736 }
2737 \seq_gput_right:Nn
2738 \g_@@_renderers_seq
2739 { imageAttributeContextEnd }
2740 \prop_gput:Nnn
2741 \g_@@_renderer_arities_prop
2742 { imageAttributeContextEnd }
2743 { 0 }
2744 \ExplSyntaxOff

```

#### 2.2.5.24 Interblock Separator Renderers

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

```

2745 \ExplSyntaxOn
2746 \cs_gset_protected:Npn
2747 \markdownRendererInterblockSeparator
2748 {
2749 \markdownRendererInterblockSeparatorPrototype
2750 }
2751 \seq_gput_right:Nn
2752 \g_@@_renderers_seq
2753 { interblockSeparator }
2754 \prop_gput:Nnn
2755 \g_@@_renderer_arities_prop
2756 { interblockSeparator }
2757 { 0 }
2758 \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.

```

2759 \ExplSyntaxOn
2760 \cs_gset_protected:Npn
2761 \markdownRendererParagraphSeparator
2762 {
2763 \markdownRendererParagraphSeparatorPrototype
2764 }
2765 \seq_gput_right:Nn
2766 \g_@@_renderers_seq
2767 { paragraphSeparator }
2768 \prop_gput:Nnn
2769 \g_@@_renderer_arities_prop
2770 { paragraphSeparator }
2771 { 0 }
2772 \ExplSyntaxOff

```

#### 2.2.5.25 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.

```

2773 \ExplSyntaxOn
2774 \cs_gset_protected:Npn
2775 \markdownRendererLineBlockBegin
2776 {
2777 \markdownRendererLineBlockBeginPrototype
2778 }
2779 \seq_gput_right:Nn
2780 \g_@@_renderers_seq
2781 { lineBlockBegin }
2782 \prop_gput:Nnn
2783 \g_@@_renderer_arities_prop
2784 { lineBlockBegin }
2785 { 0 }
2786 \cs_gset_protected:Npn
2787 \markdownRendererLineBlockEnd
2788 {
2789 \markdownRendererLineBlockEndPrototype
2790 }
2791 \seq_gput_right:Nn
2792 \g_@@_renderers_seq
2793 { lineBlockEnd }

```

```

2794 \prop_gput:Nnn
2795 \g_@@_renderer_arities_prop
2796 { lineBlockEnd }
2797 { 0 }
2798 \ExplSyntaxOff

```

### 2.2.5.26 Line Break Renderers

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

```

2799 \ExplSyntaxOn
2800 \cs_gset_protected:Npn
2801 \markdownRendererSoftLineBreak
2802 {
2803 \markdownRendererSoftLineBreakPrototype
2804 }
2805 \seq_gput_right:Nn
2806 \g_@@_renderers_seq
2807 { softLineBreak }
2808 \prop_gput:Nnn
2809 \g_@@_renderer_arities_prop
2810 { softLineBreak }
2811 { 0 }
2812 \ExplSyntaxOff

```

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

```

2813 \ExplSyntaxOn
2814 \cs_gset_protected:Npn
2815 \markdownRendererHardLineBreak
2816 {
2817 \markdownRendererHardLineBreakPrototype
2818 }
2819 \seq_gput_right:Nn
2820 \g_@@_renderers_seq
2821 { hardLineBreak }
2822 \prop_gput:Nnn
2823 \g_@@_renderer_arities_prop
2824 { hardLineBreak }
2825 { 0 }
2826 \ExplSyntaxOff

```

### 2.2.5.27 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.

```

2827 \ExplSyntaxOn
2828 \cs_gset_protected:Npn
2829 \markdownRendererLink
2830 {
2831 \markdownRendererLinkPrototype
2832 }
2833 \seq_gput_right:Nn
2834 \g_@@_renderers_seq
2835 { link }
2836 \prop_gput:Nnn
2837 \g_@@_renderer_arities_prop
2838 { link }
2839 { 4 }
2840 \ExplSyntaxOff

```

### 2.2.5.28 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.

```

2841 \ExplSyntaxOn
2842 \cs_gset_protected:Npn
2843 \markdownRendererLinkAttributeContextBegin
2844 {
2845 \markdownRendererLinkAttributeContextBeginPrototype
2846 }
2847 \seq_gput_right:Nn
2848 \g_@@_renderers_seq
2849 { linkAttributeContextBegin }
2850 \prop_gput:Nnn
2851 \g_@@_renderer_arities_prop
2852 { linkAttributeContextBegin }
2853 { 0 }
2854 \cs_gset_protected:Npn
2855 \markdownRendererLinkAttributeContextEnd
2856 {
2857 \markdownRendererLinkAttributeContextEndPrototype
2858 }
2859 \seq_gput_right:Nn
2860 \g_@@_renderers_seq
2861 { linkAttributeContextEnd }
2862 \prop_gput:Nnn
2863 \g_@@_renderer_arities_prop
2864 { linkAttributeContextEnd }
2865 { 0 }

```

```
2866 \ExplSyntaxOff
```

### 2.2.5.29 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.

```
2867 \ExplSyntaxOn
2868 \cs_gset_protected:Npn
2869 \markdownRendererMark
2870 {
2871 \markdownRendererMarkPrototype
2872 }
2873 \seq_gput_right:Nn
2874 \g_@@_renderers_seq
2875 { mark }
2876 \prop_gput:Nnn
2877 \g_@@_renderer_arities_prop
2878 { mark }
2879 { 1 }
2880 \ExplSyntaxOff
```

### 2.2.5.30 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.

```
2881 \ExplSyntaxOn
2882 \cs_gset_protected:Npn
2883 \markdownRendererDocumentBegin
2884 {
2885 \markdownRendererDocumentBeginPrototype
2886 }
2887 \seq_gput_right:Nn
2888 \g_@@_renderers_seq
2889 { documentBegin }
2890 \prop_gput:Nnn
2891 \g_@@_renderer_arities_prop
2892 { documentBegin }
2893 { 0 }
2894 \cs_gset_protected:Npn
2895 \markdownRendererDocumentEnd
```

```

2896 {
2897 \markdownRendererDocumentEndPrototype
2898 }
2899 \seq_gput_right:Nn
2900 \g_@@_renderers_seq
2901 { documentEnd }
2902 \prop_gput:Nnn
2903 \g_@@_renderer_arities_prop
2904 { documentEnd }
2905 { 0 }
2906 \ExplSyntaxOff

```

### 2.2.5.31 Non-Breaking Space Renderer

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

```

2907 \ExplSyntaxOn
2908 \cs_gset_protected:Npn
2909 \markdownRendererNbsp
2910 {
2911 \markdownRendererNbspPrototype
2912 }
2913 \seq_gput_right:Nn
2914 \g_@@_renderers_seq
2915 { nbsp }
2916 \prop_gput:Nnn
2917 \g_@@_renderer_arities_prop
2918 { nbsp }
2919 { 0 }
2920 \ExplSyntaxOff

```

### 2.2.5.32 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.

```

2921 \def\markdownRendererNote{%
2922 \markdownRendererNotePrototype}%
2923 \ExplSyntaxOn
2924 \seq_gput_right:Nn
2925 \g_@@_renderers_seq
2926 { note }
2927 \prop_gput:Nnn
2928 \g_@@_renderer_arities_prop
2929 { note }
2930 { 1 }
2931 \ExplSyntaxOff

```

### 2.2.5.33 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.

```
2932 \ExplSyntaxOn
2933 \cs_gset_protected:Npn
2934 \markdownRendererOlBegin
2935 {
2936 \markdownRendererOlBeginPrototype
2937 }
2938 \seq_gput_right:Nn
2939 \g_@@_renderers_seq
2940 { olBegin }
2941 \prop_gput:Nnn
2942 \g_@@_renderer_arities_prop
2943 { olBegin }
2944 { 0 }
2945 \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.

```
2946 \ExplSyntaxOn
2947 \cs_gset_protected:Npn
2948 \markdownRendererOlBeginTight
2949 {
2950 \markdownRendererOlBeginTightPrototype
2951 }
2952 \seq_gput_right:Nn
2953 \g_@@_renderers_seq
2954 { olBeginTight }
2955 \prop_gput:Nnn
2956 \g_@@_renderer_arities_prop
2957 { olBeginTight }
2958 { 0 }
2959 \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`).

```

2960 \ExplSyntaxOn
2961 \cs_gset_protected:Npn
2962 \markdownRendererFancyOlBegin
2963 {
2964 \markdownRendererFancyOlBeginPrototype
2965 }
2966 \seq_gput_right:Nn
2967 \g_@@_renderers_seq
2968 { fancyOlBegin }
2969 \prop_gput:Nnn
2970 \g_@@_renderer_arities_prop
2971 { fancyOlBegin }
2972 { 2 }
2973 \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.

```

2974 \ExplSyntaxOn
2975 \cs_gset_protected:Npn
2976 \markdownRendererFancyOlBeginTight
2977 {
2978 \markdownRendererFancyOlBeginTightPrototype
2979 }
2980 \seq_gput_right:Nn
2981 \g_@@_renderers_seq
2982 { fancyOlBeginTight }
2983 \prop_gput:Nnn
2984 \g_@@_renderer_arities_prop
2985 { fancyOlBeginTight }
2986 { 2 }
2987 \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.

```

2988 \ExplSyntaxOn
2989 \cs_gset_protected:Npn
2990 \markdownRendererOlItem
2991 {
2992 \markdownRendererOlItemPrototype
2993 }
2994 \seq_gput_right:Nn

```

```

2995 \g_@@_renderers_seq
2996 { olItem }
2997 \prop_gput:Nnn
2998 \g_@@_renderer_arities_prop
2999 { olItem }
3000 { 0 }
3001 \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.

```

3002 \ExplSyntaxOn
3003 \cs_gset_protected:Npn
3004 \markdownRendererOlItemEnd
3005 {
3006 \markdownRendererOlItemEndPrototype
3007 }
3008 \seq_gput_right:Nn
3009 \g_@@_renderers_seq
3010 { olItemEnd }
3011 \prop_gput:Nnn
3012 \g_@@_renderer_arities_prop
3013 { olItemEnd }
3014 { 0 }
3015 \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.

```

3016 \ExplSyntaxOn
3017 \cs_gset_protected:Npn
3018 \markdownRendererOlItemWithNumber
3019 {
3020 \markdownRendererOlItemWithNumberPrototype
3021 }
3022 \seq_gput_right:Nn
3023 \g_@@_renderers_seq
3024 { olItemWithNumber }
3025 \prop_gput:Nnn
3026 \g_@@_renderer_arities_prop
3027 { olItemWithNumber }
3028 { 1 }
3029 \ExplSyntaxOff

```



The `\markdownRendererFancyO1Item` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is enabled. The macro receives no arguments.

```

3030 \ExplSyntaxOn
3031 \cs_gset_protected:Npn
3032 \markdownRendererFancyO1Item
3033 {
3034 \markdownRendererFancyO1ItemPrototype
3035 }
3036 \seq_gput_right:Nn
3037 \g_@@_renderers_seq
3038 { fancyO1Item }
3039 \prop_gput:Nnn
3040 \g_@@_renderer_arities_prop
3041 { fancyO1Item }
3042 { 0 }
3043 \ExplSyntaxOff

```

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

```

3044 \ExplSyntaxOn
3045 \cs_gset_protected:Npn
3046 \markdownRendererFancyO1ItemEnd
3047 {
3048 \markdownRendererFancyO1ItemEndPrototype
3049 }
3050 \seq_gput_right:Nn
3051 \g_@@_renderers_seq
3052 { fancyO1ItemEnd }
3053 \prop_gput:Nnn
3054 \g_@@_renderer_arities_prop
3055 { fancyO1ItemEnd }
3056 { 0 }
3057 \ExplSyntaxOff

```

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

```

3058 \ExplSyntaxOn
3059 \cs_gset_protected:Npn
3060 \markdownRendererFancyO1ItemWithNumber
3061 {
3062 \markdownRendererFancyO1ItemWithNumberPrototype
3063 }

```

```

3064 \seq_gput_right:Nn
3065 \g_@@_renderers_seq
3066 { fancyO1ItemWithNumber }
3067 \prop_gput:Nnn
3068 \g_@@_renderer_arities_prop
3069 { fancyO1ItemWithNumber }
3070 { 1 }
3071 \ExplSyntaxOff

```

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

```

3072 \ExplSyntaxOn
3073 \cs_gset_protected:Npn
3074 \markdownRendererO1End
3075 {
3076 \markdownRendererO1EndPrototype
3077 }
3078 \seq_gput_right:Nn
3079 \g_@@_renderers_seq
3080 { olEnd }
3081 \prop_gput:Nnn
3082 \g_@@_renderer_arities_prop
3083 { olEnd }
3084 { 0 }
3085 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

3086 \ExplSyntaxOn
3087 \cs_gset_protected:Npn
3088 \markdownRendererO1EndTight
3089 {
3090 \markdownRendererO1EndTightPrototype
3091 }
3092 \seq_gput_right:Nn
3093 \g_@@_renderers_seq
3094 { olEndTight }
3095 \prop_gput:Nnn
3096 \g_@@_renderer_arities_prop
3097 { olEndTight }
3098 { 0 }
3099 \ExplSyntaxOff

```

The `\markdownRendererFancy01End` 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.

```

3100 \ExplSyntaxOn
3101 \cs_gset_protected:Npn
3102 \markdownRendererFancy01End
3103 {
3104 \markdownRendererFancy01EndPrototype
3105 }
3106 \seq_gput_right:Nn
3107 \g_@@_renderers_seq
3108 { fancy01End }
3109 \prop_gput:Nnn
3110 \g_@@_renderer_arities_prop
3111 { fancy01End }
3112 { 0 }
3113 \ExplSyntaxOff

```

The `\markdownRendererFancy01EndTight` macro represents the end of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives no arguments.

```

3114 \ExplSyntaxOn
3115 \cs_gset_protected:Npn
3116 \markdownRendererFancy01EndTight
3117 {
3118 \markdownRendererFancy01EndTightPrototype
3119 }
3120 \seq_gput_right:Nn
3121 \g_@@_renderers_seq
3122 { fancy01EndTight }
3123 \prop_gput:Nnn
3124 \g_@@_renderer_arities_prop
3125 { fancy01EndTight }
3126 { 0 }
3127 \ExplSyntaxOff

```

#### 2.2.5.34 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.

```

3128 \ExplSyntaxOn

```

```

3129 \cs_gset_protected:Npn
3130 \markdownRendererInputRawInline
3131 {
3132 \markdownRendererInputRawInlinePrototype
3133 }
3134 \seq_gput_right:Nn
3135 \g_@@_renderers_seq
3136 { inputRawInline }
3137 \prop_gput:Nnn
3138 \g_@@_renderer_arities_prop
3139 { inputRawInline }
3140 { 2 }
3141 \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.

```

3142 \ExplSyntaxOn
3143 \cs_gset_protected:Npn
3144 \markdownRendererInputRawBlock
3145 {
3146 \markdownRendererInputRawBlockPrototype
3147 }
3148 \seq_gput_right:Nn
3149 \g_@@_renderers_seq
3150 { inputRawBlock }
3151 \prop_gput:Nnn
3152 \g_@@_renderer_arities_prop
3153 { inputRawBlock }
3154 { 2 }
3155 \ExplSyntaxOff

```

### 2.2.5.35 Section Renderers

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

```

3156 \ExplSyntaxOn
3157 \cs_gset_protected:Npn
3158 \markdownRendererSectionBegin
3159 {
3160 \markdownRendererSectionBeginPrototype
3161 }
3162 \seq_gput_right:Nn
3163 \g_@@_renderers_seq
3164 { sectionBegin }
3165 \prop_gput:Nnn

```

```

3166 \g_@@_renderer_arities_prop
3167 { sectionBegin }
3168 { 0 }
3169 \cs_gset_protected:Npn
3170 \markdownRendererSectionEnd
3171 {
3172 \markdownRendererSectionEndPrototype
3173 }
3174 \seq_gput_right:Nn
3175 \g_@@_renderers_seq
3176 { sectionEnd }
3177 \prop_gput:Nnn
3178 \g_@@_renderer_arities_prop
3179 { sectionEnd }
3180 { 0 }
3181 \ExplSyntaxOff

```

### 2.2.5.36 Replacement Character Renderers

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

```

3182 \ExplSyntaxOn
3183 \cs_gset_protected:Npn
3184 \markdownRendererReplacementCharacter
3185 {
3186 \markdownRendererReplacementCharacterPrototype
3187 }
3188 \seq_gput_right:Nn
3189 \g_@@_renderers_seq
3190 { replacementCharacter }
3191 \prop_gput:Nnn
3192 \g_@@_renderer_arities_prop
3193 { replacementCharacter }
3194 { 0 }
3195 \ExplSyntaxOff

```

### 2.2.5.37 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`.

```

3196 \ExplSyntaxOn
3197 \cs_gset_protected:Npn
3198 \markdownRendererLeftBrace
3199 {
3200 \markdownRendererLeftBracePrototype
3201 }

```

```

3202 \seq_gput_right:Nn
3203 \g_@@_renderers_seq
3204 { leftBrace }
3205 \prop_gput:Nnn
3206 \g_@@_renderer_arities_prop
3207 { leftBrace }
3208 { 0 }
3209 \cs_gset_protected:Npn
3210 \markdownRendererRightBrace
3211 {
3212 \markdownRendererRightBracePrototype
3213 }
3214 \seq_gput_right:Nn
3215 \g_@@_renderers_seq
3216 { rightBrace }
3217 \prop_gput:Nnn
3218 \g_@@_renderer_arities_prop
3219 { rightBrace }
3220 { 0 }
3221 \cs_gset_protected:Npn
3222 \markdownRendererDollarSign
3223 {
3224 \markdownRendererDollarSignPrototype
3225 }
3226 \seq_gput_right:Nn
3227 \g_@@_renderers_seq
3228 { dollarSign }
3229 \prop_gput:Nnn
3230 \g_@@_renderer_arities_prop
3231 { dollarSign }
3232 { 0 }
3233 \cs_gset_protected:Npn
3234 \markdownRendererPercentSign
3235 {
3236 \markdownRendererPercentSignPrototype
3237 }
3238 \seq_gput_right:Nn
3239 \g_@@_renderers_seq
3240 { percentSign }
3241 \prop_gput:Nnn
3242 \g_@@_renderer_arities_prop
3243 { percentSign }
3244 { 0 }
3245 \cs_gset_protected:Npn
3246 \markdownRendererAmpersand
3247 {
3248 \markdownRendererAmpersandPrototype

```

```

3249 }
3250 \seq_gput_right:Nn
3251 \g_@@_renderers_seq
3252 { ampersand }
3253 \prop_gput:Nnn
3254 \g_@@_renderer_arities_prop
3255 { ampersand }
3256 { 0 }
3257 \cs_gset_protected:Npn
3258 \markdownRendererUnderscore
3259 {
3260 \markdownRendererUnderscorePrototype
3261 }
3262 \seq_gput_right:Nn
3263 \g_@@_renderers_seq
3264 { underscore }
3265 \prop_gput:Nnn
3266 \g_@@_renderer_arities_prop
3267 { underscore }
3268 { 0 }
3269 \cs_gset_protected:Npn
3270 \markdownRendererHash
3271 {
3272 \markdownRendererHashPrototype
3273 }
3274 \seq_gput_right:Nn
3275 \g_@@_renderers_seq
3276 { hash }
3277 \prop_gput:Nnn
3278 \g_@@_renderer_arities_prop
3279 { hash }
3280 { 0 }
3281 \cs_gset_protected:Npn
3282 \markdownRendererCircumflex
3283 {
3284 \markdownRendererCircumflexPrototype
3285 }
3286 \seq_gput_right:Nn
3287 \g_@@_renderers_seq
3288 { circumflex }
3289 \prop_gput:Nnn
3290 \g_@@_renderer_arities_prop
3291 { circumflex }
3292 { 0 }
3293 \cs_gset_protected:Npn
3294 \markdownRendererBackslash
3295 {

```

```

3296 \markdownRendererBackslashPrototype
3297 }
3298 \seq_gput_right:Nn
3299 \g_@@_renderers_seq
3300 { backslash }
3301 \prop_gput:Nnn
3302 \g_@@_renderer_arities_prop
3303 { backslash }
3304 { 0 }
3305 \cs_gset_protected:Npn
3306 \markdownRendererTilde
3307 {
3308 \markdownRendererTildePrototype
3309 }
3310 \seq_gput_right:Nn
3311 \g_@@_renderers_seq
3312 { tilde }
3313 \prop_gput:Nnn
3314 \g_@@_renderer_arities_prop
3315 { tilde }
3316 { 0 }
3317 \cs_gset_protected:Npn
3318 \markdownRendererPipe
3319 {
3320 \markdownRendererPipePrototype
3321 }
3322 \seq_gput_right:Nn
3323 \g_@@_renderers_seq
3324 { pipe }
3325 \prop_gput:Nnn
3326 \g_@@_renderer_arities_prop
3327 { pipe }
3328 { 0 }
3329 \ExplSyntaxOff

```

#### 2.2.5.38 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.

```

3330 \ExplSyntaxOn
3331 \cs_gset_protected:Npn
3332 \markdownRendererStrikeThrough
3333 {
3334 \markdownRendererStrikeThroughPrototype
3335 }

```



```

3336 \seq_gput_right:Nn
3337 \g_@@_renderers_seq
3338 { strikeThrough }
3339 \prop_gput:Nnn
3340 \g_@@_renderer_arities_prop
3341 { strikeThrough }
3342 { 1 }
3343 \ExplSyntaxOff

```

### 2.2.5.39 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.

```

3344 \ExplSyntaxOn
3345 \cs_gset_protected:Npn
3346 \markdownRendererSubscript
3347 {
3348 \markdownRendererSubscriptPrototype
3349 }
3350 \seq_gput_right:Nn
3351 \g_@@_renderers_seq
3352 { subscript }
3353 \prop_gput:Nnn
3354 \g_@@_renderer_arities_prop
3355 { subscript }
3356 { 1 }

```

### 2.2.5.40 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.

```

3357 \cs_gset_protected:Npn
3358 \markdownRendererSuperscript
3359 {
3360 \markdownRendererSuperscriptPrototype
3361 }
3362 \seq_gput_right:Nn
3363 \g_@@_renderers_seq
3364 { superscript }
3365 \prop_gput:Nnn
3366 \g_@@_renderer_arities_prop
3367 { superscript }
3368 { 1 }
3369 \ExplSyntaxOff

```

#### 2.2.5.41 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.

```
3370 \ExplSyntaxOn
3371 \cs_gset_protected:Npn
3372 \markdownRendererTableAttributeContextBegin
3373 {
3374 \markdownRendererTableAttributeContextBeginPrototype
3375 }
3376 \seq_gput_right:Nn
3377 \g_@@_renderers_seq
3378 { tableAttributeContextBegin }
3379 \prop_gput:Nnn
3380 \g_@@_renderer_arities_prop
3381 { tableAttributeContextBegin }
3382 { 0 }
3383 \cs_gset_protected:Npn
3384 \markdownRendererTableAttributeContextEnd
3385 {
3386 \markdownRendererTableAttributeContextEndPrototype
3387 }
3388 \seq_gput_right:Nn
3389 \g_@@_renderers_seq
3390 { tableAttributeContextEnd }
3391 \prop_gput:Nnn
3392 \g_@@_renderer_arities_prop
3393 { tableAttributeContextEnd }
3394 { 0 }
3395 \ExplSyntaxOff
```

#### 2.2.5.42 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.

```

3396 \ExplSyntaxOn
3397 \cs_gset_protected:Npn
3398 \markdownRendererTable
3399 {
3400 \markdownRendererTablePrototype
3401 }
3402 \seq_gput_right:Nn
3403 \g_@@_renderers_seq
3404 { table }
3405 \prop_gput:Nnn
3406 \g_@@_renderer_arities_prop
3407 { table }
3408 { 3 }
3409 \ExplSyntaxOff

```

### 2.2.5.43 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.

```

3410 \ExplSyntaxOn
3411 \cs_gset_protected:Npn
3412 \markdownRendererInlineMath
3413 {
3414 \markdownRendererInlineMathPrototype
3415 }
3416 \seq_gput_right:Nn
3417 \g_@@_renderers_seq
3418 { inlineMath }
3419 \prop_gput:Nnn
3420 \g_@@_renderer_arities_prop
3421 { inlineMath }
3422 { 1 }
3423 \cs_gset_protected:Npn
3424 \markdownRendererDisplayMath
3425 {
3426 \markdownRendererDisplayMathPrototype
3427 }
3428 \seq_gput_right:Nn
3429 \g_@@_renderers_seq
3430 { displayMath }
3431 \prop_gput:Nnn
3432 \g_@@_renderer_arities_prop

```

```

3433 { displayMath }
3434 { 1 }
3435 \ExplSyntaxOff

```

#### 2.2.5.44 Thematic Break Renderer

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

```

3436 \ExplSyntaxOn
3437 \cs_gset_protected:Npn
3438 \markdownRendererThematicBreak
3439 {
3440 \markdownRendererThematicBreakPrototype
3441 }
3442 \seq_gput_right:Nn
3443 \g_@@_renderers_seq
3444 { thematicBreak }
3445 \prop_gput:Nnn
3446 \g_@@_renderer_arities_prop
3447 { thematicBreak }
3448 { 0 }
3449 \ExplSyntaxOff

```

#### 2.2.5.45 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.

```

3450 \ExplSyntaxOn
3451 \cs_gset_protected:Npn
3452 \markdownRendererTickedBox
3453 {
3454 \markdownRendererTickedBoxPrototype
3455 }
3456 \seq_gput_right:Nn
3457 \g_@@_renderers_seq
3458 { tickedBox }
3459 \prop_gput:Nnn
3460 \g_@@_renderer_arities_prop
3461 { tickedBox }
3462 { 0 }
3463 \cs_gset_protected:Npn
3464 \markdownRendererHalfTickedBox
3465 {

```

```

3466 \markdownRendererHalfTickedBoxPrototype
3467 }
3468 \seq_gput_right:Nn
3469 \g_@@_renderers_seq
3470 { halfTickedBox }
3471 \prop_gput:Nnn
3472 \g_@@_renderer_arities_prop
3473 { halfTickedBox }
3474 { 0 }
3475 \cs_gset_protected:Npn
3476 \markdownRendererUntickedBox
3477 {
3478 \markdownRendererUntickedBoxPrototype
3479 }
3480 \seq_gput_right:Nn
3481 \g_@@_renderers_seq
3482 { untickedBox }
3483 \prop_gput:Nnn
3484 \g_@@_renderer_arities_prop
3485 { untickedBox }
3486 { 0 }
3487 \ExplSyntaxOff

```

#### 2.2.5.46 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.

```

3488 \ExplSyntaxOn
3489 \cs_gset_protected:Npn
3490 \markdownRendererWarning
3491 {
3492 \markdownRendererWarningPrototype
3493 }
3494 \cs_gset_protected:Npn
3495 \markdownRendererError
3496 {

```

```

3497 \markdownRendererErrorPrototype
3498 }
3499 \seq_gput_right:Nn
3500 \g_@@_renderers_seq
3501 { warning }
3502 \prop_gput:Nnn
3503 \g_@@_renderer_arities_prop
3504 { warning }
3505 { 4 }
3506 \seq_gput_right:Nn
3507 \g_@@_renderers_seq
3508 { error }
3509 \prop_gput:Nnn
3510 \g_@@_renderer_arities_prop
3511 { error }
3512 { 4 }
3513 \ExplSyntaxOff

```

### 2.2.5.47 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.

```

3514 \ExplSyntaxOn
3515 \cs_gset_protected:Npn
3516 \markdownRendererJekyllDataBegin
3517 {
3518 \markdownRendererJekyllDataBeginPrototype
3519 }
3520 \seq_gput_right:Nn
3521 \g_@@_renderers_seq
3522 { jekyllDataBegin }
3523 \prop_gput:Nnn
3524 \g_@@_renderer_arities_prop
3525 { jekyllDataBegin }
3526 { 0 }
3527 \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.

```

3528 \ExplSyntaxOn
3529 \cs_gset_protected:Npn
3530 \markdownRendererJekyllDataEnd
3531 {
3532 \markdownRendererJekyllDataEndPrototype
3533 }

```

```

3534 \seq_gput_right:Nn
3535 \g_@@_renderers_seq
3536 { jekyllDataEnd }
3537 \prop_gput:Nnn
3538 \g_@@_renderer_arities_prop
3539 { jekyllDataEnd }
3540 { 0 }
3541 \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.

```

3542 \ExplSyntaxOn
3543 \cs_gset_protected:Npn
3544 \markdownRendererJekyllDataMappingBegin
3545 {
3546 \markdownRendererJekyllDataMappingBeginPrototype
3547 }
3548 \seq_gput_right:Nn
3549 \g_@@_renderers_seq
3550 { jekyllDataMappingBegin }
3551 \prop_gput:Nnn
3552 \g_@@_renderer_arities_prop
3553 { jekyllDataMappingBegin }
3554 { 2 }
3555 \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.

```

3556 \ExplSyntaxOn
3557 \cs_gset_protected:Npn
3558 \markdownRendererJekyllDataMappingEnd
3559 {
3560 \markdownRendererJekyllDataMappingEndPrototype
3561 }
3562 \seq_gput_right:Nn
3563 \g_@@_renderers_seq
3564 { jekyllDataMappingEnd }
3565 \prop_gput:Nnn
3566 \g_@@_renderer_arities_prop
3567 { jekyllDataMappingEnd }
3568 { 0 }
3569 \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.

```

3570 \ExplSyntaxOn
3571 \cs_gset_protected:Npn
3572 \markdownRendererJekyllDataSequenceBegin
3573 {
3574 \markdownRendererJekyllDataSequenceBeginPrototype
3575 }
3576 \seq_gput_right:Nn
3577 \g_@@_renderers_seq
3578 { jekyllDataSequenceBegin }
3579 \prop_gput:Nnn
3580 \g_@@_renderer_arities_prop
3581 { jekyllDataSequenceBegin }
3582 { 2 }
3583 \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.

```

3584 \ExplSyntaxOn
3585 \cs_gset_protected:Npn
3586 \markdownRendererJekyllDataSequenceEnd
3587 {
3588 \markdownRendererJekyllDataSequenceEndPrototype
3589 }
3590 \seq_gput_right:Nn
3591 \g_@@_renderers_seq
3592 { jekyllDataSequenceEnd }
3593 \prop_gput:Nnn
3594 \g_@@_renderer_arities_prop
3595 { jekyllDataSequenceEnd }
3596 { 0 }
3597 \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.

```

3598 \ExplSyntaxOn
3599 \cs_gset_protected:Npn

```



```

3600 \markdownRendererJekyllDataBoolean
3601 {
3602 \markdownRendererJekyllDataBooleanPrototype
3603 }
3604 \seq_gput_right:Nn
3605 \g_@@_renderers_seq
3606 { jekyllDataBoolean }
3607 \prop_gput:Nnn
3608 \g_@@_renderer_arities_prop
3609 { jekyllDataBoolean }
3610 { 2 }
3611 \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.

```

3612 \ExplSyntaxOn
3613 \cs_gset_protected:Npn
3614 \markdownRendererJekyllDataNumber
3615 {
3616 \markdownRendererJekyllDataNumberPrototype
3617 }
3618 \seq_gput_right:Nn
3619 \g_@@_renderers_seq
3620 { jekyllDataNumber }
3621 \prop_gput:Nnn
3622 \g_@@_renderer_arities_prop
3623 { jekyllDataNumber }
3624 { 2 }
3625 \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 T<sub>E</sub>X.

```

3626 \ExplSyntaxOn
3627 \cs_gset_protected:Npn
3628 \markdownRendererJekyllDataTypographicString
3629 {
3630 \markdownRendererJekyllDataTypographicStringPrototype
3631 }
3632 \cs_gset_protected:Npn
3633 \markdownRendererJekyllDataProgrammaticString
3634 {
3635 \markdownRendererJekyllDataProgrammaticStringPrototype
3636 }
3637 \seq_gput_right:Nn
3638 \g_@@_renderers_seq
3639 { jekyllDataTypographicString }
3640 \prop_gput:Nnn
3641 \g_@@_renderer_arities_prop
3642 { jekyllDataTypographicString }
3643 { 2 }
3644 \seq_gput_right:Nn
3645 \g_@@_renderers_seq
3646 { jekyllDataProgrammaticString }
3647 \prop_gput:Nnn
3648 \g_@@_renderer_arities_prop
3649 { jekyllDataProgrammaticString }
3650 { 2 }
3651 \ExplSyntaxOff

```

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

```

3652 \ExplSyntaxOn
3653 \cs_gset:Npn
3654 \markdownRendererJekyllDataTypographicString
3655 {
3656 \cs_if_exist:NTF
3657 \markdownRendererJekyllDataString
3658 {
3659 \@@_if_option:nTF
3660 { experimental }
3661 {
3662 \markdownError
3663 {
3664 The~jekyllDataString~renderer~has~been~deprecated,~

```

```

3665 to-be-removed-in~Markdown~4.0.0
3666 }
3667 }
3668 {
3669 \markdownWarning
3670 {
3671 The~jekyllDataString~renderer~has~been~deprecated,~
3672 to-be-removed-in~Markdown~4.0.0
3673 }
3674 \markdownRendererJekyllDataString
3675 }
3676 }
3677 {
3678 \cs_if_exist:NTF
3679 \markdownRendererJekyllDataStringPrototype
3680 {
3681 \@@_if_option:nTF
3682 { experimental }
3683 {
3684 \markdownError
3685 {
3686 The~jekyllDataString~renderer~prototype~
3687 has~been~deprecated,~
3688 to-be-removed-in~Markdown~4.0.0
3689 }
3690 }
3691 {
3692 \markdownWarning
3693 {
3694 The~jekyllDataString~renderer~prototype~
3695 has~been~deprecated,~
3696 to-be-removed-in~Markdown~4.0.0
3697 }
3698 \markdownRendererJekyllDataStringPrototype
3699 }
3700 }
3701 {
3702 \markdownRendererJekyllDataTypographicStringPrototype
3703 }
3704 }
3705 }
3706 \seq_gput_right:Nn
3707 \g_@@_renderers_seq
3708 { jekyllDataString }
3709 \prop_gput:Nnn
3710 \g_@@_renderer_arities_prop
3711 { jekyllDataString }

```

```

3712 { 2 }
3713 \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.

```

3714 \ExplSyntaxOn
3715 \cs_gset_protected:Npn
3716 \markdownRendererJekyllDataEmpty
3717 {
3718 \markdownRendererJekyllDataEmptyPrototype
3719 }
3720 \seq_gput_right:Nn
3721 \g_@@_renderers_seq
3722 { jekyllDataEmpty }
3723 \prop_gput:Nnn
3724 \g_@@_renderer_arities_prop
3725 { jekyllDataEmpty }
3726 { 1 }
3727 \ExplSyntaxOff

```

#### 2.2.5.48 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.

```

3728 \ExplSyntaxOn
3729 \cs_new_protected:Nn \@@_define_renderers:
3730 {
3731 \seq_map_inline:Nn
3732 \g_@@_renderers_seq
3733 {
3734 \@@_define_renderer:n
3735 { #1 }
3736 }
3737 }
3738 \cs_new_protected:Nn \@@_define_renderer:n
3739 {

```

```

3740 \@@_renderer_tl_to_csname:nN
3741 { #1 }
3742 \l_tmpa_tl
3743 \prop_get:NnN
3744 \g_@@_renderer_arities_prop
3745 { #1 }
3746 \l_tmpb_tl
3747 \@@_define_renderer:ncV
3748 { #1 }
3749 { \l_tmpa_tl }
3750 \l_tmpb_tl
3751 }
3752 \cs_new_protected:Nn \@@_renderer_tl_to_csname:nN
3753 {
3754 \tl_set:Nn
3755 \l_tmpa_tl
3756 { \str_uppercase:n { #1 } }
3757 \tl_set:Nx
3758 #2
3759 {
3760 markdownRenderer
3761 \tl_head:f { \l_tmpa_tl }
3762 \tl_tail:n { #1 }
3763 }
3764 }
3765 \tl_new:N
3766 \l_@@_renderer_definition_tl
3767 \bool_new:N
3768 \g_@@_prepending_renderer_bool
3769 \bool_new:N
3770 \g_@@_appending_renderer_bool
3771 \bool_new:N
3772 \g_@@_unprotected_renderer_bool
3773 \cs_new_protected:Nn \@@_define_renderer:nNn
3774 {
3775 \keys_define:nn
3776 { markdown/options/renderers }
3777 {
3778 #1 .value_required:n = { true },
3779 #1 .code:n = {
3780 \tl_set:Nn
3781 \l_@@_renderer_definition_tl
3782 { ##1 }
3783 \regex_replace_all:nnN
3784 { \cP\#0 }
3785 { #1 }
3786 \l_@@_renderer_definition_tl

```

```

3787 \bool_if:nT
3788 {
3789 \g_@@_prepending_renderer_bool ||
3790 \g_@@_appending_renderer_bool
3791 }
3792 {
3793 \@@_tl_set_from_cs:NNn
3794 \l_tmpa_tl
3795 #2
3796 { #3 }
3797 \bool_if:NTF
3798 \g_@@_prepending_renderer_bool
3799 {
3800 \tl_put_right:NV
3801 \l_@@_renderer_definition_tl
3802 \l_tmpa_tl
3803 }
3804 {
3805 \tl_put_left:NV
3806 \l_@@_renderer_definition_tl
3807 \l_tmpa_tl
3808 }
3809 }
3810 \bool_if:NTF
3811 \g_@@_unprotected_renderer_bool
3812 {
3813 \tl_set:Nn
3814 \l_tmpa_tl
3815 { \cs_set:Npn }
3816 }
3817 {
3818 \tl_set:Nn
3819 \l_tmpa_tl
3820 { \cs_set_protected:Npn }
3821 }
3822 \exp_last_unbraced:NNV
3823 \cs_generate_from_arg_count:NNnV
3824 #2
3825 \l_tmpa_tl
3826 { #3 }
3827 \l_@@_renderer_definition_tl
3828 },
3829 }

```

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.

```

3830 \str_if_eq:nnT
3831 { #1 }
3832 { jekyllDataString }
3833 {
3834 \cs_undefine:N
3835 #2
3836 }
3837 }

```

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.

```

3838 \cs_new_protected:Nn
3839 \@@_tl_set_from_cs:NNn
3840 {
3841 \tl_set:Nn
3842 \l_tmpa_tl
3843 { #2 }
3844 \int_step_inline:nn
3845 { #3 }
3846 {
3847 \exp_args:Nnc
3848 \tl_put_right:Nn
3849 \l_tmpa_tl
3850 { @@_tl_set_from_cs_parameter_ ##1 }
3851 }
3852 \exp_args:NNV
3853 \tl_set:No
3854 \l_tmpb_tl
3855 \l_tmpa_tl
3856 \regex_replace_all:nnN
3857 { \cP. }
3858 { \0\0 }
3859 \l_tmpb_tl
3860 \int_step_inline:nn
3861 { #3 }
3862 {
3863 \regex_replace_all:nnN
3864 { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3865 { \cP\# ##1 }
3866 \l_tmpb_tl
3867 }
3868 \tl_set:NV
3869 #1
3870 \l_tmpb_tl
3871 }

```

```

3872 \cs_generate_variant:Nn
3873 \@@_define_renderer:nNn
3874 { ncV }
3875 \cs_generate_variant:Nn
3876 \cs_generate_from_arg_count:NNnn
3877 { NNnV }
3878 \cs_generate_variant:Nn
3879 \tl_put_left:Nn
3880 { Nv }
3881 \keys_define:nn
3882 { markdown/options }
3883 {
3884 renderers .value_required:n = { true },
3885 renderers .code:n = {
3886 \bool_gset_false:N
3887 \g_@@_unprotected_renderer_bool
3888 \keys_set:nn
3889 { markdown/options/renderers }
3890 { #1 }
3891 },
3892 unprotectedRenderers .value_required:n = { true },
3893 unprotectedRenderers .code:n = {
3894 \bool_gset_true:N
3895 \g_@@_unprotected_renderer_bool
3896 \keys_set:nn
3897 { markdown/options/renderers }
3898 { #1 }
3899 },
3900 }

```

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.
 }
}

```

```

3901 \tl_new:N
3902 \l_@@_renderer_glob_definition_tl
3903 \seq_new:N
3904 \l_@@_renderer_glob_results_seq
3905 \regex_const:Nn
3906 \c_@@_prepending_key_regex
3907 { \^$ }

```



```

3908 \regex_const:Nn
3909 \c_@@_appending_key_regex
3910 { [\${+}]$ }
3911 \regex_const:Nn
3912 \c_@@_prepending_or_appending_key_regex
3913 { [^\${+}]$ }
3914 \keys_define:nn
3915 { markdown/options/renderers }
3916 {
3917 unknown .value_required:n = { true },
3918 unknown .code:n = {

```

Besides defining renderers at once, we can also define them incrementally using the prepending ( $\text{\textasciitilde{=}}$ ) and appending ( $\text{\$=}$  or  $\text{+=}$ ) operators. This can be especially useful in defining rules for processing different HTML class names and identifiers:

```

\markdownSetup{
 renderers = {
 % Start with empty renderers.
 headerAttributeContextBegin = \begingroup,
 headerAttributeContextEnd = \endgroup,
 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 += {...},
 },
 }
 },
 },
}

```

```

3919 % TODO: Use ` \regex_if_match` in TeX Live 2025.
3920 \regex_match:NVT

```

```

3921 \c_@@_prepending_key_regex
3922 \l_keys_key_str
3923 {
3924 \bool_gset_true:N
3925 \g_@@_prepending_renderer_bool
3926 }
3927 % TODO: Use `regex_if_match` in TeX Live 2025.
3928 \regex_match:NVT
3929 \c_@@_appending_key_regex
3930 \l_keys_key_str
3931 {
3932 \bool_gset_true:N
3933 \g_@@_appending_renderer_bool
3934 }
3935 % TODO: Use `regex_if_match` in TeX Live 2025.
3936 \regex_match:NVTF
3937 \c_@@_prepending_or_appending_key_regex
3938 \l_keys_key_str
3939 {
3940 \tl_set:NV
3941 \l_tmpa_tl
3942 \l_keys_key_str
3943 \regex_replace_once:NnN
3944 \c_@@_prepending_or_appending_key_regex
3945 { }
3946 \l_tmpa_tl
3947 \tl_set:Nx
3948 \l_tmpb_tl
3949 { { \l_tmpa_tl } = }
3950 \tl_put_right:Nn
3951 \l_tmpb_tl
3952 { { #1 } }
3953 \keys_set:nV
3954 { markdown/options/renderers }
3955 \l_tmpb_tl
3956 \bool_gset_false:N
3957 \g_@@_prepending_renderer_bool
3958 \bool_gset_false:N
3959 \g_@@_appending_renderer_bool
3960 }

```

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

```

\markdownSetup{
 renderers = {
 heading* = {{\bf #1}}, % Render headings using the bold face.

```

```

 jekyllData(String|Number) = {% % Render YAML string and numbers
 {\it #2}% % using italics.
 },
 }
}

```

Wildcards and enumerations can be combined:

```

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

```

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

```

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

```

```

3961 {
3962 \@@_glob_seq:VnN
3963 \l_keys_key_str
3964 { g_@@_renderers_seq }
3965 \l_@@_renderer_glob_results_seq
3966 \seq_if_empty:NTF
3967 \l_@@_renderer_glob_results_seq
3968 {
3969 \msg_error:nnV
3970 { markdown }
3971 { undefined-renderer }
3972 \l_keys_key_str
3973 }
3974 {
3975 \tl_set:Nn
3976 \l_@@_renderer_glob_definition_tl
3977 { \exp_not:n { #1 } }
3978 \seq_map_inline:Nn
3979 \l_@@_renderer_glob_results_seq
3980 {
3981 \tl_set:Nn
3982 \l_tmpa_tl

```

```

3983 { { ##1 } = }
3984 \tl_put_right:Nx
3985 \l_tmpa_tl
3986 { { \l_@@_renderer_glob_definition_tl } }
3987 \keys_set:nV
3988 { markdown/options/renderers }
3989 \l_tmpa_tl
3990 }
3991 }
3992 }
3993 },
3994 }
3995 \msg_new:nnn
3996 { markdown }
3997 { undefined-renderer }
3998 {
3999 Renderer~#1~is~undefined.
4000 }
4001 \cs_generate_variant:Nn
4002 \@@_glob_seq:nnN
4003 { VnN }
4004 \cs_generate_variant:Nn
4005 \cs_generate_from_arg_count:NNnn
4006 { cNVV }
4007 \cs_generate_variant:Nn
4008 \msg_error:nnn
4009 { nnV }
4010 \prg_generate_conditional_variant:Nnn
4011 % TODO: Use `regex_if_match` in TeX Live 2025.
4012 \regex_match:Nn % noqa: w202
4013 { NV }
4014 { T, TF }
4015 \prop_new:N
4016 \g_@@_glob_cache_prop
4017 \tl_new:N
4018 \l_@@_current_glob_tl
4019 \cs_new_protected:Nn
4020 \@@_glob_seq:nnN
4021 {
4022 \tl_set:Nn
4023 \l_@@_current_glob_tl
4024 { ^ #1 $ }
4025 \prop_get:NeNTF
4026 \g_@@_glob_cache_prop
4027 { #2 / \l_@@_current_glob_tl }
4028 \l_tmpa_clist
4029 {

```

```

4030 \seq_set_from_clist:NN
4031 #3
4032 \l_tmpa_clist
4033 }
4034 {
4035 \seq_clear:N
4036 #3
4037 \regex_replace_all:nnN
4038 { * }
4039 { .* }
4040 \l_@@_current_glob_tl
4041 \regex_set:NV
4042 \l_tmpa_regex
4043 \l_@@_current_glob_tl
4044 \seq_map_inline:cn
4045 { #2 }
4046 {
4047 % TODO: Use \regex_if_match in TeX Live 2025.
4048 \regex_match:NnT % noqa: w202
4049 \l_tmpa_regex
4050 { ##1 }
4051 {
4052 \seq_put_right:Nn
4053 #3
4054 { ##1 }
4055 }
4056 }
4057 \clist_set_from_seq:NN
4058 \l_tmpa_clist
4059 #3
4060 \prop_gput:NeV
4061 \g_@@_glob_cache_prop
4062 { #2 / \l_@@_current_glob_tl }
4063 \l_tmpa_clist
4064 }
4065 }
4066 \cs_generate_variant:Nn
4067 \regex_set:Nn
4068 { NV }
4069 \cs_generate_variant:Nn
4070 \prop_gput:Nnn
4071 { NeV }

```

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

```

4072 \str_if_eq:VVT

```

```

4073 \c_@@_top_layer_tl
4074 \c_@@_option_layer_plain_tex_tl
4075 {
4076 \@@_define_renderers:
4077 }
4078 \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`.

```

4079 \ExplSyntaxOn
4080 \keys_define:nn
4081 { markdown/jekyllData }
4082 { }
4083 \ExplSyntaxOff

```

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

```

4084 \ExplSyntaxOn
4085 \@@_with_various_cases:nn
4086 { jekyllDataRenderers }
4087 {
4088 \keys_define:nn
4089 { markdown/options }
4090 {
4091 #1 .value_required:n = { true },
4092 #1 .code:n = {
4093 \tl_set:Nn
4094 \l_tmpa_tl
4095 { ##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.

```

4096 \tl_replace_all:NnV
4097 \l_tmpa_tl
4098 { / }
4099 \c_backslash_str
4100 \keys_set:nV
4101 { markdown/options/jekyll-data-renderers }
4102 \l_tmpa_tl
4103 },
4104 }

```

```

4105 }
4106 \keys_define:nn
4107 { markdown/options/jekyll-data-renderers }
4108 {
4109 unknown .value_required:n = { true },
4110 unknown .code:n = {
4111 \tl_set_eq:NN
4112 \l_tmpa_tl
4113 \l_keys_key_str
4114 \tl_replace_all:Nvn
4115 \l_tmpa_tl
4116 \c_backslash_str
4117 { / }
4118 \tl_put_right:Nn
4119 \l_tmpa_tl
4120 {
4121 .code:n = { #1 }
4122 }
4123 \keys_define:nV
4124 { markdown/jekyllData }
4125 \l_tmpa_tl
4126 }
4127 }
4128 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [14] can be used to route the processing of all YAML metadata in the current  $\TeX$  group to the key-values from  $\langle module \rangle$ .

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

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

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

```

4129 \ExplSyntaxOn
4130 \cs_new_protected:Nn \@@_define_renderer_prototypes:
4131 {
4132 \seq_map_inline:Nn

```

```

4133 \g_@@_renderers_seq
4134 {
4135 \@@_define_renderer_prototype:n
4136 { ##1 }
4137 }
4138 }
4139 \cs_new_protected:Nn \@@_define_renderer_prototype:n
4140 {
4141 \@@_renderer_prototype_tl_to_csname:nN
4142 { #1 }
4143 \l_tmpa_tl
4144 \prop_get:NnN
4145 \g_@@_renderer_arities_prop
4146 { #1 }
4147 \l_tmpb_tl
4148 \@@_define_renderer_prototype:ncV
4149 { #1 }
4150 { \l_tmpa_tl }
4151 \l_tmpb_tl
4152 }
4153 \cs_new_protected:Nn \@@_renderer_prototype_tl_to_csname:nN
4154 {
4155 \tl_set:Nn
4156 \l_tmpa_tl
4157 { \str_uppercase:n { #1 } }
4158 \tl_set:Nx
4159 #2
4160 {
4161 markdownRenderer
4162 \tl_head:f { \l_tmpa_tl }
4163 \tl_tail:n { #1 }
4164 Prototype
4165 }
4166 }
4167 \tl_new:N
4168 \l_@@_renderer_prototype_definition_tl
4169 \bool_new:N
4170 \g_@@_prepending_renderer_prototype_bool
4171 \bool_new:N
4172 \g_@@_appending_renderer_prototype_bool
4173 \bool_new:N
4174 \g_@@_unprotected_renderer_prototype_bool
4175 \cs_new_protected:Nn \@@_define_renderer_prototype:nNn
4176 {
4177 \keys_define:nn
4178 { markdown/options/renderer-prototypes }
4179 {

```



```

4180 #1 .value_required:n = { true },
4181 #1 .code:n = {
4182 \tl_set:Nn
4183 \l_@@_renderer_prototype_definition_tl
4184 { ##1 }
4185 \regex_replace_all:nnN
4186 { \cP\#0 }
4187 { #1 }
4188 \l_@@_renderer_prototype_definition_tl
4189 \bool_if:nT
4190 {
4191 \g_@@_prepending_renderer_prototype_bool ||
4192 \g_@@_appending_renderer_prototype_bool
4193 }
4194 {
4195 \@@_tl_set_from_cs:NNn
4196 \l_tmpa_tl
4197 #2
4198 { #3 }
4199 \bool_if:NTF
4200 \g_@@_prepending_renderer_prototype_bool
4201 {
4202 \tl_put_right:NV
4203 \l_@@_renderer_prototype_definition_tl
4204 \l_tmpa_tl
4205 }
4206 {
4207 \tl_put_left:NV
4208 \l_@@_renderer_prototype_definition_tl
4209 \l_tmpa_tl
4210 }
4211 }
4212 \bool_if:NTF
4213 \g_@@_unprotected_renderer_prototype_bool
4214 {
4215 \tl_set:Nn
4216 \l_tmpa_tl
4217 { \cs_set:Npn }
4218 }
4219 {
4220 \tl_set:Nn
4221 \l_tmpa_tl
4222 { \cs_set_protected:Npn }
4223 }
4224 \exp_last_unbraced:NNV
4225 \cs_generate_from_arg_count:NNnV
4226 #2

```

```

4227 \l_tmpa_tl
4228 { #3 }
4229 \l_@@_renderer_prototype_definition_tl
4230 },
4231 }

```

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.

```

4232 \str_if_eq:nnF
4233 { #1 }
4234 { jekyllDataString }
4235 {
4236 \cs_if_free:NT
4237 #2
4238 {
4239 \cs_generate_from_arg_count:NNnn
4240 #2
4241 \cs_gset_protected:Npn
4242 { #3 }
4243 { }
4244 }
4245 }
4246 }
4247 \cs_generate_variant:Nn
4248 \@@_define_renderer_prototype:nNn
4249 { ncV }

```

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

```

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

```

```

4250 \keys_define:nn
4251 { markdown/options/renderer-prototypes }
4252 {
4253 unknown .value_required:n = { true },
4254 unknown .code:n = {

```

Besides defining renderer prototypes at once, we can also define them incrementally using the prepending ( $\wedge=$ ) and appending ( $\$=$  or  $\ +=$ ) operators. This can be especially useful in defining rules for processing different HTML class names and identifiers:

```
\markdownSetup{
 rendererPrototypes = {
 % Start with empty renderer prototypes.
 headerAttributeContextBegin = \begingroup,
 headerAttributeContextEnd = \endgroup,
 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 += {...},
 },
 },
 },
 },
}
```

```
4255 % TODO: Use `regex_if_match` in TeX Live 2025.
4256 \regex_match:NVT
4257 \c_@@_prepending_key_regex
4258 \l_keys_key_str
4259 {
4260 \bool_gset_true:N
4261 \g_@@_prepending_renderer_prototype_bool
4262 }
4263 % TODO: Use `regex_if_match` in TeX Live 2025.
4264 \regex_match:NVT
4265 \c_@@_appending_key_regex
4266 \l_keys_key_str
4267 {
4268 \bool_gset_true:N
```

```

4269 \g_@@_appending_renderer_prototype_bool
4270 }
4271 % TODO: Use ` \regex_if_match` in TeX Live 2025.
4272 \regex_match:NVTF
4273 \c_@@_prepending_or_appending_key_regex
4274 \l_keys_key_str
4275 {
4276 \tl_set:NV
4277 \l_tmpa_tl
4278 \l_keys_key_str
4279 \regex_replace_once:NnN
4280 \c_@@_prepending_or_appending_key_regex
4281 { }
4282 \l_tmpa_tl
4283 \tl_set:Nx
4284 \l_tmpb_tl
4285 { { \l_tmpa_tl } = }
4286 \tl_put_right:Nn
4287 \l_tmpb_tl
4288 { { #1 } }
4289 \keys_set:nV
4290 { markdown/options/renderer-prototypes }
4291 \l_tmpb_tl
4292 \bool_gset_false:N
4293 \g_@@_prepending_renderer_prototype_bool
4294 \bool_gset_false:N
4295 \g_@@_appending_renderer_prototype_bool
4296 }

```

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.
}

```

```

4297 {
4298 \@@_glob_seq:VnN
4299 \l_keys_key_str
4300 { g_@@_renderers_seq }
4301 \l_@@_renderer_glob_results_seq
4302 \seq_if_empty:NTF
4303 \l_@@_renderer_glob_results_seq
4304 {
4305 \msg_error:nnV
4306 { markdown }
4307 { undefined-renderer-prototype }
4308 \l_keys_key_str
4309 }
4310 {
4311 \tl_set:Nn
4312 \l_@@_renderer_glob_definition_tl
4313 { \exp_not:n { #1 } }
4314 \seq_map_inline:Nn
4315 \l_@@_renderer_glob_results_seq
4316 {
4317 \tl_set:Nn
4318 \l_tmpa_tl
4319 { { ##1 } = }
4320 \tl_put_right:Nx
4321 \l_tmpa_tl
4322 { { \l_@@_renderer_glob_definition_tl } }
4323 \keys_set:nV
4324 { markdown/options/renderer-prototypes }
4325 \l_tmpa_tl
4326 }
4327 }
4328 }

```

```

4329 },
4330 }
4331 \msg_new:nnn
4332 { markdown }
4333 { undefined-renderer-prototype }
4334 {
4335 Renderer~prototype~#1~is~undefined.
4336 }
4337 \@@_with_various_cases:nn
4338 { rendererPrototypes }
4339 {
4340 \keys_define:nn
4341 { markdown/options }
4342 {
4343 #1 .value_required:n = { true },
4344 #1 .code:n = {
4345 \bool_gset_false:N
4346 \g_@@_unprotected_renderer_prototype_bool
4347 \keys_set:nn
4348 { markdown/options/renderer-prototypes }
4349 { ##1 }
4350 },
4351 }
4352 }
4353 \@@_with_various_cases:nn
4354 { unprotectedRendererPrototypes }
4355 {
4356 \keys_define:nn
4357 { markdown/options }
4358 {
4359 #1 .value_required:n = { true },
4360 #1 .code:n = {
4361 \bool_gset_true:N
4362 \g_@@_unprotected_renderer_prototype_bool
4363 \keys_set:nn
4364 { markdown/options/renderer-prototypes }
4365 { ##1 }
4366 },
4367 }
4368 }

```

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.

```

4369 \str_if_eq:VVT
4370 \c_@@_top_layer_tl
4371 \c_@@_option_layer_plain_tex_tl

```

```

4372 {
4373 \@@_define_renderer_prototypes:
4374 }
4375 \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*.

```

4376 \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.

```

4377 \let\markdownReadAndConvert\relax
4378 \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*.

```

4379 \catcode`\|=0\catcode`\=12%
4380 |gdef|markdownBegin{%
4381 |markdownReadAndConvert{\markdownEnd}%
4382 {\markdownEnd}}%
4383 |gdef|yamlBegin{%
4384 |begingroup
4385 |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
4386 |markdownReadAndConvert{\yamlEnd}%

```

```

4387 {\yamlEnd}}%
4388 |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.

```

4389 \ExplSyntaxOn
4390 \keys_define:nn
4391 { markdown/options }
4392 {
4393 code .value_required:n = { true },
4394 code .code:n = { #1 },
4395 }
4396 \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.

```

4397 \ExplSyntaxOn
4398 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
4399 \cs_generate_variant:Nn
4400 \tl_const:Nn
4401 { NV }
4402 \tl_if_exist:NF
4403 \c_@@_top_layer_tl
4404 {
4405 \tl_const:NV
4406 \c_@@_top_layer_tl
4407 \c_@@_option_layer_latex_tl
4408 }
4409 \ExplSyntaxOff
4410 \input markdown/markdown

```

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

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



where  $\langle options \rangle$  are the  $\text{\LaTeX}$  interface options (see Section 2.3.3). Note that  $\langle options \rangle$  inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.48) 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 and YAML

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.

```
4411 \newenvironment{markdown}\relax\relax
4412 \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 TeX'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` LaTeX 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` LaTeX environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```
4413 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts LaTeX 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 LaTeX code showcases the usage of the `yaml` environment:

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

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

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

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

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

### 2.3.1.2 Typesetting Markdown and yaml from external documents

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

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

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

```
\end{document}
```

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain  $\text{\TeX}$ . 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 2.2.5.17. 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 2.2.5.42.

### 2.3.3 Options

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

$\LaTeX$  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.

```
4414 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
4415 \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.

```
4416 \ExplSyntaxOn
4417 \str_if_eq:VVT
4418 \c_@@_top_layer_tl
4419 \c_@@_option_layer_latex_tl
4420 {
4421 \@@define_option_commands_and_keyvals:
4422 \@@define_renderers:
4423 \@@define_renderer_prototypes:
4424 }
4425 \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}
```

```
4426 \newif\ifmarkdownLaTeXLoaded
4427 \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.

```

4428 \ExplSyntaxOn
4429 \prop_new:N
4430 \g_@@_latex_built_in_themes_prop
4431 \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.

```

4432 \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).

```

4433 \ExplSyntaxOn
4434 \str_if_eq:VVT
4435 \c_@@_top_layer_tl
4436 \c_@@_option_layer_latex_tl
4437 {
4438 \use:c
4439 { ExplSyntaxOff }
4440 \AtEndOfPackage
4441 {
4442 \@@_if_option:nF
4443 { noDefaults }
4444 {
4445 \@@_if_option:nTF
4446 { experimental }
4447 {
4448 \@@_setup:n
4449 { theme = witiko/markdown/defaults@experimental }
4450 }
4451 {
4452 \@@_setup:n
4453 { theme = witiko/markdown/defaults }
4454 }
4455 }
4456 }
4457 \use:c
4458 { ExplSyntaxOn }
4459 }
4460 \ExplSyntaxOff

```

See Section 3.3.2 for implementation details of the built-in  $\text{\LaTeX}$  themes.



### 2.3.5 Snippets

In Section 2.2.4, we described the concept of snippets.

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

**witiko/glossaries/import-acronyms** Imports all acronyms from the L<sup>A</sup>T<sub>E</sub>X package glossaries, appends them to the **acronyms** option and updates the corresponding renderers to recognize these acronyms in running text and link them to the corresponding glossary entries.

This snippet is provided by the theme **witiko/glossaries@v1**.

See Section 3.3.3 for implementation details of the built-in L<sup>A</sup>T<sub>E</sub>X snippets.

## 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.

```
4461 \ExplSyntaxOn
4462 \tl_const:Nn \c_@@_option_layer_context_tl { context }
4463 \cs_generate_variant:Nn
4464 \tl_const:Nn
4465 { NV }
4466 \tl_if_exist:NF
4467 \c_@@_top_layer_tl
4468 {
4469 \tl_const:NV
4470 \c_@@_top_layer_tl
4471 \c_@@_option_layer_context_tl
4472 }
4473 \ExplSyntaxOff
```

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

```
4474 \writestatus{loading}{ConTEXt User Module / markdown}%
4475 \startmodule[markdown]
4476 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
4477 \do\#\do\^\do_do\%do\~}%
4478 \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 T<sub>E</sub>X interface.

```
4479 \let\startmarkdown\relax
4480 \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 ConT<sub>E</sub>Xt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

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

The `\startyaml` and `\stopyaml` macros are aliases for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain T<sub>E</sub>X interface.

```
4481 \let\startyaml\relax
4482 \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 ConT<sub>E</sub>Xt 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 T<sub>E</sub>X interface.

```
4483 \let\inputmarkdown\relax
```

Furthermore, the `\inputmarkdown` macro also accepts ConT<sub>E</sub>Xt interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example ConT<sub>E</sub>Xt 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.

```
4484 \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.yaml}
\stoptext
```

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

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

## 2.4.2 Options

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

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

The Con $\text{\TeX}$ t 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}`.

```
4485 \ExplSyntaxOn
4486 \cs_new_protected:Npn
4487 \setupmarkdown
4488 [#1]
4489 {
4490 \@@_setup:n
4491 { #1 }
4492 }
```

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

```

4493 \cs_gset_eq:Nn
4494 \setupyaml
4495 \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.

```

4496 \cs_new_protected:Nn \@@_caseless:N
4497 {
4498 \regex_replace_all:nnN
4499 { ([a-z])([A-Z]) }
4500 { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
4501 #1
4502 \tl_set:Nx
4503 #1
4504 { #1 }
4505 }
4506 \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.

```

4507 \str_if_eq:VVT
4508 \c_@@_top_layer_tl
4509 \c_@@_option_layer_context_tl
4510 {
4511 \@@_define_option_commands_and_keyvals:
4512 \@@_define_renderers:
4513 \@@_define_renderer_prototypes:
4514 }

```

### 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.

```
4515 \prop_new:N
4516 \g_@@_context_built_in_themes_prop
4517 \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.

```
4518 \startmodule[markdownthemewitiko_markdown_defaults]
4519 \unprotect
```

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.

```
4520 local upper, format, length =
4521 string.upper, string.format, string.len
4522 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
4523 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
4524 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.

```
4525 local early_warnings = {}
4526 local unicode_data = require("markdown-unicode-data")
4527 if metadata.version ~= unicode_data.metadata.version then
4528 table.insert(
4529 early_warnings,
4530 "markdown.lua " .. metadata.version .. " used with " ..
4531 "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
4532)
4533 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).

```
4534 local function depth_first_search(node, path, visit, leave)
4535 assert(node._type == "intermediate")
4536 visit(node, path)
4537 for label, child in pairs(node) do
4538 if label == "_type" then
4539 goto continue
4540 end
4541 if child._type ~= "intermediate" then
4542 goto continue
4543 end
4544 depth_first_search(child, path .. label, visit, leave)
```

```

4545 ::continue::
4546 end
4547 for label, child in pairs(node) do
4548 if label == "_type" then
4549 goto continue
4550 end
4551 if child._type ~= "leaf" then
4552 goto continue
4553 end
4554 visit(child, path)
4555 ::continue::
4556 end
4557 leave(node, path)
4558 end

```

This function is defined in the file [markdown-parser.lua](#) as well, so that it can be used to define the [extensions.acronyms](#) built-in syntax extension. The function [serialize\\_byte\\_parser](#) produces the Lua code for a PEG parser that matches a single byte.

```

4559 local function serialize_byte_parser(byte)
4560 if byte == '"' then
4561 return "P('' .. byte .. '')"
4562 elseif byte == "\\" then
4563 return "P("\\\\")"
4564 else
4565 return "P('' .. byte .. '')"
4566 end
4567 end

```

The function [serialize\\_byte\\_range\\_parser](#) produces the Lua code for a PEG parser that matches a contiguous range of bytes.

```

4568 local function serialize_byte_range_parser(start_byte, end_byte)
4569 assert(start_byte <= end_byte)
4570 if start_byte == "\\" then
4571 start_byte = "\\\\"
4572 end
4573 if end_byte == "\\" then
4574 end_byte = "\\\\"
4575 end
4576 if start_byte == '"' or end_byte == '"' then
4577 return "R('' .. start_byte .. end_byte .. '')"
4578 else
4579 return "R('' .. start_byte .. end_byte .. '')"
4580 end
4581 end

```

The function [serialize\\_replacement](#) produces the Lua code for a string literal with one or more UTF-8-encoded Unicode characters.



```

4582 local function serialize_replacement(codepoints)
4583 assert(#codepoints > 0)
4584 local buffer = {' '}
4585 for _, codepoint in ipairs(codepoints) do
4586 local code = utf8.char(codepoint)
4587 for i = 1, #code do
4588 local byte = code:sub(i, i)
4589 if byte == ' ' then
4590 table.insert(buffer, '\\\\')
4591 elseif byte == "\\\" then
4592 table.insert(buffer, "\\\\")
4593 else
4594 table.insert(buffer, byte)
4595 end
4596 end
4597 end
4598 table.insert(buffer, ' ')
4599 return table.concat(buffer)
4600 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.

```

4601 local function read_decompositions(file)
4602 return function()
4603 local from_codepoint, mapping
4604 for line in file:lines() do
4605 from_codepoint, mapping
4606 = line:match("^(%x+);[~;]*;%a*;%d+;%a*;([<%a>%x]*)")
4607 assert(from_codepoint ~= nil)
4608 if #mapping > 0 then
4609 break
4610 end
4611 end
4612 if #mapping == 0 then
4613 return nil
4614 end
4615 from_codepoint = tonumber(from_codepoint, 16)
4616 local to_codepoints, decomposition_type = {}, "canonical"
4617 for raw_codepoint in mapping:gmatch("%S+") do
4618 assert(#raw_codepoint > 0)
4619 if raw_codepoint:sub(1, 1) == "<" then
4620 decomposition_type = "compatibility"
4621 else
4622 local codepoint = tonumber(raw_codepoint, 16)
4623 table.insert(to_codepoints, codepoint)
4624 end
4625 end
4626 end
4627 end

```

```

4625 end
4626 assert(#to_codepoints > 0)
4627 return decomposition_type, from_codepoint, to_codepoints
4628 end
4629 end

```

Next, let's define some aliases in the generated file.

```

4630 print("local P, R, Cc, C = lpeg.P, lpeg.R, lpeg.Cc, lpeg.C")
4631 print("local fail = P(false)")
4632 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`.

```

4633 ;(function()
4634 local file = assert(io.open("UnicodeData.txt", "r"),
4635 [[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.

```

4636 local decomposition_types = {"canonical", "compatibility"}
4637 local prefix_trees = {}
4638 for _, decomposition_type in ipairs(decomposition_types) do
4639 prefix_trees[decomposition_type] = {}
4640 for char_length = 1, 4 do
4641 prefix_trees[decomposition_type][char_length]
4642 = {_type = "intermediate"}
4643 end
4644 end
4645 for decomposition_type, from_codepoint, to_codepoints
4646 in read_decompositions(file) do
4647 local from_code = utf8.char(from_codepoint)
4648 local node = prefix_trees[decomposition_type][#from_code]
4649 for i = 1, #from_code do
4650 local from_byte = from_code:sub(i, i)
4651 if i < #from_code then
4652 if node[from_byte] == nil then
4653 node[from_byte] = {_type = "intermediate"}
4654 end
4655 node = node[from_byte]
4656 else
4657 table.insert(node, {from_byte, to_codepoints, _type = "leaf"})

```

```

4658 end
4659 end
4660 end
4661 assert(file:close())

```

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

```

4662 print("M.decomposition_mapping = {}")
4663 for _, decomposition_type in ipairs(decomposition_types) do
4664 print("M.decomposition_mapping." .. decomposition_type .. " = {}")
4665 for length, prefix_tree in pairs(prefix_trees[decomposition_type])
4666 do
4667 local subparsers = {}
4668 depth_first_search(prefix_tree, "", function(node, path)
4669 if node._type == "leaf" then
4670 local from_byte, to_codepoints = table.unpack(node)
4671 local suffix = serialize_byte_parser(from_byte)
4672 .. " / " .. serialize_replacement(to_codepoints)
4673 if subparsers[path] ~= nil then
4674 subparsers[path] = subparsers[path] .. " + " .. suffix
4675 else
4676 subparsers[path] = suffix
4677 end
4678 end
4679 end, function(_, path)
4680 if #path > 0 then
4681 local byte = path:sub(#path, #path)
4682 local parent_path = path:sub(1, #path-1)
4683 local prefix = serialize_byte_parser(byte)
4684 local suffix
4685 if subparsers[path]:find(" %+ ") then
4686 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4687 else
4688 suffix = prefix .. " * " .. subparsers[path]
4689 end
4690 if subparsers[parent_path] ~= nil then
4691 subparsers[parent_path] = subparsers[parent_path]
4692 .. " + " .. suffix
4693 else
4694 subparsers[parent_path] = suffix
4695 end
4696 else
4697 print(
4698 "M.decomposition_mapping." .. decomposition_type
4699 .. "[" .. length .. "]" = " .. (subparsers[path] or "fail")
4700)
4701 end
4702 end)
4703 end

```

```

4704 end
4705 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](#).

```

4706 ;(function()
4707 local file = assert(io.open("HangulSyllableType.txt", "r"),
4708 [[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.

```

4709 local syllable_types = {"LV", "LVT"}
4710 local prefix_trees = {}
4711 for _, syllable_type in ipairs(syllable_types) do
4712 prefix_trees[syllable_type] = {}
4713 for char_length = 1, 4 do
4714 prefix_trees[syllable_type][char_length]
4715 = {_type = "intermediate"}
4716 end
4717 end
4718 for line in file:lines() do
4719 if #line == 0 or line:sub(1, 1) == "#" then
4720 goto continue
4721 end
4722 local codepoint, syllable_type = line:match("^([%x.]+)%s*;%s*(%a*)")
4723 assert(codepoint ~= nil)
4724 if prefix_trees[syllable_type] == nil then
4725 goto continue
4726 end
4727 local codepoint_start, codepoint_end
4728 if codepoint:find("%.%.") then
4729 codepoint_start, codepoint_end
4730 = codepoint:match("^(%x+)%.%.(%x+)$")
4731 else
4732 codepoint_start, codepoint_end = codepoint, codepoint
4733 end
4734 codepoint_start = tonumber(codepoint_start, 16)
4735 codepoint_end = tonumber(codepoint_end, 16)
4736 local prev_code, prev_leaf_node
4737 for codepoint = codepoint_start, codepoint_end do
4738 local code = utf8.char(codepoint)
4739 local node = prefix_trees[syllable_type][#code]

```

```

4740 for i = 1, #code do
4741 local byte = code:sub(i, i)
4742 if i < #code then
4743 if node[byte] == nil then
4744 node[byte] = {_type = "intermediate"}
4745 end
4746 node = node[byte]
4747 else
4748 local leaf_node
4749 if (
4750 prev_code ~= nil
4751 and #prev_code == #code
4752 and code:sub(1, #code - 1)
4753 == prev_code:sub(1, #code - 1)
4754) then
4755 assert(prev_leaf_node ~= nil)
4756 leaf_node = prev_leaf_node
4757 leaf_node[2] = byte
4758 else
4759 leaf_node = {byte, byte, _type = "leaf"}
4760 table.insert(node, leaf_node)
4761 end
4762 prev_code, prev_leaf_node = code, leaf_node
4763 end
4764 end
4765 end
4766 ::continue::
4767 end
4768 assert(file:close())

```

Next, we will define constants and functions with the Hangul syllable (de)composition algorithm.

```

4769 print(string.format("local s_base = %d", tonumber("AC00", 16)))
4770 print(string.format("local l_base = %d", tonumber("1100", 16)))
4771 print(string.format("local v_base = %d", tonumber("1161", 16)))
4772 print(string.format("local t_base = %d", tonumber("11A7", 16)))
4773 print(string.format("local l_count = %d", 19))
4774 print(string.format("local v_count = %d", 21))
4775 print(string.format("local t_count = %d", 28))
4776 print("local n_count = v_count * t_count")
4777 print("local s_count = l_count * n_count")
4778
4779 print("local function hangul_decompose_LV(byte)")
4780 print(" local s = utf8.codepoint(byte)")
4781 print(" local s_index = s - s_base")
4782 print(" local l_index = s_index // n_count")
4783 print(" local v_index = (s_index % n_count) // t_count")

```

```

4784 print(" local l_part = l_base + l_index")
4785 print(" local v_part = v_base + v_index")
4786 print(" return utf8.char(l_part) .. utf8.char(v_part)")
4787 print("end")
4788
4789 print("local function hangul_decompose_LVT(byte)")
4790 print(" local s = utf8.codepoint(byte)")
4791 print(" local s_index = s - s_base")
4792 print(" local lv_index = (s_index // t_count) * t_count")
4793 print(" local t_index = s_index % t_count")
4794 print(" local lv_part = s_base + lv_index")
4795 print(" local t_part = t_base + t_index")
4796 print(" return utf8.char(lv_part) .. utf8.char(t_part)")
4797 print("end")
4798
4799 print("function M.hangul_compose(first_byte, second_byte)")
4800 print(" local last = utf8.codepoint(first_byte)")
4801 print(" local ch = utf8.codepoint(second_byte)")
4802 print(" local l_index = last - l_base")
4803 print(" if 0 <= l_index and l_index < l_count then")
4804 print(" local v_index = ch - v_base")
4805 print(" if 0 <= v_index and v_index < v_count then")
4806 print(" return utf8.char("
4807 s_base + (l_index * v_count + v_index) * t_count"
4808 ")")
4809 print(" end")
4810 print(" end")
4811 print(" local s_index = last - s_base")
4812 print(" if 0 <= s_index and s_index < s_count")
4813 print(" and s_index % t_count == 0 then")
4814 print(" local t_index = ch - t_base")
4815 print(" if 0 < t_index and t_index < t_count then")
4816 print(" return utf8.char(last + t_index)")
4817 print(" end")
4818 print(" end")
4819 print(" return nil")
4820 print("end")

```

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

```

4821 print("M.decomposition_mapping.hangul = {}")
4822 for _, syllable_type in ipairs(syllable_types) do
4823 print("M.decomposition_mapping.hangul." .. syllable_type .. " = {}")
4824 for length, prefix_tree in pairs(prefix_trees[syllable_type]) do
4825 local subparsers = {}
4826 depth_first_search(prefix_tree, "", function(node, path)
4827 if node._type == "leaf" then
4828 local start_byte, end_byte = table.unpack(node)
4829 local suffix

```

```

4830 if start_byte == end_byte then
4831 suffix = serialize_byte_parser(start_byte)
4832 else
4833 assert(start_byte < end_byte)
4834 suffix = serialize_byte_range_parser(start_byte, end_byte)
4835 end
4836 if subparsers[path] ~= nil then
4837 subparsers[path] = subparsers[path] .. " + " .. suffix
4838 else
4839 subparsers[path] = suffix
4840 end
4841 end
4842 end, function(_, path)
4843 if #path > 0 then
4844 local byte = path:sub(#path, #path)
4845 local parent_path = path:sub(1, #path-1)
4846 local prefix = serialize_byte_parser(byte)
4847 local suffix
4848 if subparsers[path]:find(" %+ ") then
4849 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4850 else
4851 suffix = prefix .. " * " .. subparsers[path]
4852 end
4853 if subparsers[parent_path] ~= nil then
4854 subparsers[parent_path] = subparsers[parent_path]
4855 .. " + " .. suffix
4856 else
4857 subparsers[parent_path] = suffix
4858 end
4859 else
4860 if subparsers[path] then
4861 print(
4862 "M.decomposition_mapping.hangul." .. syllable_type
4863 .. "[" .. length .. "] = C(" .. subparsers[path] .. ")"
4864 .. " / hangul_decompose_" .. syllable_type
4865)
4866 else
4867 print(
4868 "M.decomposition_mapping.hangul." .. syllable_type
4869 .. "[" .. length .. "] = fail"
4870)
4871 end
4872 end
4873 end)
4874 end
4875 end
4876 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.

```
4877 ;(function()
4878 local file = assert(io.open("DerivedNormalizationProps.txt", "r"),
4879 [[Could not open file "DerivedNormalizationProps.txt"]])
4880 local full_composition_exclusions = {}
4881 for line in file:lines() do
4882 if #line == 0 or line:sub(1, 1) == "#" then
4883 goto continue
4884 end
4885 local codepoint, property = line:match("^([%x.]+)%s*;%s*([%a_]+)")
4886 assert(codepoint ~= nil)
4887 if property ~= "Full_Composition_Exclusion" then
4888 goto continue
4889 end
4890 local codepoint_start, codepoint_end
4891 if codepoint:find("%.%.") then
4892 codepoint_start, codepoint_end
4893 = codepoint:match("^(%x+)%.%.(%x+)$")
4894 else
4895 codepoint_start, codepoint_end = codepoint, codepoint
4896 end
4897 codepoint_start = tonumber(codepoint_start, 16)
4898 codepoint_end = tonumber(codepoint_end, 16)
4899 for codepoint = codepoint_start, codepoint_end do
4900 full_composition_exclusions[codepoint] = true
4901 end
4902 ::continue::
4903 end
4904 assert(file:close())
```

Next, let's also read the file `UnicodeData.txt`.

```
4905 file = assert(io.open("UnicodeData.txt", "r"),
4906 [[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.

```
4907 local prefix_trees = {starters = {}, both = {}}
4908 for first_char_length = 1, 4 do
4909 prefix_trees.starters[first_char_length]
4910 = {_type = "intermediate"}
4911 prefix_trees.both[first_char_length] = {}
```



```

4912 for second_char_length = 1, 4 do
4913 prefix_trees.both[first_char_length][second_char_length]
4914 = {_type = "intermediate"}
4915 end
4916 end
4917 local seen_starter_codes = {}
4918 for decomposition_type, to_codepoint, from_codepoints
4919 in read_decompositions(file) do
4920 if (
4921 decomposition_type ~= "canonical"
4922 or #from_codepoints ~= 2
4923 or full_composition_exclusions[to_codepoint]
4924) then
4925 goto continue
4926 end
4927 local starter_code = utf8.char(from_codepoints[1])
4928 local combining_character_code = utf8.char(from_codepoints[2])
4929 local starter_node = prefix_trees.starters[#starter_code]
4930 local both_node
4931 = prefix_trees.both[#starter_code][#combining_character_code]
4932 for i = 1, #starter_code do
4933 local from_byte = starter_code:sub(i, i)
4934 if both_node[from_byte] == nil then
4935 both_node[from_byte] = {_type = "intermediate"}
4936 end
4937 both_node = both_node[from_byte]
4938 if i < #starter_code then
4939 if starter_node[from_byte] == nil then
4940 starter_node[from_byte] = {_type = "intermediate"}
4941 end
4942 starter_node = starter_node[from_byte]
4943 elseif seen_starter_codes[starter_code] == nil then
4944 seen_starter_codes[starter_code] = true
4945 table.insert(starter_node, {from_byte, _type = "leaf"})
4946 end
4947 end
4948 for i = 1, #combining_character_code do
4949 local from_byte = combining_character_code:sub(i, i)
4950 if i < #combining_character_code then
4951 if both_node[from_byte] == nil then
4952 both_node[from_byte] = {_type = "intermediate"}
4953 end
4954 both_node = both_node[from_byte]
4955 else
4956 table.insert(
4957 both_node,
4958 {from_byte, to_codepoint, _type = "leaf"}

```

```

4959)
4960 end
4961 end
4962 ::continue::
4963 end
4964 assert(file:close())

```

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

```

4965 print("M.composition_mapping = {starters = {}, both = {}}")
4966 for first_char_length = 1, 4 do
4967 local prefix_tree = prefix_trees.starters[first_char_length]
4968 local subparsers = {}
4969 depth_first_search(prefix_tree, "", function(node, path)
4970 if node._type == "leaf" then
4971 local from_byte = table.unpack(node)
4972 local suffix = serialize_byte_parser(from_byte)
4973 if subparsers[path] ~= nil then
4974 subparsers[path] = subparsers[path] .. " + " .. suffix
4975 else
4976 subparsers[path] = suffix
4977 end
4978 end
4979 end, function(_, path)
4980 if #path > 0 then
4981 local byte = path:sub(#path, #path)
4982 local parent_path = path:sub(1, #path-1)
4983 local prefix = serialize_byte_parser(byte)
4984 local suffix
4985 if subparsers[path]:find(" %+ ") then
4986 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4987 else
4988 suffix = prefix .. " * " .. subparsers[path]
4989 end
4990 if subparsers[parent_path] ~= nil then
4991 subparsers[parent_path] = subparsers[parent_path]
4992 .. " + " .. suffix
4993 else
4994 subparsers[parent_path] = suffix
4995 end
4996 else
4997 print(
4998 "M.composition_mapping.starters["
4999 .. first_char_length .. "] = " .. (subparsers[path] or "fail")
5000)
5001 end
5002 end)
5003 print(
5004 string.format(

```

```

5005 "M.composition_mapping.both[%d] = {}",
5006 first_char_length
5007)
5008)
5009 for second_char_length = 1, 4 do
5010 prefix_tree
5011 = prefix_trees.both[first_char_length][second_char_length]
5012 subparsers = {}
5013 depth_first_search(prefix_tree, "", function(node, path)
5014 if node._type == "leaf" then
5015 local from_byte, to_codepoint = table.unpack(node)
5016 local suffix = serialize_byte_parser(from_byte)
5017 .. " / " .. serialize_replacement({to_codepoint})
5018 if subparsers[path] ~= nil then
5019 subparsers[path] = subparsers[path] .. " + " .. suffix
5020 else
5021 subparsers[path] = suffix
5022 end
5023 end
5024 end, function(_, path)
5025 if #path > 0 then
5026 local byte = path:sub(#path, #path)
5027 local parent_path = path:sub(1, #path-1)
5028 local prefix = serialize_byte_parser(byte)
5029 local suffix
5030 if subparsers[path]:find(" %+ ") then
5031 suffix = prefix .. " * (" .. subparsers[path] .. ")"
5032 else
5033 suffix = prefix .. " * " .. subparsers[path]
5034 end
5035 if subparsers[parent_path] ~= nil then
5036 subparsers[parent_path] = subparsers[parent_path]
5037 .. " + " .. suffix
5038 else
5039 subparsers[parent_path] = suffix
5040 end
5041 else
5042 print(
5043 "M.composition_mapping.both[" .. first_char_length .. "]["
5044 .. second_char_length .. "] = "
5045 .. (subparsers[path] or "fail")
5046)
5047 end
5048 end)
5049 end
5050 end
5051 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`.

```
5052 ;(function()
5053 local file = assert(io.open("CaseFolding.txt", "r"),
5054 [[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.

```
5055 local prefix_trees = {}
5056 for char_length = 1, 4 do
5057 prefix_trees[char_length] = {_type = "intermediate"}
5058 end
5059 for line in file:lines() do
5060 if #line == 0 or line:sub(1, 1) == "#" then
5061 goto continue
5062 end
5063 local raw_from_codepoint, status, raw_to_codepoints
5064 = line:match("^(%x+); ([CFST]); ([%x]+);")
5065 assert(raw_from_codepoint ~= nil)
5066 assert(status ~= nil)
5067 assert(raw_to_codepoints ~= nil)
5068 if status ~= "C" and status ~= "F" then
5069 goto continue
5070 end
5071 local from_codepoint = tonumber(raw_from_codepoint, 16)
5072 local to_codepoints = {}
5073 for raw_codepoint in raw_to_codepoints:gmatch('%x+') do
5074 local codepoint = tonumber(raw_codepoint, 16)
5075 table.insert(to_codepoints, codepoint)
5076 end
5077 local from_code = utf8.char(from_codepoint)
5078 local node = prefix_trees[#from_code]
5079 for i = 1, #from_code do
5080 local from_byte = from_code:sub(i, i)
5081 if i < #from_code then
5082 if node[from_byte] == nil then
5083 node[from_byte] = {_type = "intermediate"}
5084 end
5085 node = node[from_byte]
5086 else
5087 table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
5088 end
5089 end
```

```

5090 ::continue::
5091 end
5092 assert(file:close())

```

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

```

5093 print("M.casefold_mapping = {}")
5094 for length, prefix_tree in pairs(prefix_trees) do
5095 local subparsers = {}
5096 depth_first_search(prefix_tree, "", function(node, path)
5097 if node._type == "leaf" then
5098 local from_byte, to_codepoints = table.unpack(node)
5099 local suffix = serialize_byte_parser(from_byte)
5100 .. " / " .. serialize_replacement(to_codepoints)
5101 if subparsers[path] ~= nil then
5102 subparsers[path] = subparsers[path] .. " + " .. suffix
5103 else
5104 subparsers[path] = suffix
5105 end
5106 end
5107 end, function(_, path)
5108 if #path > 0 then
5109 local byte = path:sub(#path, #path)
5110 local parent_path = path:sub(1, #path-1)
5111 local prefix = serialize_byte_parser(byte)
5112 local suffix
5113 if subparsers[path]:find(" %+ ") then
5114 suffix = prefix .. " * (" .. subparsers[path] .. ")"
5115 else
5116 suffix = prefix .. " * " .. subparsers[path]
5117 end
5118 if subparsers[parent_path] ~= nil then
5119 subparsers[parent_path] = subparsers[parent_path]
5120 .. " + " .. suffix
5121 else
5122 subparsers[parent_path] = suffix
5123 end
5124 else
5125 print(
5126 "M.casefold_mapping[" .. length .. "] = "
5127 .. (subparsers[path] or "fail")
5128)
5129 end
5130 end)
5131 end
5132 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`.

```
5133 ;(function()
5134 local file = assert(io.open("UnicodeData.txt", "r"),
5135 [[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.

```
5136 local categories = {"L", "N", "P", "Pc", "S", "Z"}
5137 local prefix_trees = {}
5138 for _, category in ipairs(categories) do
5139 prefix_trees[category] = {}
5140 for char_length = 1, 4 do
5141 prefix_trees[category][char_length] = {_type = "intermediate"}
5142 end
5143 end
5144 for line in file:lines() do
5145 local codepoint, full_category = line:match("^(%x+);[~;]*;(%a*)")
5146 assert(#full_category >= 1)
5147 local major_category = full_category:sub(1, 1)
5148 for _, category in ipairs({full_category, major_category}) do
5149 if prefix_trees[category] == nil then
5150 goto continue
5151 end
5152 local code = utf8.char(tonumber(codepoint, 16))
5153 local node = prefix_trees[category][#code]
5154 for i = 1, #code do
5155 local byte = code:sub(i, i)
5156 if i < #code then
5157 if node[byte] == nil then
5158 node[byte] = {_type = "intermediate"}
5159 end
5160 node = node[byte]
5161 else
5162 table.insert(node, {byte, _type = "leaf"})
5163 end
5164 end
5165 ::continue::
5166 end
5167 end
5168 assert(file:close())
```

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

```
5169 print("M.categories = {}")
```

```

5170 for _, category in ipairs(categories) do
5171 print("M.categories." .. category .. " = {}".format({}))
5172 for length, prefix_tree in pairs(prefix_trees[category]) do
5173 local subparsers = {}
5174 depth_first_search(prefix_tree, "", function(node, path)
5175 if node._type == "leaf" then
5176 local byte = node[1]
5177 local suffix = serialize_byte_parser(byte)
5178 if subparsers[path] ~= nil then
5179 subparsers[path] = subparsers[path] .. " + " .. suffix
5180 else
5181 subparsers[path] = suffix
5182 end
5183 end
5184 end, function(_, path)
5185 if #path > 0 then
5186 local byte = path:sub(#path, #path)
5187 local parent_path = path:sub(1, #path-1)
5188 local prefix = serialize_byte_parser(byte)
5189 local suffix
5190 if subparsers[path]:find(" %+ ") then
5191 suffix = prefix .. " * (" .. subparsers[path] .. ")"
5192 else
5193 suffix = prefix .. " * " .. subparsers[path]
5194 end
5195 if subparsers[parent_path] ~= nil then
5196 subparsers[parent_path] = subparsers[parent_path]
5197 .. " + " .. suffix
5198 else
5199 subparsers[parent_path] = suffix
5200 end
5201 else
5202 print(
5203 "M.categories." .. category .. "[" .. length .. "] = "
5204 .. (subparsers[path] or "fail")
5205)
5206 end
5207 end)
5208 end
5209 end
5210 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`.

```
5211 ;(function()
5212 local file = assert(io.open("UnicodeData.txt", "r"),
5213 [[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.

```
5214 local prefix_trees = {}
5215 for char_length = 1, 4 do
5216 prefix_trees[char_length] = {_type = "intermediate"}
5217 end
5218 for line in file:lines() do
5219 local codepoint, combining_class
5220 = line:match("^(%x+);[~;]*;%a*;%d+")
5221 combining_class = tonumber(combining_class)
5222 if combining_class == 0 then
5223 goto continue
5224 end
5225 local code = utf8.char(tonumber(codepoint, 16))
5226 local node = prefix_trees[#code]
5227 for i = 1, #code do
5228 local byte = code:sub(i, i)
5229 if i < #code then
5230 if node[byte] == nil then
5231 node[byte] = {_type = "intermediate"}
5232 end
5233 node = node[byte]
5234 else
5235 table.insert(node, {byte, combining_class, _type = "leaf"})
5236 end
5237 end
5238 ::continue::
5239 end
5240 assert(file:close())
```

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

```
5241 print("M.ccc = {}")
5242 for length, prefix_tree in pairs(prefix_trees) do
5243 local subparsers = {}
5244 depth_first_search(prefix_tree, "", function(node, path)
5245 if node._type == "leaf" then
5246 local byte, combining_class = table.unpack(node)
5247 local suffix = serialize_byte_parser(byte)
5248 .. " * Cc(" .. tostring(combining_class) .. ")"
5249 if subparsers[path] ~= nil then
5250 subparsers[path] = subparsers[path] .. " + " .. suffix
```



```

5251 else
5252 subparsers[path] = suffix
5253 end
5254 end
5255 end, function(_, path)
5256 if #path > 0 then
5257 local byte = path:sub(#path, #path)
5258 local parent_path = path:sub(1, #path-1)
5259 local prefix = serialize_byte_parser(byte)
5260 local suffix
5261 if subparsers[path]:find(" %+ ") then
5262 suffix = prefix .. " * (" .. subparsers[path] .. ")"
5263 else
5264 suffix = prefix .. " * " .. subparsers[path]
5265 end
5266 if subparsers[parent_path] ~= nil then
5267 subparsers[parent_path] = subparsers[parent_path]
5268 .. " + " .. suffix
5269 else
5270 subparsers[parent_path] = suffix
5271 end
5272 else
5273 print(
5274 "M.ccc[" .. length .. "] = " .. (subparsers[path] or "fail")
5275)
5276 end
5277 end)
5278 end
5279 end)()
5280 print("-- luacheck: pop")
5281 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 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.

```

5282 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.

```

5283 function util.err(msg, exit_code)
5284 io.stderr:write("markdown.lua: " .. msg .. "\n")
5285 os.exit(exit_code or 1)
5286 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.

```

5287 function util.cache(dir, string, salt, transform, suffix)
5288 local digest = md5.sumhexa(string .. (salt or ""))
5289 local name = util.pathname(dir, digest .. suffix)
5290 local file = io.open(name, "r")
5291 local result = nil
5292 if file == nil then -- If no cache entry exists, create a new one.
5293 file = assert(io.open(name, "w"),
5294 [[Could not open file]] .. name .. [[for writing]])
5295 result = string
5296 if transform ~= nil then
5297 result = transform(result)
5298 end
5299 assert(file:write(result))
5300 assert(file:close())
5301 end
5302 return name, result
5303 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.

```

5304 function util.cache_verbatim(dir, string)
5305 local name = util.cache(dir, string, nil, nil, ".verbatim")
5306 return name
5307 end

```

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

```

5308 function util.table_copy(t)
5309 local u = { }
5310 for k, v in pairs(t) do u[k] = v end
5311 return setmetatable(u, getmetatable(t))
5312 end

```

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

```

5313 function util.encode_json_string(s)
5314 s = s:gsub([[\\]], [[\\]])
5315 s = s:gsub([[\"]], [[\"]])
5316 return [["]] .. s .. [["]]
5317 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is

used. The method is a copy of the tab expansion algorithm from Ierusalimschy [18, Chapter 21].

```
5318 function util.expand_tabs_in_line(s, tabstop)
5319 local tab = tabstop or 4
5320 local corr = 0
5321 return (s:gsub("\t", function(p)
5322 local sp = tab - (p - 1 + corr) % tab
5323 corr = corr - 1 + sp
5324 return string.rep(" ", sp)
5325 end))
5326 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.

```
5327 function util.walk(t, f)
5328 local typ = type(t)
5329 if typ == "string" then
5330 f(t)
5331 elseif typ == "table" then
5332 local i = 1
5333 local n
5334 n = t[i]
5335 while n do
5336 util.walk(n, f)
5337 i = i + 1
5338 n = t[i]
5339 end
5340 elseif typ == "function" then
5341 local ok, val = pcall(t)
5342 if ok then
5343 util.walk(val, f)
5344 end
5345 else
5346 f(tostring(t))
5347 end
5348 end
```

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

```
5349 function util.flatten(ary)
5350 local new = {}
5351 for _,v in ipairs(ary) do
5352 if type(v) == "table" then
5353 for _,w in ipairs(util.flatten(v)) do
5354 new[#new + 1] = w
5355 end
5356 end
5357 end
5358 return new
```

```

5355 end
5356 else
5357 new[#new + 1] = v
5358 end
5359 end
5360 return new
5361 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.

```

5362 function util.rope_to_string(rope)
5363 local buffer = {}
5364 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
5365 return table.concat(buffer)
5366 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.

```

5367 function util.rope_last(rope)
5368 if #rope == 0 then
5369 return nil
5370 else
5371 local l = rope[#rope]
5372 if type(l) == "table" then
5373 return util.rope_last(l)
5374 else
5375 return l
5376 end
5377 end
5378 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  $1 \leq i \leq \#ary$ .

```

5379 function util.intersperse(ary, x)
5380 local new = {}
5381 local l = #ary
5382 for i,v in ipairs(ary) do
5383 local n = #new
5384 new[n + 1] = v
5385 if i ~= l then
5386 new[n + 2] = x
5387 end
5388 end
5389 return new
5390 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  $1 \leq i \leq \#ary$ .

```
5391 function util.map(ary, f)
5392 local new = {}
5393 for i,v in ipairs(ary) do
5394 new[i] = f(v)
5395 end
5396 return new
5397 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.

```
5398 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
5399 local char_escapes_list = ""
5400 for i,_ in pairs(char_escapes) do
5401 char_escapes_list = char_escapes_list .. i
5402 end
```

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

```
5403 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.

```
5404 if string_escapes then
5405 for k,v in pairs(string_escapes) do
5406 escapable = P(k) / v + escapable
5407 end
5408 end
```

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

```
5409 local escape_string = Cs((escapable + any)^0)
```

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

```

5410 return function(s)
5411 return lpeg.match(escape_string, s)
5412 end
5413 end

```

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

```

5414 function util.pathname(dir, file)
5415 if #dir == 0 then
5416 return file
5417 else
5418 return dir .. "/" .. file
5419 end
5420 end

```

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

```

5421 function util.salt(options)
5422 local opt_string = {}
5423 for k, _ in pairs(defaultOptions) do
5424 local v = options[k]
5425 if type(v) == "table" then
5426 for _, i in ipairs(v) do
5427 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
5428 end
5429 end
5430 end
5431 table.sort(opt_string)
5432 local salt = table.concat(opt_string, ",")
5433 .. "," .. metadata.version
5434 return salt
5435 end

```

The `cacheDir` option is disregarded.

```

5429 elseif k ~= "cacheDir" then
5430 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5431 end
5432 end
5433 table.sort(opt_string)
5434 local salt = table.concat(opt_string, ",")
5435 .. "," .. metadata.version
5436 return salt
5437 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.

```

5438 util.warning = (function()
5439 local function warning(s)
5440 io.stderr:write("Warning: " .. s .. "\n")
5441 end
5442 for _, message in ipairs(early_warnings) do
5443 warning(message)
5444 end
5445 return warning
5446 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.

```
5447 util.casefold = (function()
5448 local fail, any = P(false), P(1)
5449 local eof = -any
```

First, define a parser that will case-fold a character.

```
5450 local fold_character = fail
5451 for n = 1, 4 do
5452 fold_character
5453 = fold_character
5454 + unicode_data.casefold_mapping[n]
5455 end
5456 fold_character
5457 = fold_character
5458 + C(any)
```

Next, define a parser that will case-fold a string.

```
5459 local fold_string = Ct(fold_character^0) * eof
5460 return function(s, form)
5461 local result = table.concat(lpeg.match(fold_string, s))
5462 assert(result ~= nil)
```

For NFD and NFKD normalization forms, normalize the case-folded string and then repeat the fold-and-normalize operation.

```
5463 if form == "nfd" or form == "nfkd" then
5464 result = util.normalize(result, form)
5465 result = table.concat(lpeg.match(fold_string, result))
5466 assert(result ~= nil)
5467 result = util.normalize(result, form)
5468 end
5469 return result
5470 end
5471 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.

```
5472 util.canonically_order = (function()
5473 local fail, any = P(false), P(1)
5474 local eof = -any
5475 local cont = R("\128\191")
5476 local utf8_character
```

```

5477 = R("\0\127")
5478 + R("\194\223") * cont
5479 + R("\224\239") * cont * cont
5480 + R("\240\244") * cont * cont * cont

```

First, define a parser that will determine the combining class of a character.

```

5481 local classify_character = fail
5482 for n = 1, 4 do
5483 classify_character
5484 = classify_character
5485 + unicode_data.ccc[n]
5486 end
5487 classify_character
5488 = classify_character
5489 + utf8_character * Cc(0)

```

Next, define a parser that will determine the combining classes of all characters in a string.

```

5490 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.

```

5491 return function(s)
5492 local s_len = utf8.len(s)
5493 if s == false or s_len <= 1 then
5494 return s
5495 end

```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```

5496 local classes = lpeg.match(classify_string, s)
5497 if classes == nil then
5498 return s
5499 end
5500 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.

```

5501 local non_starter_ranges = {}
5502 local first_non_starter, last_non_starter = nil, nil
5503 for i = 1, #classes do
5504 if first_non_starter == nil then
5505 if classes[i] ~= 0 then
5506 first_non_starter, last_non_starter = i, i
5507 end
5508 else
5509 if classes[i] == 0 then

```



```

5510 table.insert(
5511 non_starter_ranges,
5512 {first_non_starter, last_non_starter}
5513)
5514 first_non_starter, last_non_starter = nil, nil
5515 else
5516 last_non_starter = i
5517 end
5518 end
5519 end
5520 if first_non_starter ~= nil then
5521 table.insert(
5522 non_starter_ranges,
5523 {first_non_starter, last_non_starter}
5524)
5525 end
5526 if #non_starter_ranges == 0 then
5527 return s
5528 end
5529 local max_range_length = 0
5530 for _, range in ipairs(non_starter_ranges) do
5531 local range_start, range_end = table.unpack(range)
5532 local range_length = range_end - range_start + 1
5533 if range_length > max_range_length then
5534 max_range_length = range_length
5535 end
5536 end
5537 if max_range_length <= 1 then
5538 return s
5539 end

```

Then, construct a buffer of all characters in the string.

```

5540 local buffer = {}
5541 for _, code in utf8.codes(s) do
5542 local char = utf8.char(code)
5543 table.insert(buffer, char)
5544 end
5545 assert(#buffer == s_len)

```

Next, perform a local bubble sort over the ranges of non-starter characters.

```

5546 for _, range in ipairs(non_starter_ranges) do
5547 local range_start, range_end = table.unpack(range)
5548 local range_length = range_end - range_start + 1
5549 for _ = 1, range_length - 1 do
5550 local swapped = false
5551 for i = range_start, range_end - 1 do
5552 local j = i + 1
5553 if classes[i] > classes[j] then

```

```

5554 classes[i], classes[j] = classes[j], classes[i]
5555 buffer[i], buffer[j] = buffer[j], buffer[i]
5556 swapped = true
5557 end
5558 end
5559 if not swapped then
5560 break
5561 end
5562 end
5563 end

```

Finally, concatenate the buffer and return an ordered string.

```

5564 return table.concat(buffer, "")
5565 end
5566 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.

```

5567 util.decompose = (function()
5568 local fail, any = P(false), P(1)
5569 local eof = -any
5570 local decomposition_types = {"canonical", "compatibility"}

```

First, define parsers that will decompose a character.

```

5571 local decompose_character = {}
5572 for _, decomposition_type in ipairs(decomposition_types) do
5573 decompose_character[decomposition_type] = fail
5574 for n = 1, 4 do
5575 decompose_character[decomposition_type]
5576 = decompose_character[decomposition_type]
5577 + unicode_data.decomposition_mapping[decomposition_type][n]
5578 end
5579 decompose_character[decomposition_type]
5580 = decompose_character[decomposition_type]
5581 + C(any)
5582 end
5583 local hangul = unicode_data.decomposition_mapping.hangul
5584 decompose_character.hangul = {}
5585 for syllable_type, _ in pairs(hangul) do
5586 decompose_character.hangul[syllable_type] = fail
5587 for n = 1, 4 do
5588 decompose_character.hangul[syllable_type]
5589 = decompose_character.hangul[syllable_type]
5590 + hangul[syllable_type][n]
5591 end

```

```

5592 decompose_character.hangul[syllable_type]
5593 = decompose_character.hangul[syllable_type]
5594 + C(any)
5595 end

```

Next, define a parser that will decompose a string.

```

5596 local decompose_string = {}
5597 for _, decomposition_type in ipairs(decomposition_types) do
5598 decompose_string[decomposition_type]
5599 = Ct(decompose_character[decomposition_type]^0) * eof
5600 end
5601 decompose_string.hangul = {}
5602 for syllable_type, _ in pairs(hangul) do
5603 decompose_string.hangul[syllable_type]
5604 = Ct(decompose_character.hangul[syllable_type]^0) * eof
5605 end
5606 local function _decompose(s, parser)
5607 assert(s ~= nil)
5608 local result = table.concat(lpeg.match(parser, s), "")
5609 assert(result ~= nil)
5610 return result
5611 end
5612 return function(s, decomposition_type)
5613 assert(
5614 decomposition_type == "canonical"
5615 or decomposition_type == "compatibility"
5616)
5617 local prev_s
5618 local next_s = s
5619 repeat
5620 prev_s = next_s
5621 local function decompose(...) next_s = _decompose(next_s, ...) end
5622 decompose(decompose_string.canonical)
5623 if decomposition_type == "compatibility" then
5624 decompose(decompose_string.compatibility)
5625 end
5626 decompose(decompose_string.hangul.LVT)
5627 decompose(decompose_string.hangul.LV)
5628 until prev_s == next_s
5629 return util.canonically_order(next_s)
5630 end
5631 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. Un-

like the low-level parsers, this high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```
5632 util.compose = (function()
5633 local fail, any = P(false), P(1)
5634 local eof = -any
5635 local cont = R("\128\191")
5636 local utf8_character
5637 = R("\0\127")
5638 + R("\194\223") * cont
5639 + R("\224\239") * cont * cont
5640 + R("\240\244") * cont * cont * cont
```

First, define a parser that will determine the combining class of a character.

```
5641 local classify_character = fail
5642 for n = 1, 4 do
5643 classify_character
5644 = classify_character
5645 + unicode_data.ccc[n]
5646 end
5647 classify_character
5648 = classify_character
5649 + utf8_character * Cc(0)
```

Next, define a parser that will determine the combining classes of all characters in a string.

```
5650 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.

```
5651 local compose_characters = fail
5652 for m = 1, 4 do
5653 local starter = #unicode_data.composition_mapping.starters[m]
5654 local both = fail
5655 for n = 1, 4 do
5656 both = (
5657 both
5658 + #unicode_data.composition_mapping.both[m][n]
5659 * unicode_data.composition_mapping.both[m][n]
5660)
5661 end
5662 compose_characters = compose_characters + starter * both
5663 end
```

When the function is called, first check whether the string is trivially ordered. If it is, return it without any changes.

```
5664 return function(s)
5665 local s_len = utf8.len(s)
5666 if s == false or s_len <= 1 then
```

```

5667 return s
5668 end

```

Otherwise, determine the combining classes of all characters in the string. If the string cannot be decoded with UTF-8, return it unchanged.

```

5669 local classes = lpeg.match(classify_string, s)
5670 if classes == nil then
5671 return s
5672 end
5673 assert(#classes == s_len)

```

Otherwise, construct a buffer of all characters in the string.

```

5674 local buffer = {}
5675 for _, code in utf8.codes(s) do
5676 local char = utf8.char(code)
5677 table.insert(buffer, char)
5678 end
5679 assert(#buffer == s_len)

```

Finally, implement the composition algorithm.

First, find the first starter character in the string.

```

5680 local starter = 1
5681 while starter <= s_len and classes[starter] ~= 0 do
5682 starter = starter + 1
5683 end
5684 local candidate_combining_mark = starter + 1

```

Next, apply the composition rules until we reach the end of the string.

```

5685 while candidate_combining_mark <= s_len do
5686 local L = buffer[starter]
5687 local C = buffer[candidate_combining_mark]
5688 local P = lpeg.match(compose_characters, L .. C)
5689 if P ~= nil then
5690 buffer[starter] = P
5691 buffer[candidate_combining_mark] = ""
5692 else
5693 if classes[candidate_combining_mark] == 0 then
5694 starter = candidate_combining_mark
5695 end
5696 candidate_combining_mark = candidate_combining_mark + 1
5697 end
5698 assert(starter <= s_len)
5699 end

```

Next, iterate over the string once more and compose Hangul syllables.

```

5700 for i = 1, s_len - 1 do
5701 local last, ch = buffer[i], buffer[i + 1]
5702 if last ~= "" and ch ~= "" then
5703 local composite = unicode_data.hangul_compose(last, ch)

```

```

5704 if composite ~= nil then
5705 buffer[i] = ""
5706 buffer[i + 1] = composite
5707 end
5708 end
5709 end

```

Finally, concatenate the buffer and return an ordered string.

```

5710 return table.concat(buffer, "")
5711 end
5712 end)()

```

The `util.normalize` method normalizes a UTF-8-encoded canonically ordered Unicode string `s` using the normalization form `form`.

```

5713 function util.normalize(s, form)
5714 if form == "nfd" then
5715 return util.decompose(s, "canonical")
5716 elseif form == "nfkd" then
5717 return util.decompose(s, "compatibility")
5718 elseif form == "nfc" then
5719 return util.compose(util.decompose(s, "canonical"))
5720 elseif form == "nfkc" then
5721 return util.compose(util.decompose(s, "compatibility"))
5722 else
5723 error(string.format('Unexpected normal form "%s"', form))
5724 end
5725 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.

```

5726 function util.find_file(filename)
5727 if resolvers ~= nil then
5728 return resolvers.findfile(filename)
5729 else
5730 return kpse.find_file(filename)
5731 end
5732 end
5733 function util.find_files(filename)
5734 if resolvers ~= nil then
5735 return resolvers.findfiles(filename)
5736 else
5737 return {kpse.lookup(filename, {all=true})}
5738 end
5739 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.

```
5740 local entities = {}
5741
5742 local character_entities = {
5743 ["Tab"] = 9,
5744 ["NewLine"] = 10,
5745 ["excl"] = 33,
5746 ["QUOT"] = 34,
5747 ["quot"] = 34,
5748 ["num"] = 35,
5749 ["dollar"] = 36,
5750 ["percent"] = 37,
5751 ["AMP"] = 38,
5752 ["amp"] = 38,
5753 ["apos"] = 39,
5754 ["lpar"] = 40,
5755 ["rpar"] = 41,
5756 ["ast"] = 42,
5757 ["midast"] = 42,
5758 ["plus"] = 43,
5759 ["comma"] = 44,
5760 ["period"] = 46,
5761 ["sol"] = 47,
5762 ["colon"] = 58,
5763 ["semi"] = 59,
5764 ["LT"] = 60,
5765 ["lt"] = 60,
5766 ["nvlt"] = {60, 8402},
5767 ["bne"] = {61, 8421},
5768 ["equals"] = 61,
5769 ["GT"] = 62,
5770 ["gt"] = 62,
5771 ["nvgt"] = {62, 8402},
5772 ["quest"] = 63,
5773 ["commat"] = 64,
5774 ["lbrack"] = 91,
5775 ["lsqb"] = 91,
5776 ["bsol"] = 92,
5777 ["rbrack"] = 93,
5778 ["rsqb"] = 93,
5779 ["Hat"] = 94,
5780 ["UnderBar"] = 95,
5781 ["lowbar"] = 95,
```

```

5782 ["DiacriticalGrave"] = 96,
5783 ["grave"] = 96,
5784 ["fjlig"] = {102, 106},
5785 ["lbrace"] = 123,
5786 ["lcub"] = 123,
5787 ["VerticalLine"] = 124,
5788 ["verbar"] = 124,
5789 ["vert"] = 124,
5790 ["rbrace"] = 125,
5791 ["rcub"] = 125,
5792 ["NonBreakingSpace"] = 160,
5793 ["nbsp"] = 160,
5794 ["iexcl"] = 161,
5795 ["cent"] = 162,
5796 ["pound"] = 163,
5797 ["curren"] = 164,
5798 ["yen"] = 165,
5799 ["brvbar"] = 166,
5800 ["sect"] = 167,
5801 ["Dot"] = 168,
5802 ["DoubleDot"] = 168,
5803 ["die"] = 168,
5804 ["uml"] = 168,
5805 ["COPY"] = 169,
5806 ["copy"] = 169,
5807 ["ordf"] = 170,
5808 ["laquo"] = 171,
5809 ["not"] = 172,
5810 ["shy"] = 173,
5811 ["REG"] = 174,
5812 ["circledR"] = 174,
5813 ["reg"] = 174,
5814 ["macr"] = 175,
5815 ["strns"] = 175,
5816 ["deg"] = 176,
5817 ["PlusMinus"] = 177,
5818 ["plusmn"] = 177,
5819 ["pm"] = 177,
5820 ["sup2"] = 178,
5821 ["sup3"] = 179,
5822 ["DiacriticalAcute"] = 180,
5823 ["acute"] = 180,
5824 ["micro"] = 181,
5825 ["para"] = 182,
5826 ["CenterDot"] = 183,
5827 ["centerdot"] = 183,
5828 ["middot"] = 183,

```



5829 ["Cedilla"] = 184,  
 5830 ["cedil"] = 184,  
 5831 ["sup1"] = 185,  
 5832 ["ordm"] = 186,  
 5833 ["raquo"] = 187,  
 5834 ["frac14"] = 188,  
 5835 ["frac12"] = 189,  
 5836 ["half"] = 189,  
 5837 ["frac34"] = 190,  
 5838 ["iquest"] = 191,  
 5839 ["Agrave"] = 192,  
 5840 ["Aacute"] = 193,  
 5841 ["Acirc"] = 194,  
 5842 ["Atilde"] = 195,  
 5843 ["Auml"] = 196,  
 5844 ["Aring"] = 197,  
 5845 ["angst"] = 197,  
 5846 ["AElig"] = 198,  
 5847 ["Ccedil"] = 199,  
 5848 ["Egrave"] = 200,  
 5849 ["Eacute"] = 201,  
 5850 ["Ecirc"] = 202,  
 5851 ["Euml"] = 203,  
 5852 ["Igrave"] = 204,  
 5853 ["Iacute"] = 205,  
 5854 ["Icirc"] = 206,  
 5855 ["Iuml"] = 207,  
 5856 ["ETH"] = 208,  
 5857 ["Ntilde"] = 209,  
 5858 ["Ograve"] = 210,  
 5859 ["Oacute"] = 211,  
 5860 ["Ocirc"] = 212,  
 5861 ["Otilde"] = 213,  
 5862 ["Ouml"] = 214,  
 5863 ["times"] = 215,  
 5864 ["Oslash"] = 216,  
 5865 ["Ugrave"] = 217,  
 5866 ["Uacute"] = 218,  
 5867 ["Ucirc"] = 219,  
 5868 ["Uuml"] = 220,  
 5869 ["Yacute"] = 221,  
 5870 ["THORN"] = 222,  
 5871 ["szlig"] = 223,  
 5872 ["agrave"] = 224,  
 5873 ["aacute"] = 225,  
 5874 ["acirc"] = 226,  
 5875 ["atilde"] = 227,

```

5876 ["auml"] = 228,
5877 ["aring"] = 229,
5878 ["aelig"] = 230,
5879 ["ccedil"] = 231,
5880 ["egrave"] = 232,
5881 ["eacute"] = 233,
5882 ["ecirc"] = 234,
5883 ["euml"] = 235,
5884 ["igrave"] = 236,
5885 ["iacute"] = 237,
5886 ["icirc"] = 238,
5887 ["iuml"] = 239,
5888 ["eth"] = 240,
5889 ["ntilde"] = 241,
5890 ["ograve"] = 242,
5891 ["oacute"] = 243,
5892 ["ocirc"] = 244,
5893 ["otilde"] = 245,
5894 ["ouml"] = 246,
5895 ["div"] = 247,
5896 ["divide"] = 247,
5897 ["oslash"] = 248,
5898 ["ugrave"] = 249,
5899 ["uacute"] = 250,
5900 ["ucirc"] = 251,
5901 ["uuml"] = 252,
5902 ["yacute"] = 253,
5903 ["thorn"] = 254,
5904 ["yuml"] = 255,
5905 ["Aacr"] = 256,
5906 ["aacr"] = 257,
5907 ["Abreve"] = 258,
5908 ["abreve"] = 259,
5909 ["Aogon"] = 260,
5910 ["aogon"] = 261,
5911 ["Cacute"] = 262,
5912 ["cacute"] = 263,
5913 ["Ccirc"] = 264,
5914 ["ccirc"] = 265,
5915 ["Cdot"] = 266,
5916 ["cdot"] = 267,
5917 ["Ccaron"] = 268,
5918 ["ccaron"] = 269,
5919 ["Dcaron"] = 270,
5920 ["dcaron"] = 271,
5921 ["Dstrok"] = 272,
5922 ["dstrok"] = 273,

```

```

5923 ["Emacr"] = 274,
5924 ["emacr"] = 275,
5925 ["Edot"] = 278,
5926 ["edot"] = 279,
5927 ["Eogon"] = 280,
5928 ["eogon"] = 281,
5929 ["Ecaron"] = 282,
5930 ["ecaron"] = 283,
5931 ["Gcirc"] = 284,
5932 ["gcirc"] = 285,
5933 ["Gbreve"] = 286,
5934 ["gbreve"] = 287,
5935 ["Gdot"] = 288,
5936 ["gdot"] = 289,
5937 ["Gcedil"] = 290,
5938 ["Hcirc"] = 292,
5939 ["hcirc"] = 293,
5940 ["Hstrokr"] = 294,
5941 ["hstrokr"] = 295,
5942 ["Itilde"] = 296,
5943 ["itilde"] = 297,
5944 ["Imacr"] = 298,
5945 ["imacr"] = 299,
5946 ["Iogon"] = 302,
5947 ["iogon"] = 303,
5948 ["Idot"] = 304,
5949 ["imath"] = 305,
5950 ["inodot"] = 305,
5951 ["IJlig"] = 306,
5952 ["ijlig"] = 307,
5953 ["Jcirc"] = 308,
5954 ["jcirc"] = 309,
5955 ["Kcedil"] = 310,
5956 ["kcedil"] = 311,
5957 ["kgreen"] = 312,
5958 ["Lacute"] = 313,
5959 ["lacute"] = 314,
5960 ["Lcedil"] = 315,
5961 ["lcedil"] = 316,
5962 ["Lcaron"] = 317,
5963 ["lcaron"] = 318,
5964 ["Lmidot"] = 319,
5965 ["lmidot"] = 320,
5966 ["Lstrokr"] = 321,
5967 ["lstrokr"] = 322,
5968 ["Nacute"] = 323,
5969 ["nacute"] = 324,

```

5970 ["Ncedil"] = 325,  
 5971 ["ncedil"] = 326,  
 5972 ["Ncaron"] = 327,  
 5973 ["ncaron"] = 328,  
 5974 ["napos"] = 329,  
 5975 ["ENG"] = 330,  
 5976 ["eng"] = 331,  
 5977 ["Omacr"] = 332,  
 5978 ["omacr"] = 333,  
 5979 ["Odblac"] = 336,  
 5980 ["odblac"] = 337,  
 5981 ["OElig"] = 338,  
 5982 ["oelig"] = 339,  
 5983 ["Racute"] = 340,  
 5984 ["racute"] = 341,  
 5985 ["Rcedil"] = 342,  
 5986 ["rcedil"] = 343,  
 5987 ["Rcaron"] = 344,  
 5988 ["rcaron"] = 345,  
 5989 ["Sacute"] = 346,  
 5990 ["sacute"] = 347,  
 5991 ["Scirc"] = 348,  
 5992 ["scirc"] = 349,  
 5993 ["Scedil"] = 350,  
 5994 ["scedil"] = 351,  
 5995 ["Scaron"] = 352,  
 5996 ["scaron"] = 353,  
 5997 ["Tcedil"] = 354,  
 5998 ["tcedil"] = 355,  
 5999 ["Tcaron"] = 356,  
 6000 ["tcaron"] = 357,  
 6001 ["Tstrok"] = 358,  
 6002 ["tstrok"] = 359,  
 6003 ["Utilde"] = 360,  
 6004 ["utilde"] = 361,  
 6005 ["Umacr"] = 362,  
 6006 ["umacr"] = 363,  
 6007 ["Ubreve"] = 364,  
 6008 ["ubreve"] = 365,  
 6009 ["Uring"] = 366,  
 6010 ["uring"] = 367,  
 6011 ["Udblac"] = 368,  
 6012 ["udblac"] = 369,  
 6013 ["Uogon"] = 370,  
 6014 ["uogon"] = 371,  
 6015 ["Wcirc"] = 372,  
 6016 ["wcirc"] = 373,

6017 ["Ycirc"] = 374,  
 6018 ["ycirc"] = 375,  
 6019 ["Yuml"] = 376,  
 6020 ["Zacute"] = 377,  
 6021 ["zacute"] = 378,  
 6022 ["Zdot"] = 379,  
 6023 ["zdot"] = 380,  
 6024 ["Zcaron"] = 381,  
 6025 ["zcaron"] = 382,  
 6026 ["fnof"] = 402,  
 6027 ["imped"] = 437,  
 6028 ["gacute"] = 501,  
 6029 ["jmath"] = 567,  
 6030 ["circ"] = 710,  
 6031 ["Hacek"] = 711,  
 6032 ["caron"] = 711,  
 6033 ["Breve"] = 728,  
 6034 ["breve"] = 728,  
 6035 ["DiacriticalDot"] = 729,  
 6036 ["dot"] = 729,  
 6037 ["ring"] = 730,  
 6038 ["ogon"] = 731,  
 6039 ["DiacriticalTilde"] = 732,  
 6040 ["tilde"] = 732,  
 6041 ["DiacriticalDoubleAcute"] = 733,  
 6042 ["dblac"] = 733,  
 6043 ["DownBreve"] = 785,  
 6044 ["Alpha"] = 913,  
 6045 ["Beta"] = 914,  
 6046 ["Gamma"] = 915,  
 6047 ["Delta"] = 916,  
 6048 ["Epsilon"] = 917,  
 6049 ["Zeta"] = 918,  
 6050 ["Eta"] = 919,  
 6051 ["Theta"] = 920,  
 6052 ["Iota"] = 921,  
 6053 ["Kappa"] = 922,  
 6054 ["Lambda"] = 923,  
 6055 ["Mu"] = 924,  
 6056 ["Nu"] = 925,  
 6057 ["Xi"] = 926,  
 6058 ["Omicron"] = 927,  
 6059 ["Pi"] = 928,  
 6060 ["Rho"] = 929,  
 6061 ["Sigma"] = 931,  
 6062 ["Tau"] = 932,  
 6063 ["Upsilon"] = 933,

```

6064 ["Phi"] = 934,
6065 ["Chi"] = 935,
6066 ["Psi"] = 936,
6067 ["Omega"] = 937,
6068 ["ohm"] = 937,
6069 ["alpha"] = 945,
6070 ["beta"] = 946,
6071 ["gamma"] = 947,
6072 ["delta"] = 948,
6073 ["epsi"] = 949,
6074 ["epsilon"] = 949,
6075 ["zeta"] = 950,
6076 ["eta"] = 951,
6077 ["theta"] = 952,
6078 ["iota"] = 953,
6079 ["kappa"] = 954,
6080 ["lambda"] = 955,
6081 ["mu"] = 956,
6082 ["nu"] = 957,
6083 ["xi"] = 958,
6084 ["omicron"] = 959,
6085 ["pi"] = 960,
6086 ["rho"] = 961,
6087 ["sigmaf"] = 962,
6088 ["sigmav"] = 962,
6089 ["varsigma"] = 962,
6090 ["sigma"] = 963,
6091 ["tau"] = 964,
6092 ["upsi"] = 965,
6093 ["upsilon"] = 965,
6094 ["phi"] = 966,
6095 ["chi"] = 967,
6096 ["psi"] = 968,
6097 ["omega"] = 969,
6098 ["thetasym"] = 977,
6099 ["thetav"] = 977,
6100 ["vartheta"] = 977,
6101 ["Upsi"] = 978,
6102 ["upsih"] = 978,
6103 ["phiv"] = 981,
6104 ["straightphi"] = 981,
6105 ["varphi"] = 981,
6106 ["piv"] = 982,
6107 ["varpi"] = 982,
6108 ["Gammad"] = 988,
6109 ["digamma"] = 989,
6110 ["gammad"] = 989,

```

```

6111 ["kappav"] = 1008,
6112 ["varkappa"] = 1008,
6113 ["rhov"] = 1009,
6114 ["varrho"] = 1009,
6115 ["epsiv"] = 1013,
6116 ["straightepsilon"] = 1013,
6117 ["varepsilon"] = 1013,
6118 ["backepsilon"] = 1014,
6119 ["bepsi"] = 1014,
6120 ["IOcy"] = 1025,
6121 ["DJcy"] = 1026,
6122 ["GJcy"] = 1027,
6123 ["Jukcy"] = 1028,
6124 ["DScy"] = 1029,
6125 ["Iukcy"] = 1030,
6126 ["YIcy"] = 1031,
6127 ["Jsercy"] = 1032,
6128 ["LJcy"] = 1033,
6129 ["NJcy"] = 1034,
6130 ["TSHcy"] = 1035,
6131 ["KJcy"] = 1036,
6132 ["Ubrcy"] = 1038,
6133 ["DZcy"] = 1039,
6134 ["Acy"] = 1040,
6135 ["Bcy"] = 1041,
6136 ["Vcy"] = 1042,
6137 ["Gcy"] = 1043,
6138 ["Dcy"] = 1044,
6139 ["IEcy"] = 1045,
6140 ["ZHcy"] = 1046,
6141 ["Zcy"] = 1047,
6142 ["Icy"] = 1048,
6143 ["Jcy"] = 1049,
6144 ["Kcy"] = 1050,
6145 ["Lcy"] = 1051,
6146 ["Mcy"] = 1052,
6147 ["Ncy"] = 1053,
6148 ["Ocy"] = 1054,
6149 ["Pcy"] = 1055,
6150 ["Rcy"] = 1056,
6151 ["Scy"] = 1057,
6152 ["Tcy"] = 1058,
6153 ["Ucy"] = 1059,
6154 ["Fcy"] = 1060,
6155 ["KHcy"] = 1061,
6156 ["TScy"] = 1062,
6157 ["CHcy"] = 1063,

```

```

6158 ["SHcy"] = 1064,
6159 ["SHCHcy"] = 1065,
6160 ["HARDcy"] = 1066,
6161 ["Ycy"] = 1067,
6162 ["SOFTcy"] = 1068,
6163 ["Ecy"] = 1069,
6164 ["YUcy"] = 1070,
6165 ["YAcy"] = 1071,
6166 ["acy"] = 1072,
6167 ["bcy"] = 1073,
6168 ["vcy"] = 1074,
6169 ["gcy"] = 1075,
6170 ["dcy"] = 1076,
6171 ["iecy"] = 1077,
6172 ["zhcy"] = 1078,
6173 ["zcy"] = 1079,
6174 ["icy"] = 1080,
6175 ["jcy"] = 1081,
6176 ["kcy"] = 1082,
6177 ["lcy"] = 1083,
6178 ["mcy"] = 1084,
6179 ["ncy"] = 1085,
6180 ["ocy"] = 1086,
6181 ["pcy"] = 1087,
6182 ["rcy"] = 1088,
6183 ["scy"] = 1089,
6184 ["tcy"] = 1090,
6185 ["ucy"] = 1091,
6186 ["fcy"] = 1092,
6187 ["khcy"] = 1093,
6188 ["tscy"] = 1094,
6189 ["chcy"] = 1095,
6190 ["shcy"] = 1096,
6191 ["shchcy"] = 1097,
6192 ["hardcy"] = 1098,
6193 ["ycy"] = 1099,
6194 ["softcy"] = 1100,
6195 ["ecy"] = 1101,
6196 ["yucy"] = 1102,
6197 ["yacy"] = 1103,
6198 ["iocy"] = 1105,
6199 ["djcy"] = 1106,
6200 ["gjcy"] = 1107,
6201 ["jukcy"] = 1108,
6202 ["dscy"] = 1109,
6203 ["iukcy"] = 1110,
6204 ["yicy"] = 1111,

```



```

6205 ["jsercy"] = 1112,
6206 ["ljcy"] = 1113,
6207 ["njcy"] = 1114,
6208 ["tshcy"] = 1115,
6209 ["kjcy"] = 1116,
6210 ["ubrcy"] = 1118,
6211 ["dzcyc"] = 1119,
6212 ["ensp"] = 8194,
6213 ["emsp"] = 8195,
6214 ["emsp13"] = 8196,
6215 ["emsp14"] = 8197,
6216 ["numsp"] = 8199,
6217 ["puncsp"] = 8200,
6218 ["ThinSpace"] = 8201,
6219 ["thinsp"] = 8201,
6220 ["VeryThinSpace"] = 8202,
6221 ["hairsp"] = 8202,
6222 ["NegativeMediumSpace"] = 8203,
6223 ["NegativeThickSpace"] = 8203,
6224 ["NegativeThinSpace"] = 8203,
6225 ["NegativeVeryThinSpace"] = 8203,
6226 ["ZeroWidthSpace"] = 8203,
6227 ["zwnj"] = 8204,
6228 ["zwj"] = 8205,
6229 ["lrm"] = 8206,
6230 ["rlm"] = 8207,
6231 ["dash"] = 8208,
6232 ["hyphen"] = 8208,
6233 ["ndash"] = 8211,
6234 ["mdash"] = 8212,
6235 ["horbar"] = 8213,
6236 ["Verbar"] = 8214,
6237 ["Vert"] = 8214,
6238 ["OpenCurlyQuote"] = 8216,
6239 ["lsquo"] = 8216,
6240 ["CloseCurlyQuote"] = 8217,
6241 ["rsquo"] = 8217,
6242 ["rsquor"] = 8217,
6243 ["lsquor"] = 8218,
6244 ["sbquo"] = 8218,
6245 ["OpenCurlyDoubleQuote"] = 8220,
6246 ["ldquo"] = 8220,
6247 ["CloseCurlyDoubleQuote"] = 8221,
6248 ["rdquo"] = 8221,
6249 ["rdquor"] = 8221,
6250 ["bdquo"] = 8222,
6251 ["ldquor"] = 8222,

```

```

6252 ["dagger"] = 8224,
6253 ["Dagger"] = 8225,
6254 ["ddagger"] = 8225,
6255 ["bull"] = 8226,
6256 ["bullet"] = 8226,
6257 ["nldr"] = 8229,
6258 ["hellip"] = 8230,
6259 ["mldr"] = 8230,
6260 ["permil"] = 8240,
6261 ["pertenk"] = 8241,
6262 ["prime"] = 8242,
6263 ["Prime"] = 8243,
6264 ["tprime"] = 8244,
6265 ["backprime"] = 8245,
6266 ["bprime"] = 8245,
6267 ["lsaquo"] = 8249,
6268 ["rsaquo"] = 8250,
6269 ["OverBar"] = 8254,
6270 ["oline"] = 8254,
6271 ["caret"] = 8257,
6272 ["hybull"] = 8259,
6273 ["frasl"] = 8260,
6274 ["bsemi"] = 8271,
6275 ["qprime"] = 8279,
6276 ["MediumSpace"] = 8287,
6277 ["ThickSpace"] = {8287, 8202},
6278 ["NoBreak"] = 8288,
6279 ["ApplyFunction"] = 8289,
6280 ["af"] = 8289,
6281 ["InvisibleTimes"] = 8290,
6282 ["it"] = 8290,
6283 ["InvisibleComma"] = 8291,
6284 ["ic"] = 8291,
6285 ["euro"] = 8364,
6286 ["TripleDot"] = 8411,
6287 ["tdot"] = 8411,
6288 ["DotDot"] = 8412,
6289 ["Copf"] = 8450,
6290 ["complexes"] = 8450,
6291 ["incare"] = 8453,
6292 ["gscr"] = 8458,
6293 ["HilbertSpace"] = 8459,
6294 ["Hscr"] = 8459,
6295 ["hamilt"] = 8459,
6296 ["Hfr"] = 8460,
6297 ["Poincareplane"] = 8460,
6298 ["Hopf"] = 8461,

```

```

6299 ["quaternions"] = 8461,
6300 ["planckh"] = 8462,
6301 ["hbar"] = 8463,
6302 ["hslash"] = 8463,
6303 ["planck"] = 8463,
6304 ["plankv"] = 8463,
6305 ["Iscr"] = 8464,
6306 ["imagline"] = 8464,
6307 ["Ifr"] = 8465,
6308 ["Im"] = 8465,
6309 ["image"] = 8465,
6310 ["imagpart"] = 8465,
6311 ["Laplacetrfr"] = 8466,
6312 ["Lscr"] = 8466,
6313 ["lagran"] = 8466,
6314 ["ell"] = 8467,
6315 ["Nopf"] = 8469,
6316 ["naturals"] = 8469,
6317 ["numero"] = 8470,
6318 ["copysr"] = 8471,
6319 ["weierp"] = 8472,
6320 ["wp"] = 8472,
6321 ["Popf"] = 8473,
6322 ["primes"] = 8473,
6323 ["Qopf"] = 8474,
6324 ["rationals"] = 8474,
6325 ["Rscr"] = 8475,
6326 ["realine"] = 8475,
6327 ["Re"] = 8476,
6328 ["Rfr"] = 8476,
6329 ["real"] = 8476,
6330 ["realpart"] = 8476,
6331 ["Ropf"] = 8477,
6332 ["reals"] = 8477,
6333 ["rx"] = 8478,
6334 ["TRADE"] = 8482,
6335 ["trade"] = 8482,
6336 ["Zopf"] = 8484,
6337 ["integers"] = 8484,
6338 ["mho"] = 8487,
6339 ["Zfr"] = 8488,
6340 ["zeetrfr"] = 8488,
6341 ["iiota"] = 8489,
6342 ["Bernoullis"] = 8492,
6343 ["Bscr"] = 8492,
6344 ["bernou"] = 8492,
6345 ["Cayleys"] = 8493,

```

```

6346 ["Cfr"] = 8493,
6347 ["escr"] = 8495,
6348 ["Escr"] = 8496,
6349 ["expectation"] = 8496,
6350 ["Fouriertrf"] = 8497,
6351 ["Fscr"] = 8497,
6352 ["Mellintrf"] = 8499,
6353 ["Mscr"] = 8499,
6354 ["phmmat"] = 8499,
6355 ["order"] = 8500,
6356 ["orderof"] = 8500,
6357 ["oscr"] = 8500,
6358 ["alefsym"] = 8501,
6359 ["aleph"] = 8501,
6360 ["beth"] = 8502,
6361 ["gimel"] = 8503,
6362 ["daleth"] = 8504,
6363 ["CapitalDifferentialD"] = 8517,
6364 ["DD"] = 8517,
6365 ["DifferentialD"] = 8518,
6366 ["dd"] = 8518,
6367 ["ExponentialE"] = 8519,
6368 ["ee"] = 8519,
6369 ["exponentiale"] = 8519,
6370 ["ImaginaryI"] = 8520,
6371 ["ii"] = 8520,
6372 ["frac13"] = 8531,
6373 ["frac23"] = 8532,
6374 ["frac15"] = 8533,
6375 ["frac25"] = 8534,
6376 ["frac35"] = 8535,
6377 ["frac45"] = 8536,
6378 ["frac16"] = 8537,
6379 ["frac56"] = 8538,
6380 ["frac18"] = 8539,
6381 ["frac38"] = 8540,
6382 ["frac58"] = 8541,
6383 ["frac78"] = 8542,
6384 ["LeftArrow"] = 8592,
6385 ["ShortLeftArrow"] = 8592,
6386 ["larr"] = 8592,
6387 ["leftarrow"] = 8592,
6388 ["slarr"] = 8592,
6389 ["ShortUpArrow"] = 8593,
6390 ["UpArrow"] = 8593,
6391 ["uarr"] = 8593,
6392 ["uparrow"] = 8593,

```

```

6393 ["RightArrow"] = 8594,
6394 ["ShortRightArrow"] = 8594,
6395 ["rarr"] = 8594,
6396 ["rightarrow"] = 8594,
6397 ["srarr"] = 8594,
6398 ["DownArrow"] = 8595,
6399 ["ShortDownArrow"] = 8595,
6400 ["darr"] = 8595,
6401 ["downarrow"] = 8595,
6402 ["LeftRightArrow"] = 8596,
6403 ["harr"] = 8596,
6404 ["leftrightarrow"] = 8596,
6405 ["UpDownArrow"] = 8597,
6406 ["updownarrow"] = 8597,
6407 ["varr"] = 8597,
6408 ["UpperLeftArrow"] = 8598,
6409 ["nwarr"] = 8598,
6410 ["nwarrow"] = 8598,
6411 ["UpperRightArrow"] = 8599,
6412 ["nearr"] = 8599,
6413 ["nearrow"] = 8599,
6414 ["LowerRightArrow"] = 8600,
6415 ["searr"] = 8600,
6416 ["searrow"] = 8600,
6417 ["LowerLeftArrow"] = 8601,
6418 ["swarr"] = 8601,
6419 ["swarrow"] = 8601,
6420 ["nlarr"] = 8602,
6421 ["nleftarrow"] = 8602,
6422 ["nrarr"] = 8603,
6423 ["nrightarrow"] = 8603,
6424 ["nrarrw"] = {8605, 824},
6425 ["rarrw"] = 8605,
6426 ["rightsquigarrow"] = 8605,
6427 ["Larr"] = 8606,
6428 ["twoheadleftarrow"] = 8606,
6429 ["Uarr"] = 8607,
6430 ["Rarr"] = 8608,
6431 ["twoheadrightarrow"] = 8608,
6432 ["Darr"] = 8609,
6433 ["larrtl"] = 8610,
6434 ["leftarrowtail"] = 8610,
6435 ["rarrtl"] = 8611,
6436 ["rightarrowtail"] = 8611,
6437 ["LeftTeeArrow"] = 8612,
6438 ["mapstoleft"] = 8612,
6439 ["UpTeeArrow"] = 8613,

```

```

6440 ["mapstoup"] = 8613,
6441 ["RightTeeArrow"] = 8614,
6442 ["map"] = 8614,
6443 ["mapsto"] = 8614,
6444 ["DownTeeArrow"] = 8615,
6445 ["mapstodown"] = 8615,
6446 ["hookleftarrow"] = 8617,
6447 ["larrhk"] = 8617,
6448 ["hookrightarrow"] = 8618,
6449 ["rarrhk"] = 8618,
6450 ["larrlp"] = 8619,
6451 ["looparrowleft"] = 8619,
6452 ["looparrowright"] = 8620,
6453 ["rarrlp"] = 8620,
6454 ["harrw"] = 8621,
6455 ["leftrightsquigarrow"] = 8621,
6456 ["nharr"] = 8622,
6457 ["nletrightarrow"] = 8622,
6458 ["Lsh"] = 8624,
6459 ["lsh"] = 8624,
6460 ["Rsh"] = 8625,
6461 ["rsh"] = 8625,
6462 ["ldsh"] = 8626,
6463 ["rdsh"] = 8627,
6464 ["crarr"] = 8629,
6465 ["cularr"] = 8630,
6466 ["curvearrowleft"] = 8630,
6467 ["curarr"] = 8631,
6468 ["curvearrowright"] = 8631,
6469 ["circlearrowleft"] = 8634,
6470 ["olarr"] = 8634,
6471 ["circlearrowright"] = 8635,
6472 ["orarr"] = 8635,
6473 ["LeftVector"] = 8636,
6474 ["leftharpoonup"] = 8636,
6475 ["lharu"] = 8636,
6476 ["DownLeftVector"] = 8637,
6477 ["leftharpoondown"] = 8637,
6478 ["lhard"] = 8637,
6479 ["RightUpVector"] = 8638,
6480 ["uharr"] = 8638,
6481 ["upharpoonright"] = 8638,
6482 ["LeftUpVector"] = 8639,
6483 ["uharl"] = 8639,
6484 ["upharpoonleft"] = 8639,
6485 ["RightVector"] = 8640,
6486 ["rharu"] = 8640,

```

```

6487 ["rightharpoonup"] = 8640,
6488 ["DownRightVector"] = 8641,
6489 ["rhard"] = 8641,
6490 ["rightharpoondown"] = 8641,
6491 ["RightDownVector"] = 8642,
6492 ["dharr"] = 8642,
6493 ["downharpoonright"] = 8642,
6494 ["LeftDownVector"] = 8643,
6495 ["dharl"] = 8643,
6496 ["downharpoonleft"] = 8643,
6497 ["RightArrowLeftArrow"] = 8644,
6498 ["rightleftarrows"] = 8644,
6499 ["rlarr"] = 8644,
6500 ["UpArrowDownArrow"] = 8645,
6501 ["udarr"] = 8645,
6502 ["LeftArrowRightArrow"] = 8646,
6503 ["leftrightarrows"] = 8646,
6504 ["lrarr"] = 8646,
6505 ["leftleftarrows"] = 8647,
6506 ["llarr"] = 8647,
6507 ["upuparrows"] = 8648,
6508 ["uuarr"] = 8648,
6509 ["rightrightarrows"] = 8649,
6510 ["rrarr"] = 8649,
6511 ["ddarr"] = 8650,
6512 ["downdownarrows"] = 8650,
6513 ["ReverseEquilibrium"] = 8651,
6514 ["leftrightharpoons"] = 8651,
6515 ["lrhar"] = 8651,
6516 ["Equilibrium"] = 8652,
6517 ["rightleftharpoons"] = 8652,
6518 ["rlhar"] = 8652,
6519 ["nLeftarrow"] = 8653,
6520 ["nLArr"] = 8653,
6521 ["nLeftrightarrow"] = 8654,
6522 ["nhArr"] = 8654,
6523 ["nRightarrow"] = 8655,
6524 ["nrArr"] = 8655,
6525 ["DoubleLeftArrow"] = 8656,
6526 ["Leftarrow"] = 8656,
6527 ["lArr"] = 8656,
6528 ["DoubleUpArrow"] = 8657,
6529 ["Uparrow"] = 8657,
6530 ["uArr"] = 8657,
6531 ["DoubleRightArrow"] = 8658,
6532 ["Implies"] = 8658,
6533 ["Rightarrow"] = 8658,

```

```

6534 ["rArr"] = 8658,
6535 ["DoubleDownArrow"] = 8659,
6536 ["Downarrow"] = 8659,
6537 ["dArr"] = 8659,
6538 ["DoubleLeftRightArrow"] = 8660,
6539 ["Leftrightarrow"] = 8660,
6540 ["hArr"] = 8660,
6541 ["iff"] = 8660,
6542 ["DoubleUpDownArrow"] = 8661,
6543 ["Updownarrow"] = 8661,
6544 ["vArr"] = 8661,
6545 ["nwArr"] = 8662,
6546 ["neArr"] = 8663,
6547 ["seArr"] = 8664,
6548 ["swArr"] = 8665,
6549 ["Lleftarrow"] = 8666,
6550 ["lAarr"] = 8666,
6551 ["Rrightarrow"] = 8667,
6552 ["rAarr"] = 8667,
6553 ["zigrarr"] = 8669,
6554 ["LeftArrowBar"] = 8676,
6555 ["larrb"] = 8676,
6556 ["RightArrowBar"] = 8677,
6557 ["rarrb"] = 8677,
6558 ["DownArrowUpArrow"] = 8693,
6559 ["duarr"] = 8693,
6560 ["loarr"] = 8701,
6561 ["roarr"] = 8702,
6562 ["hoarr"] = 8703,
6563 ["ForAll"] = 8704,
6564 ["forall"] = 8704,
6565 ["comp"] = 8705,
6566 ["complement"] = 8705,
6567 ["PartialD"] = 8706,
6568 ["npart"] = {8706, 824},
6569 ["part"] = 8706,
6570 ["Exists"] = 8707,
6571 ["exist"] = 8707,
6572 ["NotExists"] = 8708,
6573 ["nexist"] = 8708,
6574 ["nexists"] = 8708,
6575 ["empty"] = 8709,
6576 ["emptyset"] = 8709,
6577 ["emptyv"] = 8709,
6578 ["varnothing"] = 8709,
6579 ["Del"] = 8711,
6580 ["nabla"] = 8711,

```



```

6581 ["Element"] = 8712,
6582 ["in"] = 8712,
6583 ["isin"] = 8712,
6584 ["isinv"] = 8712,
6585 ["NotElement"] = 8713,
6586 ["notin"] = 8713,
6587 ["notinva"] = 8713,
6588 ["ReverseElement"] = 8715,
6589 ["SuchThat"] = 8715,
6590 ["ni"] = 8715,
6591 ["niv"] = 8715,
6592 ["NotReverseElement"] = 8716,
6593 ["notni"] = 8716,
6594 ["notniva"] = 8716,
6595 ["Product"] = 8719,
6596 ["prod"] = 8719,
6597 ["Coproduct"] = 8720,
6598 ["coprod"] = 8720,
6599 ["Sum"] = 8721,
6600 ["sum"] = 8721,
6601 ["minus"] = 8722,
6602 ["MinusPlus"] = 8723,
6603 ["mnplus"] = 8723,
6604 ["mp"] = 8723,
6605 ["dotplus"] = 8724,
6606 ["plusdo"] = 8724,
6607 ["Backslash"] = 8726,
6608 ["setminus"] = 8726,
6609 ["setmn"] = 8726,
6610 ["smallsetminus"] = 8726,
6611 ["ssetmn"] = 8726,
6612 ["lowast"] = 8727,
6613 ["SmallCircle"] = 8728,
6614 ["compfn"] = 8728,
6615 ["Sqrt"] = 8730,
6616 ["radic"] = 8730,
6617 ["Proportional"] = 8733,
6618 ["prop"] = 8733,
6619 ["propto"] = 8733,
6620 ["varpropto"] = 8733,
6621 ["vprop"] = 8733,
6622 ["infin"] = 8734,
6623 ["angrt"] = 8735,
6624 ["ang"] = 8736,
6625 ["angle"] = 8736,
6626 ["nang"] = {8736, 8402},
6627 ["angmsd"] = 8737,

```

```

6628 ["measuredangle"] = 8737,
6629 ["angsph"] = 8738,
6630 ["VerticalBar"] = 8739,
6631 ["mid"] = 8739,
6632 ["shortmid"] = 8739,
6633 ["smid"] = 8739,
6634 ["NotVerticalBar"] = 8740,
6635 ["nmid"] = 8740,
6636 ["nshortmid"] = 8740,
6637 ["nsmid"] = 8740,
6638 ["DoubleVerticalBar"] = 8741,
6639 ["par"] = 8741,
6640 ["parallel"] = 8741,
6641 ["shortparallel"] = 8741,
6642 ["spar"] = 8741,
6643 ["NotDoubleVerticalBar"] = 8742,
6644 ["npar"] = 8742,
6645 ["nparallel"] = 8742,
6646 ["nshortparallel"] = 8742,
6647 ["nspar"] = 8742,
6648 ["and"] = 8743,
6649 ["wedge"] = 8743,
6650 ["or"] = 8744,
6651 ["vee"] = 8744,
6652 ["cap"] = 8745,
6653 ["caps"] = {8745, 65024},
6654 ["cup"] = 8746,
6655 ["cups"] = {8746, 65024},
6656 ["Integral"] = 8747,
6657 ["int"] = 8747,
6658 ["Int"] = 8748,
6659 ["iiint"] = 8749,
6660 ["tint"] = 8749,
6661 ["ContourIntegral"] = 8750,
6662 ["conint"] = 8750,
6663 ["oint"] = 8750,
6664 ["Conint"] = 8751,
6665 ["DoubleContourIntegral"] = 8751,
6666 ["Cconint"] = 8752,
6667 ["cwint"] = 8753,
6668 ["ClockwiseContourIntegral"] = 8754,
6669 ["cwconint"] = 8754,
6670 ["CounterClockwiseContourIntegral"] = 8755,
6671 ["awconint"] = 8755,
6672 ["Therefore"] = 8756,
6673 ["there4"] = 8756,
6674 ["therefore"] = 8756,

```

```

6675 ["Because"] = 8757,
6676 ["becaus"] = 8757,
6677 ["because"] = 8757,
6678 ["ratio"] = 8758,
6679 ["Colon"] = 8759,
6680 ["Proportion"] = 8759,
6681 ["dotminus"] = 8760,
6682 ["minusd"] = 8760,
6683 ["mDDot"] = 8762,
6684 ["homtht"] = 8763,
6685 ["Tilde"] = 8764,
6686 ["nvsim"] = {8764, 8402},
6687 ["sim"] = 8764,
6688 ["thicksim"] = 8764,
6689 ["thksim"] = 8764,
6690 ["backsim"] = 8765,
6691 ["bsim"] = 8765,
6692 ["race"] = {8765, 817},
6693 ["ac"] = 8766,
6694 ["acE"] = {8766, 819},
6695 ["mstpos"] = 8766,
6696 ["acd"] = 8767,
6697 ["VerticalTilde"] = 8768,
6698 ["wr"] = 8768,
6699 ["wreath"] = 8768,
6700 ["NotTilde"] = 8769,
6701 ["nsim"] = 8769,
6702 ["EqualTilde"] = 8770,
6703 ["NotEqualTilde"] = {8770, 824},
6704 ["eqsim"] = 8770,
6705 ["esim"] = 8770,
6706 ["nesim"] = {8770, 824},
6707 ["TildeEqual"] = 8771,
6708 ["sime"] = 8771,
6709 ["simeq"] = 8771,
6710 ["NotTildeEqual"] = 8772,
6711 ["nsime"] = 8772,
6712 ["nsimeq"] = 8772,
6713 ["TildeFullEqual"] = 8773,
6714 ["cong"] = 8773,
6715 ["simne"] = 8774,
6716 ["NotTildeFullEqual"] = 8775,
6717 ["ncong"] = 8775,
6718 ["TildeTilde"] = 8776,
6719 ["ap"] = 8776,
6720 ["approx"] = 8776,
6721 ["asymp"] = 8776,

```

```

6722 ["thickapprox"] = 8776,
6723 ["thkap"] = 8776,
6724 ["NotTildeTilde"] = 8777,
6725 ["nap"] = 8777,
6726 ["napprox"] = 8777,
6727 ["ape"] = 8778,
6728 ["approxeq"] = 8778,
6729 ["apid"] = 8779,
6730 ["napid"] = {8779, 824},
6731 ["backcong"] = 8780,
6732 ["bcong"] = 8780,
6733 ["CupCap"] = 8781,
6734 ["asympeq"] = 8781,
6735 ["nvap"] = {8781, 8402},
6736 ["Bumpeq"] = 8782,
6737 ["HumpDownHump"] = 8782,
6738 ["NotHumpDownHump"] = {8782, 824},
6739 ["bump"] = 8782,
6740 ["nbump"] = {8782, 824},
6741 ["HumpEqual"] = 8783,
6742 ["NotHumpEqual"] = {8783, 824},
6743 ["bumpe"] = 8783,
6744 ["bumpeq"] = 8783,
6745 ["nbumpe"] = {8783, 824},
6746 ["DotEqual"] = 8784,
6747 ["doteq"] = 8784,
6748 ["esdot"] = 8784,
6749 ["nedot"] = {8784, 824},
6750 ["doteqdot"] = 8785,
6751 ["eDot"] = 8785,
6752 ["efDot"] = 8786,
6753 ["fallingdotseq"] = 8786,
6754 ["erDot"] = 8787,
6755 ["risingdotseq"] = 8787,
6756 ["Assign"] = 8788,
6757 ["colone"] = 8788,
6758 ["coloneq"] = 8788,
6759 ["ecolon"] = 8789,
6760 ["eqcolon"] = 8789,
6761 ["ecir"] = 8790,
6762 ["eqcirc"] = 8790,
6763 ["circeq"] = 8791,
6764 ["cire"] = 8791,
6765 ["wedgeq"] = 8793,
6766 ["veeeq"] = 8794,
6767 ["triangleq"] = 8796,
6768 ["trie"] = 8796,

```

```

6769 ["equest"] = 8799,
6770 ["questeq"] = 8799,
6771 ["NotEqual"] = 8800,
6772 ["ne"] = 8800,
6773 ["Congruent"] = 8801,
6774 ["bnequiv"] = {8801, 8421},
6775 ["equiv"] = 8801,
6776 ["NotCongruent"] = 8802,
6777 ["nequiv"] = 8802,
6778 ["le"] = 8804,
6779 ["leq"] = 8804,
6780 ["nvle"] = {8804, 8402},
6781 ["GreaterEqual"] = 8805,
6782 ["ge"] = 8805,
6783 ["geq"] = 8805,
6784 ["nvge"] = {8805, 8402},
6785 ["LessFullEqual"] = 8806,
6786 ["lE"] = 8806,
6787 ["leqq"] = 8806,
6788 ["nLE"] = {8806, 824},
6789 ["nleqq"] = {8806, 824},
6790 ["GreaterFullEqual"] = 8807,
6791 ["NotGreaterFullEqual"] = {8807, 824},
6792 ["gE"] = 8807,
6793 ["geqq"] = 8807,
6794 ["ngE"] = {8807, 824},
6795 ["ngeqq"] = {8807, 824},
6796 ["lnE"] = 8808,
6797 ["lneqq"] = 8808,
6798 ["lvertneqq"] = {8808, 65024},
6799 ["lvnE"] = {8808, 65024},
6800 ["gnE"] = 8809,
6801 ["gneqq"] = 8809,
6802 ["gvertneqq"] = {8809, 65024},
6803 ["gvnE"] = {8809, 65024},
6804 ["Lt"] = 8810,
6805 ["NestedLessLess"] = 8810,
6806 ["NotLessLess"] = {8810, 824},
6807 ["ll"] = 8810,
6808 ["nLt"] = {8810, 8402},
6809 ["nLtv"] = {8810, 824},
6810 ["Gt"] = 8811,
6811 ["NestedGreaterGreater"] = 8811,
6812 ["NotGreaterGreater"] = {8811, 824},
6813 ["gg"] = 8811,
6814 ["nGt"] = {8811, 8402},
6815 ["nGtv"] = {8811, 824},

```

```

6816 ["between"] = 8812,
6817 ["twixt"] = 8812,
6818 ["NotCupCap"] = 8813,
6819 ["NotLess"] = 8814,
6820 ["nless"] = 8814,
6821 ["nlt"] = 8814,
6822 ["NotGreater"] = 8815,
6823 ["ngt"] = 8815,
6824 ["ngtr"] = 8815,
6825 ["NotLessEqual"] = 8816,
6826 ["nle"] = 8816,
6827 ["nleq"] = 8816,
6828 ["NotGreaterEqual"] = 8817,
6829 ["nge"] = 8817,
6830 ["ngeq"] = 8817,
6831 ["LessTilde"] = 8818,
6832 ["lesssim"] = 8818,
6833 ["lsim"] = 8818,
6834 ["GreaterTilde"] = 8819,
6835 ["gsim"] = 8819,
6836 ["gtrsim"] = 8819,
6837 ["NotLessTilde"] = 8820,
6838 ["nlsim"] = 8820,
6839 ["NotGreaterTilde"] = 8821,
6840 ["ngsim"] = 8821,
6841 ["LessGreater"] = 8822,
6842 ["lessgtr"] = 8822,
6843 ["lg"] = 8822,
6844 ["GreaterLess"] = 8823,
6845 ["gl"] = 8823,
6846 ["gtrless"] = 8823,
6847 ["NotLessGreater"] = 8824,
6848 ["ntlg"] = 8824,
6849 ["NotGreaterLess"] = 8825,
6850 ["ntgl"] = 8825,
6851 ["Precedes"] = 8826,
6852 ["pr"] = 8826,
6853 ["prec"] = 8826,
6854 ["Succeeds"] = 8827,
6855 ["sc"] = 8827,
6856 ["succ"] = 8827,
6857 ["PrecedesSlantEqual"] = 8828,
6858 ["prcue"] = 8828,
6859 ["preccurlyeq"] = 8828,
6860 ["SucceedsSlantEqual"] = 8829,
6861 ["sccue"] = 8829,
6862 ["succcurlyeq"] = 8829,

```

```

6863 ["PrecedesTilde"] = 8830,
6864 ["precsim"] = 8830,
6865 ["prsim"] = 8830,
6866 ["NotSucceedsTilde"] = {8831, 824},
6867 ["SucceedsTilde"] = 8831,
6868 ["scsim"] = 8831,
6869 ["succsim"] = 8831,
6870 ["NotPrecedes"] = 8832,
6871 ["npr"] = 8832,
6872 ["nprec"] = 8832,
6873 ["NotSucceeds"] = 8833,
6874 ["nsc"] = 8833,
6875 ["nsucc"] = 8833,
6876 ["NotSubset"] = {8834, 8402},
6877 ["nsubset"] = {8834, 8402},
6878 ["sub"] = 8834,
6879 ["subset"] = 8834,
6880 ["vnsup"] = {8834, 8402},
6881 ["NotSuperset"] = {8835, 8402},
6882 ["Superset"] = 8835,
6883 ["nsupset"] = {8835, 8402},
6884 ["sup"] = 8835,
6885 ["supset"] = 8835,
6886 ["vnsup"] = {8835, 8402},
6887 ["nsub"] = 8836,
6888 ["nsup"] = 8837,
6889 ["SubsetEqual"] = 8838,
6890 ["sube"] = 8838,
6891 ["subseteq"] = 8838,
6892 ["SupersetEqual"] = 8839,
6893 ["supe"] = 8839,
6894 ["supseteq"] = 8839,
6895 ["NotSubsetEqual"] = 8840,
6896 ["nsube"] = 8840,
6897 ["nsubseteq"] = 8840,
6898 ["NotSupersetEqual"] = 8841,
6899 ["nsupe"] = 8841,
6900 ["nsupseteq"] = 8841,
6901 ["subne"] = 8842,
6902 ["subsetneq"] = 8842,
6903 ["varsubsetneq"] = {8842, 65024},
6904 ["vsubne"] = {8842, 65024},
6905 ["supne"] = 8843,
6906 ["supsetneq"] = 8843,
6907 ["varsupsetneq"] = {8843, 65024},
6908 ["vsupne"] = {8843, 65024},
6909 ["cupdot"] = 8845,

```

```

6910 ["UnionPlus"] = 8846,
6911 ["uplus"] = 8846,
6912 ["NotSquareSubset"] = {8847, 824},
6913 ["SquareSubset"] = 8847,
6914 ["sqsub"] = 8847,
6915 ["sqsubset"] = 8847,
6916 ["NotSquareSuperset"] = {8848, 824},
6917 ["SquareSuperset"] = 8848,
6918 ["sqsup"] = 8848,
6919 ["sqsupset"] = 8848,
6920 ["SquareSubsetEqual"] = 8849,
6921 ["sqsube"] = 8849,
6922 ["sqsubseteq"] = 8849,
6923 ["SquareSupersetEqual"] = 8850,
6924 ["sqsupe"] = 8850,
6925 ["sqsupseteq"] = 8850,
6926 ["SquareIntersection"] = 8851,
6927 ["sqcap"] = 8851,
6928 ["sqcaps"] = {8851, 65024},
6929 ["SquareUnion"] = 8852,
6930 ["sqcup"] = 8852,
6931 ["sqcups"] = {8852, 65024},
6932 ["CirclePlus"] = 8853,
6933 ["oplus"] = 8853,
6934 ["CircleMinus"] = 8854,
6935 ["ominus"] = 8854,
6936 ["CircleTimes"] = 8855,
6937 ["otimes"] = 8855,
6938 ["osol"] = 8856,
6939 ["CircleDot"] = 8857,
6940 ["odot"] = 8857,
6941 ["circledcirc"] = 8858,
6942 ["ocir"] = 8858,
6943 ["circledast"] = 8859,
6944 ["oast"] = 8859,
6945 ["circleddash"] = 8861,
6946 ["odash"] = 8861,
6947 ["boxplus"] = 8862,
6948 ["plusb"] = 8862,
6949 ["boxminus"] = 8863,
6950 ["minusb"] = 8863,
6951 ["boxtimes"] = 8864,
6952 ["timesb"] = 8864,
6953 ["dotssquare"] = 8865,
6954 ["sdotb"] = 8865,
6955 ["RightTee"] = 8866,
6956 ["vdash"] = 8866,

```



```

6957 ["LeftTee"] = 8867,
6958 ["dashv"] = 8867,
6959 ["DownTee"] = 8868,
6960 ["top"] = 8868,
6961 ["UpTee"] = 8869,
6962 ["bot"] = 8869,
6963 ["bottom"] = 8869,
6964 ["perp"] = 8869,
6965 ["models"] = 8871,
6966 ["DoubleRightTee"] = 8872,
6967 ["vDash"] = 8872,
6968 ["Vdash"] = 8873,
6969 ["Vvdash"] = 8874,
6970 ["VDash"] = 8875,
6971 ["nvdash"] = 8876,
6972 ["nvDash"] = 8877,
6973 ["nVdash"] = 8878,
6974 ["nVDash"] = 8879,
6975 ["prurel"] = 8880,
6976 ["LeftTriangle"] = 8882,
6977 ["vartriangleleft"] = 8882,
6978 ["vltri"] = 8882,
6979 ["RightTriangle"] = 8883,
6980 ["vartriangleright"] = 8883,
6981 ["vrtri"] = 8883,
6982 ["LeftTriangleEqual"] = 8884,
6983 ["ltrie"] = 8884,
6984 ["nvltrie"] = {8884, 8402},
6985 ["trianglelefteq"] = 8884,
6986 ["RightTriangleEqual"] = 8885,
6987 ["nvrtrie"] = {8885, 8402},
6988 ["rtrie"] = 8885,
6989 ["trianglerighteq"] = 8885,
6990 ["origof"] = 8886,
6991 ["imof"] = 8887,
6992 ["multimap"] = 8888,
6993 ["mumap"] = 8888,
6994 ["hercon"] = 8889,
6995 ["intcal"] = 8890,
6996 ["intercal"] = 8890,
6997 ["veebar"] = 8891,
6998 ["barvee"] = 8893,
6999 ["angrtvb"] = 8894,
7000 ["lrtri"] = 8895,
7001 ["Wedge"] = 8896,
7002 ["bigwedge"] = 8896,
7003 ["xwedge"] = 8896,

```

```

7004 ["Vee"] = 8897,
7005 ["bigvee"] = 8897,
7006 ["xvee"] = 8897,
7007 ["Intersection"] = 8898,
7008 ["bigcap"] = 8898,
7009 ["xcap"] = 8898,
7010 ["Union"] = 8899,
7011 ["bigcup"] = 8899,
7012 ["xcup"] = 8899,
7013 ["Diamond"] = 8900,
7014 ["diam"] = 8900,
7015 ["diamond"] = 8900,
7016 ["sdot"] = 8901,
7017 ["Star"] = 8902,
7018 ["sstarf"] = 8902,
7019 ["divideontimes"] = 8903,
7020 ["divonx"] = 8903,
7021 ["bowtie"] = 8904,
7022 ["ltimes"] = 8905,
7023 ["rtimes"] = 8906,
7024 ["leftthreetimes"] = 8907,
7025 ["lthree"] = 8907,
7026 ["rightthreetimes"] = 8908,
7027 ["rthree"] = 8908,
7028 ["backsimeq"] = 8909,
7029 ["bsime"] = 8909,
7030 ["curlyvee"] = 8910,
7031 ["cuvee"] = 8910,
7032 ["curlywedge"] = 8911,
7033 ["cuwed"] = 8911,
7034 ["Sub"] = 8912,
7035 ["Subset"] = 8912,
7036 ["Sup"] = 8913,
7037 ["Supset"] = 8913,
7038 ["Cap"] = 8914,
7039 ["Cup"] = 8915,
7040 ["fork"] = 8916,
7041 ["pitchfork"] = 8916,
7042 ["epar"] = 8917,
7043 ["lessdot"] = 8918,
7044 ["ltdot"] = 8918,
7045 ["gtdot"] = 8919,
7046 ["gtrdot"] = 8919,
7047 ["Ll"] = 8920,
7048 ["nLl"] = {8920, 824},
7049 ["Gg"] = 8921,
7050 ["ggg"] = 8921,

```

```

7051 ["nGg"] = {8921, 824},
7052 ["LessEqualGreater"] = 8922,
7053 ["leg"] = 8922,
7054 ["lesg"] = {8922, 65024},
7055 ["lesseqgtr"] = 8922,
7056 ["GreaterEqualLess"] = 8923,
7057 ["gel"] = 8923,
7058 ["gesl"] = {8923, 65024},
7059 ["gtreqless"] = 8923,
7060 ["cuepr"] = 8926,
7061 ["curlyeqprec"] = 8926,
7062 ["cuesc"] = 8927,
7063 ["curlyeqsucc"] = 8927,
7064 ["NotPrecedesSlantEqual"] = 8928,
7065 ["nprcue"] = 8928,
7066 ["NotSucceedsSlantEqual"] = 8929,
7067 ["nsccue"] = 8929,
7068 ["NotSquareSubsetEqual"] = 8930,
7069 ["nsqsube"] = 8930,
7070 ["NotSquareSupersetEqual"] = 8931,
7071 ["nsqsupe"] = 8931,
7072 ["lnsim"] = 8934,
7073 ["gnsim"] = 8935,
7074 ["precnsim"] = 8936,
7075 ["prnsim"] = 8936,
7076 ["scnsim"] = 8937,
7077 ["succnsim"] = 8937,
7078 ["NotLeftTriangle"] = 8938,
7079 ["nltri"] = 8938,
7080 ["ntriangleleft"] = 8938,
7081 ["NotRightTriangle"] = 8939,
7082 ["nrtri"] = 8939,
7083 ["ntriangleright"] = 8939,
7084 ["NotLeftTriangleEqual"] = 8940,
7085 ["nltrie"] = 8940,
7086 ["ntrianglelefteq"] = 8940,
7087 ["NotRightTriangleEqual"] = 8941,
7088 ["nrtrie"] = 8941,
7089 ["ntrianglerighteq"] = 8941,
7090 ["vellip"] = 8942,
7091 ["ctdot"] = 8943,
7092 ["utdot"] = 8944,
7093 ["dtdot"] = 8945,
7094 ["disin"] = 8946,
7095 ["isinsv"] = 8947,
7096 ["isins"] = 8948,
7097 ["isindot"] = 8949,

```

```

7098 ["notindot"] = {8949, 824},
7099 ["notinvc"] = 8950,
7100 ["notinvb"] = 8951,
7101 ["isinE"] = 8953,
7102 ["notinE"] = {8953, 824},
7103 ["nisd"] = 8954,
7104 ["xnis"] = 8955,
7105 ["nis"] = 8956,
7106 ["notnivc"] = 8957,
7107 ["notnivb"] = 8958,
7108 ["barwed"] = 8965,
7109 ["barwedge"] = 8965,
7110 ["Barwed"] = 8966,
7111 ["doublebarwedge"] = 8966,
7112 ["LeftCeiling"] = 8968,
7113 ["lceil"] = 8968,
7114 ["RightCeiling"] = 8969,
7115 ["rceil"] = 8969,
7116 ["LeftFloor"] = 8970,
7117 ["lfloor"] = 8970,
7118 ["RightFloor"] = 8971,
7119 ["rfloor"] = 8971,
7120 ["drcrop"] = 8972,
7121 ["dlcrop"] = 8973,
7122 ["urcrop"] = 8974,
7123 ["ulcrop"] = 8975,
7124 ["bnot"] = 8976,
7125 ["profline"] = 8978,
7126 ["profsurf"] = 8979,
7127 ["telrec"] = 8981,
7128 ["target"] = 8982,
7129 ["ulcorn"] = 8988,
7130 ["ulcorner"] = 8988,
7131 ["urcorn"] = 8989,
7132 ["urcorner"] = 8989,
7133 ["dlcorn"] = 8990,
7134 ["llcorner"] = 8990,
7135 ["drcorn"] = 8991,
7136 ["lrcorner"] = 8991,
7137 ["frown"] = 8994,
7138 ["sfrown"] = 8994,
7139 ["smile"] = 8995,
7140 ["ssmile"] = 8995,
7141 ["cylcty"] = 9005,
7142 ["profalar"] = 9006,
7143 ["topbot"] = 9014,
7144 ["ovbar"] = 9021,

```

```

7145 ["solbar"] = 9023,
7146 ["angzarr"] = 9084,
7147 ["lmoust"] = 9136,
7148 ["lmoustache"] = 9136,
7149 ["rmoust"] = 9137,
7150 ["rmoustache"] = 9137,
7151 ["OverBracket"] = 9140,
7152 ["tbrk"] = 9140,
7153 ["UnderBracket"] = 9141,
7154 ["bbrk"] = 9141,
7155 ["bbrktbrk"] = 9142,
7156 ["OverParenthesis"] = 9180,
7157 ["UnderParenthesis"] = 9181,
7158 ["OverBrace"] = 9182,
7159 ["UnderBrace"] = 9183,
7160 ["trpezium"] = 9186,
7161 ["elinters"] = 9191,
7162 ["blank"] = 9251,
7163 ["circledS"] = 9416,
7164 ["oS"] = 9416,
7165 ["HorizontalLine"] = 9472,
7166 ["boxh"] = 9472,
7167 ["boxv"] = 9474,
7168 ["boxdr"] = 9484,
7169 ["boxdl"] = 9488,
7170 ["boxur"] = 9492,
7171 ["boxul"] = 9496,
7172 ["boxvr"] = 9500,
7173 ["boxvl"] = 9508,
7174 ["boxhd"] = 9516,
7175 ["boxhu"] = 9524,
7176 ["boxvh"] = 9532,
7177 ["boxH"] = 9552,
7178 ["boxV"] = 9553,
7179 ["boxdR"] = 9554,
7180 ["boxDr"] = 9555,
7181 ["boxDR"] = 9556,
7182 ["boxdL"] = 9557,
7183 ["boxDL"] = 9558,
7184 ["boxDL"] = 9559,
7185 ["boxuR"] = 9560,
7186 ["boxUr"] = 9561,
7187 ["boxUR"] = 9562,
7188 ["boxuL"] = 9563,
7189 ["boxUL"] = 9564,
7190 ["boxUL"] = 9565,
7191 ["boxvR"] = 9566,

```

```

7192 ["boxVr"] = 9567,
7193 ["boxVR"] = 9568,
7194 ["boxvL"] = 9569,
7195 ["boxVl"] = 9570,
7196 ["boxVL"] = 9571,
7197 ["boxHd"] = 9572,
7198 ["boxhD"] = 9573,
7199 ["boxHD"] = 9574,
7200 ["boxHu"] = 9575,
7201 ["boxhU"] = 9576,
7202 ["boxHU"] = 9577,
7203 ["boxvH"] = 9578,
7204 ["boxVh"] = 9579,
7205 ["boxVH"] = 9580,
7206 ["uhblk"] = 9600,
7207 ["lhblk"] = 9604,
7208 ["block"] = 9608,
7209 ["blk14"] = 9617,
7210 ["blk12"] = 9618,
7211 ["blk34"] = 9619,
7212 ["Square"] = 9633,
7213 ["squ"] = 9633,
7214 ["square"] = 9633,
7215 ["FilledVerySmallSquare"] = 9642,
7216 ["blacksquare"] = 9642,
7217 ["squarf"] = 9642,
7218 ["squf"] = 9642,
7219 ["EmptyVerySmallSquare"] = 9643,
7220 ["rect"] = 9645,
7221 ["marker"] = 9646,
7222 ["fltns"] = 9649,
7223 ["bigtriangleup"] = 9651,
7224 ["xutri"] = 9651,
7225 ["blacktriangle"] = 9652,
7226 ["utrif"] = 9652,
7227 ["triangle"] = 9653,
7228 ["utri"] = 9653,
7229 ["blacktriangleright"] = 9656,
7230 ["rtrif"] = 9656,
7231 ["rtri"] = 9657,
7232 ["triangleright"] = 9657,
7233 ["bigtriangledown"] = 9661,
7234 ["xdtri"] = 9661,
7235 ["blacktriangledown"] = 9662,
7236 ["dtrif"] = 9662,
7237 ["dtri"] = 9663,
7238 ["triangledown"] = 9663,

```

7239 ["blacktriangleleft"] = 9666,  
 7240 ["ltrif"] = 9666,  
 7241 ["ltri"] = 9667,  
 7242 ["triangleleft"] = 9667,  
 7243 ["loz"] = 9674,  
 7244 ["lozenge"] = 9674,  
 7245 ["cir"] = 9675,  
 7246 ["tridot"] = 9708,  
 7247 ["bigcirc"] = 9711,  
 7248 ["xcirc"] = 9711,  
 7249 ["ultri"] = 9720,  
 7250 ["urtri"] = 9721,  
 7251 ["lltri"] = 9722,  
 7252 ["EmptySmallSquare"] = 9723,  
 7253 ["FilledSmallSquare"] = 9724,  
 7254 ["bigstar"] = 9733,  
 7255 ["starf"] = 9733,  
 7256 ["star"] = 9734,  
 7257 ["phone"] = 9742,  
 7258 ["female"] = 9792,  
 7259 ["male"] = 9794,  
 7260 ["spades"] = 9824,  
 7261 ["spadesuit"] = 9824,  
 7262 ["clubs"] = 9827,  
 7263 ["clubsuit"] = 9827,  
 7264 ["hearts"] = 9829,  
 7265 ["heartsuit"] = 9829,  
 7266 ["diamondsuit"] = 9830,  
 7267 ["diams"] = 9830,  
 7268 ["sung"] = 9834,  
 7269 ["flat"] = 9837,  
 7270 ["natur"] = 9838,  
 7271 ["natural"] = 9838,  
 7272 ["sharp"] = 9839,  
 7273 ["check"] = 10003,  
 7274 ["checkmark"] = 10003,  
 7275 ["cross"] = 10007,  
 7276 ["malt"] = 10016,  
 7277 ["maltese"] = 10016,  
 7278 ["sext"] = 10038,  
 7279 ["VerticalSeparator"] = 10072,  
 7280 ["lbbbrk"] = 10098,  
 7281 ["rbbbrk"] = 10099,  
 7282 ["bsolhsub"] = 10184,  
 7283 ["suphsol"] = 10185,  
 7284 ["LeftDoubleBracket"] = 10214,  
 7285 ["lobrk"] = 10214,

7286 ["RightDoubleBracket"] = 10215,  
 7287 ["robrk"] = 10215,  
 7288 ["LeftAngleBracket"] = 10216,  
 7289 ["lang"] = 10216,  
 7290 ["langle"] = 10216,  
 7291 ["RightAngleBracket"] = 10217,  
 7292 ["rang"] = 10217,  
 7293 ["rangle"] = 10217,  
 7294 ["Lang"] = 10218,  
 7295 ["Rang"] = 10219,  
 7296 ["loang"] = 10220,  
 7297 ["roang"] = 10221,  
 7298 ["LongLeftArrow"] = 10229,  
 7299 ["longleftarrow"] = 10229,  
 7300 ["xlarr"] = 10229,  
 7301 ["LongRightArrow"] = 10230,  
 7302 ["longrightarrow"] = 10230,  
 7303 ["xrarr"] = 10230,  
 7304 ["LongLeftRightArrow"] = 10231,  
 7305 ["longleftrightarrow"] = 10231,  
 7306 ["xharr"] = 10231,  
 7307 ["DoubleLongLeftArrow"] = 10232,  
 7308 ["Longleftarrow"] = 10232,  
 7309 ["xlArr"] = 10232,  
 7310 ["DoubleLongRightArrow"] = 10233,  
 7311 ["Longrightarrow"] = 10233,  
 7312 ["xrArr"] = 10233,  
 7313 ["DoubleLongLeftRightArrow"] = 10234,  
 7314 ["Longleftrightarrow"] = 10234,  
 7315 ["xhArr"] = 10234,  
 7316 ["longmapsto"] = 10236,  
 7317 ["xmap"] = 10236,  
 7318 ["dzigrarr"] = 10239,  
 7319 ["nvlArr"] = 10498,  
 7320 ["nvrArr"] = 10499,  
 7321 ["nvHarr"] = 10500,  
 7322 ["Map"] = 10501,  
 7323 ["lbarr"] = 10508,  
 7324 ["bkarow"] = 10509,  
 7325 ["rbarr"] = 10509,  
 7326 ["lBarr"] = 10510,  
 7327 ["dbkarow"] = 10511,  
 7328 ["rBarr"] = 10511,  
 7329 ["RBarr"] = 10512,  
 7330 ["drbkarow"] = 10512,  
 7331 ["DDottrahd"] = 10513,  
 7332 ["UpArrowBar"] = 10514,



```

7333 ["DownArrowBar"] = 10515,
7334 ["Rarrtl"] = 10518,
7335 ["latail"] = 10521,
7336 ["ratail"] = 10522,
7337 ["lAtail"] = 10523,
7338 ["rAtail"] = 10524,
7339 ["larrfs"] = 10525,
7340 ["rarrfs"] = 10526,
7341 ["larrbfs"] = 10527,
7342 ["rarrbfs"] = 10528,
7343 ["nwarhk"] = 10531,
7344 ["nearhk"] = 10532,
7345 ["hksearow"] = 10533,
7346 ["searhk"] = 10533,
7347 ["hkswarow"] = 10534,
7348 ["swarhk"] = 10534,
7349 ["nwnear"] = 10535,
7350 ["nesear"] = 10536,
7351 ["toea"] = 10536,
7352 ["seswar"] = 10537,
7353 ["tosa"] = 10537,
7354 ["swnwar"] = 10538,
7355 ["nrarrc"] = {10547, 824},
7356 ["rarrc"] = 10547,
7357 ["cudarrr"] = 10549,
7358 ["ldca"] = 10550,
7359 ["rdca"] = 10551,
7360 ["cudarrrl"] = 10552,
7361 ["larrpl"] = 10553,
7362 ["curarrm"] = 10556,
7363 ["cularrp"] = 10557,
7364 ["rarrpl"] = 10565,
7365 ["harrcir"] = 10568,
7366 ["Uarrocir"] = 10569,
7367 ["lurdshar"] = 10570,
7368 ["ldrushar"] = 10571,
7369 ["LeftRightVector"] = 10574,
7370 ["RightUpDownVector"] = 10575,
7371 ["DownLeftRightVector"] = 10576,
7372 ["LeftUpDownVector"] = 10577,
7373 ["LeftVectorBar"] = 10578,
7374 ["RightVectorBar"] = 10579,
7375 ["RightUpVectorBar"] = 10580,
7376 ["RightDownVectorBar"] = 10581,
7377 ["DownLeftVectorBar"] = 10582,
7378 ["DownRightVectorBar"] = 10583,
7379 ["LeftUpVectorBar"] = 10584,

```

```

7380 ["LeftDownVectorBar"] = 10585,
7381 ["LeftTeeVector"] = 10586,
7382 ["RightTeeVector"] = 10587,
7383 ["RightUpTeeVector"] = 10588,
7384 ["RightDownTeeVector"] = 10589,
7385 ["DownLeftTeeVector"] = 10590,
7386 ["DownRightTeeVector"] = 10591,
7387 ["LeftUpTeeVector"] = 10592,
7388 ["LeftDownTeeVector"] = 10593,
7389 ["lHar"] = 10594,
7390 ["uHar"] = 10595,
7391 ["rHar"] = 10596,
7392 ["dHar"] = 10597,
7393 ["luruhar"] = 10598,
7394 ["ldrdhar"] = 10599,
7395 ["ruluhar"] = 10600,
7396 ["rdldhar"] = 10601,
7397 ["lharul"] = 10602,
7398 ["llhard"] = 10603,
7399 ["rharul"] = 10604,
7400 ["lrhard"] = 10605,
7401 ["UpEquilibrium"] = 10606,
7402 ["udhar"] = 10606,
7403 ["ReverseUpEquilibrium"] = 10607,
7404 ["duhar"] = 10607,
7405 ["RoundImplies"] = 10608,
7406 ["erarr"] = 10609,
7407 ["simrarr"] = 10610,
7408 ["larrsim"] = 10611,
7409 ["rarrsim"] = 10612,
7410 ["rarrap"] = 10613,
7411 ["ltlarr"] = 10614,
7412 ["gtrarr"] = 10616,
7413 ["subrarr"] = 10617,
7414 ["suplarr"] = 10619,
7415 ["lfisht"] = 10620,
7416 ["rfisht"] = 10621,
7417 ["ufisht"] = 10622,
7418 ["dfisht"] = 10623,
7419 ["lopar"] = 10629,
7420 ["ropar"] = 10630,
7421 ["lbrke"] = 10635,
7422 ["rbrke"] = 10636,
7423 ["lbrkslu"] = 10637,
7424 ["rbrksld"] = 10638,
7425 ["lbrksld"] = 10639,
7426 ["rbrkslu"] = 10640,

```

```

7427 ["langd"] = 10641,
7428 ["rangd"] = 10642,
7429 ["lparlt"] = 10643,
7430 ["rpargt"] = 10644,
7431 ["gtlPar"] = 10645,
7432 ["ltrPar"] = 10646,
7433 ["vzigzag"] = 10650,
7434 ["vangrt"] = 10652,
7435 ["angrtvbd"] = 10653,
7436 ["ange"] = 10660,
7437 ["range"] = 10661,
7438 ["dwangle"] = 10662,
7439 ["uwangle"] = 10663,
7440 ["angmsdaa"] = 10664,
7441 ["angmsdab"] = 10665,
7442 ["angmsdac"] = 10666,
7443 ["angmsdad"] = 10667,
7444 ["angmsdae"] = 10668,
7445 ["angmsdaf"] = 10669,
7446 ["angmsdag"] = 10670,
7447 ["angmsdah"] = 10671,
7448 ["bemptyv"] = 10672,
7449 ["demptyv"] = 10673,
7450 ["cemptyv"] = 10674,
7451 ["raemptyv"] = 10675,
7452 ["laemptyv"] = 10676,
7453 ["ohbar"] = 10677,
7454 ["omid"] = 10678,
7455 ["opar"] = 10679,
7456 ["operp"] = 10681,
7457 ["olcross"] = 10683,
7458 ["odsold"] = 10684,
7459 ["olcir"] = 10686,
7460 ["ofcir"] = 10687,
7461 ["olt"] = 10688,
7462 ["ogt"] = 10689,
7463 ["cirscir"] = 10690,
7464 ["cirE"] = 10691,
7465 ["solb"] = 10692,
7466 ["bsolb"] = 10693,
7467 ["boxbox"] = 10697,
7468 ["trish"] = 10701,
7469 ["rtriltri"] = 10702,
7470 ["LeftTriangleBar"] = 10703,
7471 ["NotLeftTriangleBar"] = {10703, 824},
7472 ["NotRightTriangleBar"] = {10704, 824},
7473 ["RightTriangleBar"] = 10704,

```

```

7474 ["i infin"] = 10716,
7475 ["infintie"] = 10717,
7476 ["nvinfin"] = 10718,
7477 ["eparsl"] = 10723,
7478 ["smeparsl"] = 10724,
7479 ["eqvparsl"] = 10725,
7480 ["blacklozenge"] = 10731,
7481 ["lozf"] = 10731,
7482 ["RuleDelayed"] = 10740,
7483 ["dsol"] = 10742,
7484 ["bigodot"] = 10752,
7485 ["xodot"] = 10752,
7486 ["bigoplus"] = 10753,
7487 ["xoplus"] = 10753,
7488 ["bigotimes"] = 10754,
7489 ["xotime"] = 10754,
7490 ["biguplus"] = 10756,
7491 ["xuplus"] = 10756,
7492 ["bigsqcup"] = 10758,
7493 ["xsqcup"] = 10758,
7494 ["iiiint"] = 10764,
7495 ["qint"] = 10764,
7496 ["fpartint"] = 10765,
7497 ["cirfnint"] = 10768,
7498 ["awint"] = 10769,
7499 ["rppolint"] = 10770,
7500 ["scpolint"] = 10771,
7501 ["npolint"] = 10772,
7502 ["pointint"] = 10773,
7503 ["quatint"] = 10774,
7504 ["intlarhk"] = 10775,
7505 ["pluscir"] = 10786,
7506 ["plusacir"] = 10787,
7507 ["simplus"] = 10788,
7508 ["plusdu"] = 10789,
7509 ["plussim"] = 10790,
7510 ["plustwo"] = 10791,
7511 ["mcomma"] = 10793,
7512 ["minusdu"] = 10794,
7513 ["loplus"] = 10797,
7514 ["roplus"] = 10798,
7515 ["Cross"] = 10799,
7516 ["timesd"] = 10800,
7517 ["timesbar"] = 10801,
7518 ["smashp"] = 10803,
7519 ["lotimes"] = 10804,
7520 ["rotimes"] = 10805,

```

7521 ["otimesas"] = 10806,  
 7522 ["Otimes"] = 10807,  
 7523 ["odiv"] = 10808,  
 7524 ["tripplus"] = 10809,  
 7525 ["triminus"] = 10810,  
 7526 ["tritime"] = 10811,  
 7527 ["intprod"] = 10812,  
 7528 ["iprod"] = 10812,  
 7529 ["amalg"] = 10815,  
 7530 ["capdot"] = 10816,  
 7531 ["ncup"] = 10818,  
 7532 ["ncap"] = 10819,  
 7533 ["capand"] = 10820,  
 7534 ["cupor"] = 10821,  
 7535 ["cupcap"] = 10822,  
 7536 ["capcup"] = 10823,  
 7537 ["cupbrcap"] = 10824,  
 7538 ["capbrcup"] = 10825,  
 7539 ["cupcup"] = 10826,  
 7540 ["capcap"] = 10827,  
 7541 ["ccups"] = 10828,  
 7542 ["ccaps"] = 10829,  
 7543 ["ccupssm"] = 10832,  
 7544 ["And"] = 10835,  
 7545 ["Or"] = 10836,  
 7546 ["andand"] = 10837,  
 7547 ["oror"] = 10838,  
 7548 ["orslope"] = 10839,  
 7549 ["andslope"] = 10840,  
 7550 ["andv"] = 10842,  
 7551 ["orv"] = 10843,  
 7552 ["andd"] = 10844,  
 7553 ["ord"] = 10845,  
 7554 ["wedbar"] = 10847,  
 7555 ["sdote"] = 10854,  
 7556 ["simdot"] = 10858,  
 7557 ["congdots"] = 10861,  
 7558 ["ncongdots"] = {10861, 824},  
 7559 ["easter"] = 10862,  
 7560 ["apacir"] = 10863,  
 7561 ["apE"] = 10864,  
 7562 ["napE"] = {10864, 824},  
 7563 ["eplus"] = 10865,  
 7564 ["pluse"] = 10866,  
 7565 ["Esim"] = 10867,  
 7566 ["Colone"] = 10868,  
 7567 ["Equal"] = 10869,

```

7568 ["ddotseq"] = 10871,
7569 ["eDDot"] = 10871,
7570 ["equivDD"] = 10872,
7571 ["ltcir"] = 10873,
7572 ["gtcir"] = 10874,
7573 ["ltquest"] = 10875,
7574 ["gtquest"] = 10876,
7575 ["LessSlantEqual"] = 10877,
7576 ["NotLessSlantEqual"] = {10877, 824},
7577 ["leqslant"] = 10877,
7578 ["les"] = 10877,
7579 ["nleqslant"] = {10877, 824},
7580 ["nles"] = {10877, 824},
7581 ["GreaterSlantEqual"] = 10878,
7582 ["NotGreaterSlantEqual"] = {10878, 824},
7583 ["geqslant"] = 10878,
7584 ["ges"] = 10878,
7585 ["ngeqslant"] = {10878, 824},
7586 ["nges"] = {10878, 824},
7587 ["lesdot"] = 10879,
7588 ["gesdot"] = 10880,
7589 ["lesdoto"] = 10881,
7590 ["gesdoto"] = 10882,
7591 ["lesdotor"] = 10883,
7592 ["gesdoto1"] = 10884,
7593 ["lap"] = 10885,
7594 ["lessapprox"] = 10885,
7595 ["gap"] = 10886,
7596 ["gtrapprox"] = 10886,
7597 ["lne"] = 10887,
7598 ["lneq"] = 10887,
7599 ["gne"] = 10888,
7600 ["gneq"] = 10888,
7601 ["lnap"] = 10889,
7602 ["lnapprox"] = 10889,
7603 ["gnap"] = 10890,
7604 ["gnapprox"] = 10890,
7605 ["lEg"] = 10891,
7606 ["lesseqqgtr"] = 10891,
7607 ["gEl"] = 10892,
7608 ["gtreqqless"] = 10892,
7609 ["lsime"] = 10893,
7610 ["gsime"] = 10894,
7611 ["lsimg"] = 10895,
7612 ["gsiml"] = 10896,
7613 ["lgE"] = 10897,
7614 ["glE"] = 10898,

```

```

7615 ["lesges"] = 10899,
7616 ["gesles"] = 10900,
7617 ["els"] = 10901,
7618 ["eqslantless"] = 10901,
7619 ["egs"] = 10902,
7620 ["eqslantgtr"] = 10902,
7621 ["elsdot"] = 10903,
7622 ["egsdot"] = 10904,
7623 ["el"] = 10905,
7624 ["eg"] = 10906,
7625 ["siml"] = 10909,
7626 ["simg"] = 10910,
7627 ["simlE"] = 10911,
7628 ["simgE"] = 10912,
7629 ["LessLess"] = 10913,
7630 ["NotNestedLessLess"] = {10913, 824},
7631 ["GreaterGreater"] = 10914,
7632 ["NotNestedGreaterGreater"] = {10914, 824},
7633 ["glj"] = 10916,
7634 ["gla"] = 10917,
7635 ["ltcc"] = 10918,
7636 ["gtcc"] = 10919,
7637 ["lescc"] = 10920,
7638 ["gescc"] = 10921,
7639 ["smt"] = 10922,
7640 ["lat"] = 10923,
7641 ["smte"] = 10924,
7642 ["smtes"] = {10924, 65024},
7643 ["late"] = 10925,
7644 ["lates"] = {10925, 65024},
7645 ["bumpE"] = 10926,
7646 ["NotPrecedesEqual"] = {10927, 824},
7647 ["PrecedesEqual"] = 10927,
7648 ["npre"] = {10927, 824},
7649 ["npreceq"] = {10927, 824},
7650 ["pre"] = 10927,
7651 ["preceq"] = 10927,
7652 ["NotSucceedsEqual"] = {10928, 824},
7653 ["SucceedsEqual"] = 10928,
7654 ["nsce"] = {10928, 824},
7655 ["nsucceq"] = {10928, 824},
7656 ["sce"] = 10928,
7657 ["succeq"] = 10928,
7658 ["prE"] = 10931,
7659 ["scE"] = 10932,
7660 ["precneqq"] = 10933,
7661 ["prnE"] = 10933,

```

```

7662 ["scnE"] = 10934,
7663 ["succneqq"] = 10934,
7664 ["prap"] = 10935,
7665 ["precapprox"] = 10935,
7666 ["scap"] = 10936,
7667 ["succapprox"] = 10936,
7668 ["precnapprox"] = 10937,
7669 ["prnap"] = 10937,
7670 ["scnap"] = 10938,
7671 ["succnapprox"] = 10938,
7672 ["Pr"] = 10939,
7673 ["Sc"] = 10940,
7674 ["subdot"] = 10941,
7675 ["supdot"] = 10942,
7676 ["subplus"] = 10943,
7677 ["supplus"] = 10944,
7678 ["submult"] = 10945,
7679 ["supmult"] = 10946,
7680 ["subedot"] = 10947,
7681 ["supedot"] = 10948,
7682 ["nsubE"] = {10949, 824},
7683 ["nsubseteqq"] = {10949, 824},
7684 ["subE"] = 10949,
7685 ["subseteqq"] = 10949,
7686 ["nsupE"] = {10950, 824},
7687 ["nsupseteqq"] = {10950, 824},
7688 ["supE"] = 10950,
7689 ["supseteqq"] = 10950,
7690 ["subsim"] = 10951,
7691 ["supsim"] = 10952,
7692 ["subnE"] = 10955,
7693 ["subsetneqq"] = 10955,
7694 ["varsubsetneqq"] = {10955, 65024},
7695 ["vsubnE"] = {10955, 65024},
7696 ["supnE"] = 10956,
7697 ["supsetneqq"] = 10956,
7698 ["varsupsetneqq"] = {10956, 65024},
7699 ["vsupnE"] = {10956, 65024},
7700 ["csub"] = 10959,
7701 ["csup"] = 10960,
7702 ["csube"] = 10961,
7703 ["csupe"] = 10962,
7704 ["subsup"] = 10963,
7705 ["supsub"] = 10964,
7706 ["subsub"] = 10965,
7707 ["supsup"] = 10966,
7708 ["suphsub"] = 10967,

```



```

7709 ["supdsub"] = 10968,
7710 ["forkv"] = 10969,
7711 ["topfork"] = 10970,
7712 ["mlcp"] = 10971,
7713 ["Dashv"] = 10980,
7714 ["DoubleLeftTee"] = 10980,
7715 ["Vdashl"] = 10982,
7716 ["Barv"] = 10983,
7717 ["vBar"] = 10984,
7718 ["vBarv"] = 10985,
7719 ["Vbar"] = 10987,
7720 ["Not"] = 10988,
7721 ["bNot"] = 10989,
7722 ["rnmid"] = 10990,
7723 ["cirmid"] = 10991,
7724 ["midcir"] = 10992,
7725 ["topcir"] = 10993,
7726 ["nhpar"] = 10994,
7727 ["parsim"] = 10995,
7728 ["nparsl"] = {11005, 8421},
7729 ["parsl"] = 11005,
7730 ["fflig"] = 64256,
7731 ["filig"] = 64257,
7732 ["fllig"] = 64258,
7733 ["ffilig"] = 64259,
7734 ["ffllig"] = 64260,
7735 ["Ascr"] = 119964,
7736 ["Cscr"] = 119966,
7737 ["Dscr"] = 119967,
7738 ["Gscr"] = 119970,
7739 ["Jscr"] = 119973,
7740 ["Kscr"] = 119974,
7741 ["Nscr"] = 119977,
7742 ["Oscr"] = 119978,
7743 ["Pscr"] = 119979,
7744 ["Qscr"] = 119980,
7745 ["Sscr"] = 119982,
7746 ["Tscr"] = 119983,
7747 ["Uscr"] = 119984,
7748 ["Vscr"] = 119985,
7749 ["Wscr"] = 119986,
7750 ["Xscr"] = 119987,
7751 ["Yscr"] = 119988,
7752 ["Zscr"] = 119989,
7753 ["ascr"] = 119990,
7754 ["bscr"] = 119991,
7755 ["cscr"] = 119992,

```

```

7756 ["dscr"] = 119993,
7757 ["fscr"] = 119995,
7758 ["hscr"] = 119997,
7759 ["iscr"] = 119998,
7760 ["jscr"] = 119999,
7761 ["kscr"] = 120000,
7762 ["lscr"] = 120001,
7763 ["mscr"] = 120002,
7764 ["nscr"] = 120003,
7765 ["pscr"] = 120005,
7766 ["qscr"] = 120006,
7767 ["rscr"] = 120007,
7768 ["sscr"] = 120008,
7769 ["tscr"] = 120009,
7770 ["uscr"] = 120010,
7771 ["vscr"] = 120011,
7772 ["wscr"] = 120012,
7773 ["xscr"] = 120013,
7774 ["yscr"] = 120014,
7775 ["zscr"] = 120015,
7776 ["Afr"] = 120068,
7777 ["Bfr"] = 120069,
7778 ["Dfr"] = 120071,
7779 ["Efr"] = 120072,
7780 ["Ffr"] = 120073,
7781 ["Gfr"] = 120074,
7782 ["Jfr"] = 120077,
7783 ["Kfr"] = 120078,
7784 ["Lfr"] = 120079,
7785 ["Mfr"] = 120080,
7786 ["Nfr"] = 120081,
7787 ["Ofr"] = 120082,
7788 ["Pfr"] = 120083,
7789 ["Qfr"] = 120084,
7790 ["Sfr"] = 120086,
7791 ["Tfr"] = 120087,
7792 ["Ufr"] = 120088,
7793 ["Vfr"] = 120089,
7794 ["Wfr"] = 120090,
7795 ["Xfr"] = 120091,
7796 ["Yfr"] = 120092,
7797 ["afr"] = 120094,
7798 ["bfr"] = 120095,
7799 ["cfr"] = 120096,
7800 ["dfr"] = 120097,
7801 ["efr"] = 120098,
7802 ["ffr"] = 120099,

```

```

7803 ["gfr"] = 120100,
7804 ["hfr"] = 120101,
7805 ["ifr"] = 120102,
7806 ["jfr"] = 120103,
7807 ["kfr"] = 120104,
7808 ["lfr"] = 120105,
7809 ["mfr"] = 120106,
7810 ["nfr"] = 120107,
7811 ["ofr"] = 120108,
7812 ["pfr"] = 120109,
7813 ["qfr"] = 120110,
7814 ["rfr"] = 120111,
7815 ["sfr"] = 120112,
7816 ["tfr"] = 120113,
7817 ["ufr"] = 120114,
7818 ["vfr"] = 120115,
7819 ["wfr"] = 120116,
7820 ["xfr"] = 120117,
7821 ["yfr"] = 120118,
7822 ["zfr"] = 120119,
7823 ["Aopf"] = 120120,
7824 ["Bopf"] = 120121,
7825 ["Dopf"] = 120123,
7826 ["Eopf"] = 120124,
7827 ["Fopf"] = 120125,
7828 ["Gopf"] = 120126,
7829 ["Iopf"] = 120128,
7830 ["Jopf"] = 120129,
7831 ["Kopf"] = 120130,
7832 ["Lopf"] = 120131,
7833 ["Mopf"] = 120132,
7834 ["Oopf"] = 120134,
7835 ["Sopf"] = 120138,
7836 ["Topf"] = 120139,
7837 ["Uopf"] = 120140,
7838 ["Vopf"] = 120141,
7839 ["Wopf"] = 120142,
7840 ["Xopf"] = 120143,
7841 ["Yopf"] = 120144,
7842 ["aopf"] = 120146,
7843 ["bopf"] = 120147,
7844 ["copf"] = 120148,
7845 ["dopf"] = 120149,
7846 ["eopf"] = 120150,
7847 ["fopf"] = 120151,
7848 ["gopf"] = 120152,
7849 ["hopf"] = 120153,

```

```

7850 ["iopf"] = 120154,
7851 ["jopf"] = 120155,
7852 ["kopf"] = 120156,
7853 ["lopf"] = 120157,
7854 ["mopf"] = 120158,
7855 ["nopf"] = 120159,
7856 ["oopf"] = 120160,
7857 ["popf"] = 120161,
7858 ["qopf"] = 120162,
7859 ["ropf"] = 120163,
7860 ["sopf"] = 120164,
7861 ["topf"] = 120165,
7862 ["uopf"] = 120166,
7863 ["vopf"] = 120167,
7864 ["wopf"] = 120168,
7865 ["xopf"] = 120169,
7866 ["yopf"] = 120170,
7867 ["zopf"] = 120171,
7868 }

```

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

```

7869 function entities.dec_entity(s)
7870 local n = tonumber(s)
7871 if n == nil then
7872 return "&#" .. s .. ";" -- fallback for unknown entities
7873 end
7874 return utf8.char(n)
7875 end

```

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

```

7876 function entities.hex_entity(s)
7877 local n = tonumber("0x"..s)
7878 if n == nil then
7879 return "&#x" .. s .. ";" -- fallback for unknown entities
7880 end
7881 return utf8.char(n)
7882 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.

```

7883 function entities.hex_entity_with_x_char(x, s)
7884 local n = tonumber("0x"..s)
7885 if n == nil then
7886 return "&#" .. x .. s .. ";" -- fallback for unknown entities
7887 end

```

```

7888 return utf8.char(n)
7889 end

```

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

```

7890 function entities.char_entity(s)
7891 local code_points = character_entities[s]
7892 if code_points == nil then
7893 return "&" .. s .. ";"
7894 end
7895 if type(code_points) ~= 'table' then
7896 code_points = {code_points}
7897 end
7898 local char_table = {}
7899 for _, code_point in ipairs(code_points) do
7900 table.insert(char_table, utf8.char(code_point))
7901 end
7902 return table.concat(char_table)
7903 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.

```

7904 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.

```

7905 local parsers
7906 function M.writer.new(options)
7907 local self = {}

```

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

```

7908 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.

```
7909 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.

```
7910 local slice_specifiers = {}
7911 for specifier in options.slice:gmatch("[~%s]+") do
7912 table.insert(slice_specifiers, specifier)
7913 end
7914
7915 if #slice_specifiers == 2 then
7916 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
7917 local slice_begin_type = self.slice_begin:sub(1, 1)
7918 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
7919 self.slice_begin = "^" .. self.slice_begin
7920 end
7921 local slice_end_type = self.slice_end:sub(1, 1)
7922 if slice_end_type ~= "^" and slice_end_type ~= "$" then
7923 self.slice_end = "$" .. self.slice_end
7924 end
7925 elseif #slice_specifiers == 1 then
7926 self.slice_begin = "^" .. slice_specifiers[1]
7927 self.slice_end = "$" .. slice_specifiers[1]
7928 end
7929
7930 self.slice_begin_type = self.slice_begin:sub(1, 1)
7931 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
7932 self.slice_end_type = self.slice_end:sub(1, 1)
7933 self.slice_end_identifier = self.slice_end:sub(2) or ""
7934
7935 if self.slice_begin == "^" and self.slice_end ~= "^" then
7936 self.is_writing = true
7937 else
7938 self.is_writing = false
7939 end
```

#### 3.1.4.2 Basic Formatter Variables and Functions

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

```
7940 self.space = " "
```

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

```
7941 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
7942 function self.plain(s)
7943 return s
7944 end
```

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

```
7945 function self.paragraph(s)
7946 if not self.is_writing then return "" end
7947 return s
7948 end
```

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

```
7949 self.interblocksep_text = "\\markdownRendererInterblockSeparator\\n{"
7950 function self.interblocksep()
7951 if not self.is_writing then return "" end
7952 return self.interblocksep_text
7953 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.

```
7954 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\\n{"
7955 function self.paragraphsep()
7956 if not self.is_writing then return "" end
7957 return self.paragraphsep_text
7958 end
```

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

```
7959 self.undosep_text = "\\markdownRendererUndoSeparator\\n{"
7960 function self.undosep()
7961 if not self.is_writing then return "" end
7962 return self.undosep_text
7963 end
```

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

```
7964 self.soft_line_break = function()
7965 if self.flatten_inlines then return "\\n" end
7966 return "\\markdownRendererSoftLineBreak\\n{"
7967 end
```

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

```
7968 self.hard_line_break = function()
7969 if self.flatten_inlines then return "\\n" end
7970 return "\\markdownRendererHardLineBreak\\n{"
7971 end
```

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

```
7972 self.ellipsis = "\\markdownRendererEllipsis{}
```

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

```
7973 function self.thematic_break()
7974 if not self.is_writing then return "" end
7975 return "\\markdownRendererThematicBreak{"
7976 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.

```
7977 self.escaped_uri_chars = {
7978 ["{"] = "\\markdownRendererLeftBrace{",
7979 ["}"] = "\\markdownRendererRightBrace{",
7980 ["\\"] = "\\markdownRendererBackslash{",
7981 ["\r"] = " ",
7982 ["\n"] = " ",
7983 }
7984 self.escaped_minimal_strings = {
7985 ["^"] = "\\markdownRendererCircumflex",
7986 .. "\\markdownRendererCircumflex ",
7987 ["☑"] = "\\markdownRendererTickedBox{",
7988 ["◻"] = "\\markdownRendererHalfTickedBox{",
7989 ["□"] = "\\markdownRendererUntickedBox{",
7990 [entities.hex_entity('FFFD')]
7991 = "\\markdownRendererReplacementCharacter{",
7992 }
```

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

```
7993 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
7994 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

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

```
7995 self.escaped_chars = {
7996 ["{"] = "\\markdownRendererLeftBrace{",
7997 ["}"] = "\\markdownRendererRightBrace{",
7998 ["%"] = "\\markdownRendererPercentSign{",
7999 ["\\"] = "\\markdownRendererBackslash{",
8000 ["#"] = "\\markdownRendererHash{",
8001 ["$"] = "\\markdownRendererDollarSign{",
8002 ["&"] = "\\markdownRendererAmpersand{",
8003 ["_"] = "\\markdownRendererUnderscore{",
```



```

8004 ["^"] = "\\markdownRendererCircumflex{}",
8005 ["~"] = "\\markdownRendererTilde{}",
8006 ["|"] = "\\markdownRendererPipe{}",
8007 [entities.hex_entity('0000')]
8008 = "\\markdownRendererReplacementCharacter{}",
8009 }

```

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.

```

8010 local function create_escaper(char_escapes, string_escapes)
8011 local escape = util.escaper(char_escapes, string_escapes)
8012 return function(s)
8013 if self.flatten_inlines then return s end
8014 return escape(s)
8015 end
8016 end
8017 local escape_typographic_text = create_escaper(
8018 self.escaped_chars, self.escaped_strings)
8019 local escape_programmatic_text = create_escaper(
8020 self.escaped_uri_chars, self.escaped_minimal_strings)
8021 local escape_minimal = create_escaper(
8022 {}, 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.

```

8023 self.escape = escape_typographic_text
8024 self.math = escape_minimal
8025 if options.hybrid then
8026 self.identifier = escape_minimal
8027 self.string = escape_minimal
8028 self.uri = escape_minimal
8029 self.infostring = escape_minimal
8030 else
8031 self.identifier = escape_programmatic_text
8032 self.string = escape_typographic_text
8033 self.uri = escape_programmatic_text
8034 self.infostring = escape_programmatic_text
8035 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.

```
8036 function self.warning(t, m)
8037 return {"\\markdownRendererWarning{" , self.escape(t), "}{" ,
8038 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
8039 escape_minimal(m or ""), "}"}
8040 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.

```
8041 function self.error(t, m)
8042 return {"\\markdownRendererError{" , self.escape(t), "}{" ,
8043 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
8044 escape_minimal(m or ""), "}"}
8045 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.

```
8046 function self.code(s, attributes)
8047 if self.flatten_inlines then return s end
8048 local buf = {}
8049 if attributes ~= nil then
8050 table.insert(buf,
8051 "\\markdownRendererCodeSpanAttributeContextBegin\\n")
8052 table.insert(buf, self.attributes(attributes))
8053 end
8054 table.insert(buf,
8055 {"\\markdownRendererCodeSpan{" , self.escape(s), "}"}
8056 if attributes ~= nil then
8057 table.insert(buf,
8058 "\\markdownRendererCodeSpanAttributeContextEnd{")
8059 end
8060 return buf
8061 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.

```
8062 function self.link(lab, src, tit, attributes)
8063 if self.flatten_inlines then return lab end
8064 local buf = {}
8065 if attributes ~= nil then
```

```

8066 table.insert(buf,
8067 "\\markdownRendererLinkAttributeContextBegin\n")
8068 table.insert(buf, self.attributes(attributes))
8069 end
8070 table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ",
8071 "{" ,self.escape(src),"} ",
8072 "{" ,self.uri(src),"} ",
8073 "{" ,self.string(tit or ""),"} "})
8074 if attributes ~= nil then
8075 table.insert(buf,
8076 "\\markdownRendererLinkAttributeContextEnd{ }")
8077 end
8078 return buf
8079 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.

```

8080 function self.image(lab, src, tit, attributes)
8081 if self.flatten_inlines then return lab end
8082 local buf = {}
8083 if attributes ~= nil then
8084 table.insert(buf,
8085 "\\markdownRendererImageAttributeContextBegin\n")
8086 table.insert(buf, self.attributes(attributes))
8087 end
8088 table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
8089 "{" ,self.string(src),"} ",
8090 "{" ,self.uri(src),"} ",
8091 "{" ,tit,"} "})
8092 if attributes ~= nil then
8093 table.insert(buf,
8094 "\\markdownRendererImageAttributeContextEnd{ }")
8095 end
8096 return buf
8097 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.

```

8098 function self.bulletlist(items,tight)
8099 if not self.is_writing then return "" end
8100 local buffer = {}

```

```

8101 for _,item in ipairs(items) do
8102 if item ~= "" then
8103 buffer[#buffer + 1] = self.bulletitem(item)
8104 end
8105 end
8106 local contents = util.intersperse(buffer,"\n")
8107 if tight and options.tightLists then
8108 return {"\\markdownRendererUlBeginTight\n",contents,
8109 "\n\\markdownRendererUlEndTight "}
8110 else
8111 return {"\\markdownRendererUlBegin\n",contents,
8112 "\n\\markdownRendererUlEnd "}
8113 end
8114 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.

```

8115 function self.bulletitem(s)
8116 return {"\\markdownRendererUlItem ",s,
8117 "\\markdownRendererUlItemEnd "}
8118 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.

```

8119 function self.orderedlist(items,tight,startnum)
8120 if not self.is_writing then return "" end
8121 local buffer = {}
8122 local num = startnum
8123 for _,item in ipairs(items) do
8124 if item ~= "" then
8125 buffer[#buffer + 1] = self.ordereditem(item,num)
8126 end
8127 if num ~= nil and item ~= "" then
8128 num = num + 1
8129 end
8130 end
8131 local contents = util.intersperse(buffer,"\n")
8132 if tight and options.tightLists then
8133 return {"\\markdownRendererOlBeginTight\n",contents,
8134 "\n\\markdownRendererOlEndTight "}
8135 else
8136 return {"\\markdownRendererOlBegin\n",contents,
8137 "\n\\markdownRendererOlEnd "}
8138 end
8139 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.

```

8140 function self.ordereditem(s,num)
8141 if num ~= nil then
8142 return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
8143 "\\markdownRendererOlItemEnd "}
8144 else
8145 return {"\\markdownRendererOlItem ",s,
8146 "\\markdownRendererOlItemEnd "}
8147 end
8148 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.

```

8149 function self.inline_html_comment(contents)
8150 if self.flatten_inlines then return contents end
8151 return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
8152 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.

```

8153 function self.inline_html_tag(contents)
8154 if self.flatten_inlines then return contents end
8155 return {"\\markdownRendererInlineHtmlTag{" ,
8156 self.string(contents),"}"}
8157 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.

```

8158 function self.block_html_element(s)
8159 if not self.is_writing then return "" end
8160 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8161 return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
8162 end

```

Define `writer->block_html_cdata_element` as a function that transforms the contents of a block HTML CDATA element such as `<script>`, `<pre>`, `<style>`, or `<textarea>` (CommonMark HTML block type 1) to the output format.

```

8163 function self.block_html_cdata_element(s)
8164 if not self.is_writing then return "" end
8165 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")

```

```

8166 return {"\\markdownRendererInputBlockHtmlCdataElement{" ,name,""}
8167 end

```

Define `writer->block_html_comment` as a function that transforms the contents of a block HTML comment (CommonMark HTML block type 2) to the output format, where `s` is the full comment text.

```

8168 function self.block_html_comment(contents)
8169 if self.flatten_inlines then return contents end
8170 return {"\\markdownRendererBlockHtmlComment{" ,contents,""}
8171 end

```

Define `writer->block_html_processing_instruction` as a function that transforms the contents of a block HTML processing instruction (CommonMark HTML block type 3) to the output format.

```

8172 function self.block_html_processing_instruction(s)
8173 if not self.is_writing then return "" end
8174 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8175 return {
8176 "\\markdownRendererInputBlockHtmlProcessingInstruction{" ,name,""}
8177 end

```

Define `writer->block_html_declaration` as a function that transforms the contents of a block HTML declaration (CommonMark HTML block type 4) to the output format.

```

8178 function self.block_html_declaration(s)
8179 if not self.is_writing then return "" end
8180 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8181 return {
8182 "\\markdownRendererInputBlockHtmlDeclaration{" ,name,""}
8183 end

```

Define `writer->block_html_cdata_section` as a function that transforms the contents of a block HTML CDATA section (CommonMark HTML block type 5) to the output format.

```

8184 function self.block_html_cdata_section(s)
8185 if not self.is_writing then return "" end
8186 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8187 return {"\\markdownRendererInputBlockHtmlCdataSection{" ,name,""}
8188 end

```

Define `writer->block_html_pCDATA_element` as a function that transforms the contents of a block HTML PCDATA element such as `<div>` or `<table>` (CommonMark HTML block type 6) to the output format.

```

8189 function self.block_html_pCDATA_element(s)
8190 if not self.is_writing then return "" end
8191 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
8192 return {"\\markdownRendererInputBlockHtmlPcdataElement{" ,name,""}
8193 end

```

Define `writer->block_html_standalone_tag` as a function that transforms the contents of a block HTML element that does not fall into types 1–6 (CommonMark HTML block type 7) to the output format.

```
8194 function self.block_html_standalone_tag(contents)
8195 if self.flatten_inlines then return contents end
8196 return {"\\markdownRendererBlockHtmlStandaloneTag{",
8197 self.string(contents),"}"}
8198 end
```

Define `writer->inline_html_processing_instruction` as a function that transforms the contents of an inline HTML processing instruction to the output format.

```
8199 function self.inline_html_processing_instruction(contents)
8200 if self.flatten_inlines then return contents end
8201 return {"\\markdownRendererInlineHtmlProcessingInstruction{",
8202 self.string(contents),"}"}
8203 end
```

Define `writer->inline_html_declaration` as a function that transforms the contents of an inline HTML declaration to the output format.

```
8204 function self.inline_html_declaration(contents)
8205 if self.flatten_inlines then return contents end
8206 return {"\\markdownRendererInlineHtmlDeclaration{",
8207 self.string(contents),"}"}
8208 end
```

Define `writer->inline_html_cdata_section` as a function that transforms the contents of an inline HTML CDATA section to the output format.

```
8209 function self.inline_html_cdata_section(contents)
8210 if self.flatten_inlines then return contents end
8211 return {"\\markdownRendererInlineHtmlCdataSection{",
8212 self.string(contents),"}"}
8213 end
```

Define `writer->inline_html_open_tag` as a function that transforms the contents of an inline HTML opening tag to the output format.

```
8214 function self.inline_html_open_tag(contents)
8215 if self.flatten_inlines then return contents end
8216 return {"\\markdownRendererInlineHtmlOpenTag{",
8217 self.string(contents),"}"}
8218 end
```

Define `writer->inline_html_close_tag` as a function that transforms the contents of an inline HTML closing tag to the output format.

```
8219 function self.inline_html_close_tag(contents)
8220 if self.flatten_inlines then return contents end
8221 return {"\\markdownRendererInlineHtmlCloseTag{",
8222 self.string(contents),"}"}
8223 end
```

Define `writer->inline_html_empty_tag` as a function that transforms the contents of an inline HTML self-closing (empty) tag to the output format.

```
8224 function self.inline_html_empty_tag(contents)
8225 if self.flatten_inlines then return contents end
8226 return {"\\markdownRendererInlineHtmlEmptyTag{",
8227 self.string(contents),"}"}
8228 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.

```
8229 function self.emphasis(s)
8230 if self.flatten_inlines then return s end
8231 return {"\\markdownRendererEmphasis{",s,""}
8232 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.

```
8233 function self.strong(s)
8234 if self.flatten_inlines then return s end
8235 return {"\\markdownRendererStrongEmphasis{",s,""}
8236 end
```

#### 3.1.4.12 Formatter of Tickboxes

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

```
8237 function self.tickbox(f)
8238 if f == 1.0 then
8239 return "☒ "
8240 elseif f == 0.0 then
8241 return "☐ "
8242 else
8243 return "◻ "
8244 end
8245 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.

```
8246 function self.blockquote(s)
8247 if not self.is_writing then return "" end
8248 return {"\\markdownRendererBlockQuoteBegin\\n",s,
```



```

8249 "\\markdownRendererBlockQuoteEnd "}
8250 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.

```

8251 function self.verbatim(s)
8252 if not self.is_writing then return "" end
8253 s = s:gsub("\n$", "")
8254 local name = util.cache_verbatim(options.cacheDir, s)
8255 return {"\\markdownRendererInputVerbatim{"..name.."}"}
8256 end

```

#### 3.1.4.15 Formatter of Documents

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

```

8257 function self.document(d)
8258 local buf = {"\\markdownRendererDocumentBegin\n"}
8259
8260 -- warn against the `hybrid` option
8261 if options.hybrid then
8262 local text = "The `hybrid` option has been soft-deprecated."
8263 local more = "Consider using one of the following better options "
8264 .. "for mixing TeX and markdown: `contentBlocks`, "
8265 .. "`rawAttribute`, `texComments`, `texMathDollars`, "
8266 .. "`texMathSingleBackslash`, and "
8267 .. "`texMathDoubleBackslash`. "
8268 .. "For more information, see the user manual at "
8269 .. "<https://witiko.github.io/markdown/>."
8270 table.insert(buf, self.warning(text, more))
8271 end
8272
8273 -- insert the text of the document
8274 table.insert(buf, d)
8275
8276 -- pop all attributes
8277 table.insert(buf, self.pop_attributes())
8278
8279 table.insert(buf, "\\markdownRendererDocumentEnd")
8280
8281 return buf
8282 end

```

#### 3.1.4.16 Formatter of Attributes

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

```
8283 local seen_identifiers = {}
8284 local key_value_regex = "([^\s=]+)%s*=%s*(.*)"
8285 local function normalize_attributes(attributes, auto_identifiers)
8286 -- normalize attributes
8287 local normalized_attributes = {}
8288 local has_explicit_identifiers = false
8289 local key, value
8290 for _, attribute in ipairs(attributes or {}) do
8291 if attribute:sub(1, 1) == "#" then
8292 table.insert(normalized_attributes, attribute)
8293 has_explicit_identifiers = true
8294 seen_identifiers[attribute:sub(2)] = true
8295 elseif attribute:sub(1, 1) == "." then
8296 table.insert(normalized_attributes, attribute)
8297 else
8298 key, value = attribute:match(key_value_regex)
8299 if key:lower() == "id" then
8300 table.insert(normalized_attributes, "#" .. value)
8301 elseif key:lower() == "class" then
8302 local classes = {}
8303 for class in value:gmatch("%S+") do
8304 table.insert(classes, class)
8305 end
8306 table.sort(classes)
8307 for _, class in ipairs(classes) do
8308 table.insert(normalized_attributes, "." .. class)
8309 end
8310 else
8311 table.insert(normalized_attributes, attribute)
8312 end
8313 end
8314 end
8315
8316 -- if no explicit identifiers exist, add auto identifiers
8317 if not has_explicit_identifiers and auto_identifiers ~= nil then
8318 local seen_auto_identifiers = {}
8319 for _, auto_identifier in ipairs(auto_identifiers) do
8320 if seen_auto_identifiers[auto_identifier] == nil then
8321 seen_auto_identifiers[auto_identifier] = true
8322 if seen_identifiers[auto_identifier] == nil then
8323 seen_identifiers[auto_identifier] = true
8324 table.insert(normalized_attributes,
8325 "#" .. auto_identifier)
8326 end
8327 local auto_identifier_number = 1
```

```

8328 while true do
8329 local numbered_auto_identifier = auto_identifier .. "-"
8330 .. auto_identifier_number
8331 if seen_identifiers[numbered_auto_identifier] == nil then
8332 seen_identifiers[numbered_auto_identifier] = true
8333 table.insert(normalized_attributes,
8334 "#" .. numbered_auto_identifier)
8335 break
8336 end
8337 auto_identifier_number = auto_identifier_number + 1
8338 end
8339 end
8340 end
8341 end
8342 end
8343
8344 -- sort and deduplicate normalized attributes
8345 table.sort(normalized_attributes)
8346 local seen_normalized_attributes = {}
8347 local deduplicated_normalized_attributes = {}
8348 for _, attribute in ipairs(normalized_attributes) do
8349 if seen_normalized_attributes[attribute] == nil then
8350 seen_normalized_attributes[attribute] = true
8351 table.insert(deduplicated_normalized_attributes, attribute)
8352 end
8353 end
8354
8355 return deduplicated_normalized_attributes
8356 end
8357
8358 function self.attributes(attributes, should_normalize_attributes)
8359 local normalized_attributes
8360 if should_normalize_attributes == false then
8361 normalized_attributes = attributes
8362 else
8363 normalized_attributes = normalize_attributes(attributes)
8364 end
8365
8366 local buf = {}
8367 local key, value
8368 for _, attribute in ipairs(normalized_attributes) do
8369 if attribute:sub(1, 1) == "#" then
8370 table.insert(buf, {"\\markdownRenderAttributeIdentifier{",
8371 attribute:sub(2), "}"}
8372 elseif attribute:sub(1, 1) == "." then
8373 table.insert(buf, {"\\markdownRenderAttributeClassName{",
8374 attribute:sub(2), "}"}

```

```

8375 else
8376 key, value = attribute:match(key_value_regex)
8377 table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
8378 key, "{", value, "}"})
8379 end
8380 end
8381
8382 return buf
8383 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.

```

8384 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`.

```

8385 self.attribute_type_levels = {}
8386 setmetatable(self.attribute_type_levels,
8387 { __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`.

```

8388 local function apply_attributes()
8389 local buf = {}
8390 for i = 1, #self.active_attributes do
8391 local start_output = self.active_attributes[i][3]
8392 if start_output ~= nil then
8393 table.insert(buf, start_output)
8394 end
8395 end
8396 return buf
8397 end
8398
8399 local function tear_down_attributes()
8400 local buf = {}
8401 for i = #self.active_attributes, 1, -1 do
8402 local end_output = self.active_attributes[i][4]
8403 if end_output ~= nil then
8404 table.insert(buf, end_output)
8405 end
8406 end
8407 return buf
8408 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.

```

8409 function self.push_attributes(attribute_type, attributes,
8410 start_output, end_output)
8411 local attribute_type_level
8412 = self.attribute_type_levels[attribute_type]
8413 self.attribute_type_levels[attribute_type]
8414 = attribute_type_level + 1
8415
8416 -- index attributes in a hash table for easy lookup
8417 attributes = attributes or {}
8418 for i = 1, #attributes do
8419 attributes[attributes[i]] = true
8420 end
8421
8422 local buf = {}
8423 -- handle slicing
8424 if attributes["#" .. self.slice_end_identifier] ~= nil and
8425 self.slice_end_type == "^" then
8426 if self.is_writing then
8427 table.insert(buf, self.undosep())
8428 table.insert(buf, tear_down_attributes())
8429 end
8430 self.is_writing = false
8431 end
8432 if attributes["#" .. self.slice_begin_identifier] ~= nil and
8433 self.slice_begin_type == "^" then
8434 table.insert(buf, apply_attributes())
8435 self.is_writing = true
8436 end
8437 if self.is_writing and start_output ~= nil then
8438 table.insert(buf, start_output)
8439 end
8440 table.insert(self.active_attributes,
8441 {attribute_type, attributes,
8442 start_output, end_output})
8443 return buf
8444 end
8445

```

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](#)).

```
8446 function self.pop_attributes(attribute_type)
8447 local buf = {}
8448 -- pop attributes until we find attributes of correct type
8449 -- or until no attributes remain
8450 local current_attribute_type = false
8451 while current_attribute_type ~= attribute_type and
8452 #self.active_attributes > 0 do
8453 local attributes, _, end_output
8454 current_attribute_type, attributes, _, end_output = table.unpack(
8455 self.active_attributes[#self.active_attributes])
8456 local attribute_type_level
8457 = self.attribute_type_levels[current_attribute_type]
8458 self.attribute_type_levels[current_attribute_type]
8459 = attribute_type_level - 1
8460 if self.is_writing and end_output ~= nil then
8461 table.insert(buf, end_output)
8462 end
8463 table.remove(self.active_attributes, #self.active_attributes)
8464 -- handle slicing
8465 if attributes["#" .. self.slice_end_identifier] ~= nil
8466 and self.slice_end_type == "$" then
8467 if self.is_writing then
8468 table.insert(buf, self.undosep())
8469 table.insert(buf, tear_down_attributes())
8470 end
8471 self.is_writing = false
8472 end
8473 if attributes["#" .. self.slice_begin_identifier] ~= nil and
8474 self.slice_begin_type == "$" then
8475 self.is_writing = true
8476 table.insert(buf, apply_attributes())
8477 end
8478 end
8479 return buf
8480 end
```

#### 3.1.4.18 Automatically Generated Identifiers for Headings

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

```
8481 local function create_auto_identifier(s)
8482 local buffer = {}
8483 local prev_space = false
8484 local letter_found = false
8485 local normalized_s = s
```

```

8486 if not options.unicodeNormalization
8487 or options.unicodeNormalizationForm ~= "nfc" then
8488 normalized_s = util.normalize(normalized_s, "nfc")
8489 end
8490
8491 for _, code in utf8.codes(normalized_s) do
8492 local char = utf8.char(code)
8493
8494 -- Remove everything up to the first letter.
8495 if not letter_found then
8496 local is_letter = lpeg.match(
8497 parsers.unicode.alpha,
8498 char
8499)
8500 if is_letter then
8501 letter_found = true
8502 else
8503 goto continue
8504 end
8505 end
8506
8507 -- Remove all non-alphanumeric characters, except underscores,
8508 -- hyphens, and periods.
8509 if not lpeg.match(
8510 (parsers.underscore
8511 + parsers.dash
8512 + parsers.period
8513 + parsers.unicode.word
8514 + #parsers.unicode.whitespace),
8515 char
8516) then
8517 goto continue
8518 end
8519
8520 -- Replace all spaces and newlines with hyphens.
8521 if lpeg.match(
8522 (parsers.newline
8523 + #parsers.unicode.whitespace),
8524 char
8525) then
8526 char = "-"
8527 if prev_space then
8528 goto continue
8529 else
8530 prev_space = true
8531 end
8532 else

```

```

8533 -- Case-fold all alphabetic characters.
8534 local form = nil
8535 if options.unicodeNormalization then
8536 form = options.unicodeNormalizationForm
8537 end
8538 char = util.casefold(char, form)
8539 prev_space = false
8540 end
8541
8542 table.insert(buffer, char)
8543
8544 ::continue::
8545 end
8546
8547 if prev_space then
8548 table.remove(buffer)
8549 end
8550
8551 local identifier = #buffer == 0 and "section"
8552 or table.concat(buffer, "")
8553 return identifier
8554 end

```

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

```

8555 local function create_gfm_auto_identifier(s)
8556 local buffer = {}
8557 local prev_space = false
8558 local letter_found = false
8559 local normalized_s = s
8560 if not options.unicodeNormalization
8561 or options.unicodeNormalizationForm ~= "nfc" then
8562 normalized_s = util.normalize(normalized_s, "nfc")
8563 end
8564
8565 for _, code in utf8.codes(normalized_s) do
8566 local char = utf8.char(code)
8567
8568 -- Remove everything up to the first non-space.
8569 if not letter_found then
8570 local is_letter = not lpeg.match(
8571 #parsers.unicode.whitespace,
8572 char
8573)
8574 if is_letter then
8575 letter_found = true
8576 else

```



```

8577 goto continue
8578 end
8579 end
8580
8581 -- Remove all non-alphanumeric characters, except underscores
8582 -- and hyphens.
8583 if not lpeg.match(
8584 (parsers.underscore
8585 + parsers.dash
8586 + parsers.unicode.word
8587 + #parsers.unicode.whitespace),
8588 char
8589) then
8590 prev_space = false
8591 goto continue
8592 end
8593
8594 -- Replace all spaces and newlines with hyphens.
8595 if lpeg.match(
8596 (parsers.newline
8597 + #parsers.unicode.whitespace),
8598 char
8599) then
8600 char = "-"
8601 if prev_space then
8602 goto continue
8603 else
8604 prev_space = true
8605 end
8606 else
8607 -- Case-fold all alphabetic characters.
8608 local form = nil
8609 if options.unicodeNormalization then
8610 form = options.unicodeNormalizationForm
8611 end
8612 char = util.casefold(char, form)
8613 prev_space = false
8614 end
8615
8616 table.insert(buffer, char)
8617
8618 ::continue::
8619 end
8620
8621 if prev_space then
8622 table.remove(buffer)
8623 end

```

```

8624
8625 local identifier = #buffer == 0 and "section"
8626 or table.concat(buffer, "")
8627 return identifier
8628 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.

```

8629 self.secbegin_text = "\\markdownRendererSectionBegin\n"
8630 self.secend_text = "\n\\markdownRendererSectionEnd "
8631 function self.heading(s, level, attributes)
8632 local buf = {}
8633 local flat_text, inlines = table.unpack(s)
8634
8635 -- push empty attributes for implied sections
8636 while self.attribute_type_levels["heading"] < level - 1 do
8637 table.insert(buf,
8638 self.push_attributes("heading",
8639 nil,
8640 self.secbegin_text,
8641 self.secend_text))
8642 end
8643
8644 -- pop attributes for sections that have ended
8645 while self.attribute_type_levels["heading"] >= level do
8646 table.insert(buf, self.pop_attributes("heading"))
8647 end
8648
8649 -- construct attributes for the new section
8650 local auto_identifiers = {}
8651 if self.options.autoIdentifiers then
8652 table.insert(auto_identifiers, create_auto_identifier(flat_text))
8653 end
8654 if self.options.gfmAutoIdentifiers then
8655 table.insert(auto_identifiers,
8656 create_gfm_auto_identifier(flat_text))
8657 end
8658 local normalized_attributes = normalize_attributes(attributes,
8659 auto_identifiers)
8660
8661 -- push attributes for the new section
8662 local start_output = {}
8663 local end_output = {}
8664 table.insert(start_output, self.secbegin_text)
8665 table.insert(end_output, self.secend_text)

```

```

8666
8667 table.insert(buf, self.push_attributes("heading",
8668 normalized_attributes,
8669 start_output,
8670 end_output))
8671 assert(self.attribute_type_levels["heading"] == level)
8672
8673 -- render the heading and its attributes
8674 if self.is_writing and #normalized_attributes > 0 then
8675 table.insert(buf,
8676 "\\markdownRendererHeaderAttributeContextBegin\n")
8677 table.insert(buf, self.attributes(normalized_attributes, false))
8678 end
8679
8680 local cmd
8681 level = level + options.shiftHeadings
8682 if level <= 1 then
8683 cmd = "\\markdownRendererHeadingOne"
8684 elseif level == 2 then
8685 cmd = "\\markdownRendererHeadingTwo"
8686 elseif level == 3 then
8687 cmd = "\\markdownRendererHeadingThree"
8688 elseif level == 4 then
8689 cmd = "\\markdownRendererHeadingFour"
8690 elseif level == 5 then
8691 cmd = "\\markdownRendererHeadingFive"
8692 elseif level >= 6 then
8693 cmd = "\\markdownRendererHeadingSix"
8694 else
8695 cmd = ""
8696 end
8697 if self.is_writing then
8698 table.insert(buf, {cmd, "{", inlines, "}"})
8699 end
8700
8701 if self.is_writing and #normalized_attributes > 0 then
8702 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
8703 end
8704
8705 return buf
8706 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.

```

8707 function self.get_state()

```

```

8708 return {
8709 is_writing=self.is_writing,
8710 flatten_inlines=self.flatten_inlines,
8711 active_attributes={table.unpack(self.active_attributes)},
8712 }
8713 end

```

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

```

8714 function self.set_state(s)
8715 local previous_state = self.get_state()
8716 for key, value in pairs(s) do
8717 self[key] = value
8718 end
8719 return previous_state
8720 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`.

```

8721 function self.defer_call(f)
8722 local previous_state = self.get_state()
8723 return function(...)
8724 local state = self.set_state(previous_state)
8725 local return_value = f(...)
8726 self.set_state(state)
8727 return return_value
8728 end
8729 end
8730
8731 return self
8732 end

```

### 3.1.5 Parsers

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

```

8733 parsers = {}

```

#### 3.1.5.1 Basic Parsers

```

8734 parsers.percent = P("%")
8735 parsers.at = P("@")
8736 parsers.comma = P(",")
8737 parsers.asterisk = P("*")
8738 parsers.dash = P("-")
8739 parsers.plus = P("+")

```

```

8740 parsers.underscore = P("_")
8741 parsers.period = P(".")
8742 parsers.hash = P("#")
8743 parsers.dollar = P("$")
8744 parsers.ampersand = P("&")
8745 parsers.backtick = P("`")
8746 parsers.less = P("<")
8747 parsers.more = P(">")
8748 parsers.space = P(" ")
8749 parsers.squote = P("'")
8750 parsers.dquote = P('"')
8751 parsers.lparent = P("(")
8752 parsers.rparent = P(")")
8753 parsers.lbracket = P("[")
8754 parsers.rbracket = P("]")
8755 parsers.lbrace = P("{")
8756 parsers.rbrace = P("}")
8757 parsers.circumflex = P("^")
8758 parsers.slash = P("/")
8759 parsers.equal = P("=")
8760 parsers.colon = P(":")
8761 parsers.semicolon = P(";")
8762 parsers.exclamation = P("!")
8763 parsers.pipe = P("|")
8764 parsers.tilde = P("~")
8765 parsers.backslash = P("\\")
8766 parsers.tab = P("\t")
8767 parsers.newline = P("\n")
8768
8769 parsers.digit = R("09")
8770 parsers.hexdigit = R("09","af","AF")
8771 parsers.letter = R("AZ","az")
8772 parsers.alphanumeric = R("AZ","az","09")
8773 parsers.keyword = parsers.letter
8774 * (parsers.alphanumeric + parsers.dash)^0
8775
8776 parsers.doubleasterisks = P("**")
8777 parsers.doubleunderscores = P("__")
8778 parsers.doubletildes = P("~~")
8779 parsers.fourspace = P(" ")
8780
8781 parsers.any = P(1)
8782 parsers.eof = P(-1)
8783 parsers.succeed = P(true)
8784 parsers.fail = P(false)
8785
8786 parsers.internal_punctuation = S(":;,.?")

```

```

8787 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
8788
8789 parsers.escapable = parsers.ascii_punctuation
8790 parsers.anyescaped = parsers.backslash / ""
8791 * parsers.escapable
8792 + parsers.any
8793
8794 parsers.spacechar = S("\t ")
8795 parsers.spacing = S(" \n\r\t")
8796 parsers.nonpacechar = parsers.any - parsers.spacing
8797 parsers.optionalspace = parsers.spacechar^0
8798
8799 parsers.normalchar = parsers.any - (V("SpecialChar")
8800 + parsers.spacing)
8801 parsers.nonindentspace = parsers.space^-3 * - parsers.spacechar
8802 parsers.indent = parsers.space^-3 * parsers.tab
8803 + parsers.fourspace / ""
8804 parsers.linechar = P(1 - parsers.newline)
8805
8806 parsers.blankline = parsers.optionalspace
8807 * parsers.newline / "\n"
8808 parsers.blanklines = parsers.blankline^0
8809 parsers.skipblanklines = (parsers.optionalspace
8810 * parsers.newline)^0
8811 parsers.indentedline = parsers.indent / ""
8812 * C(parsers.linechar^1
8813 * parsers.newline^-1)
8814 parsers.optionallyindentedline = parsers.indent^-1 / ""
8815 * C(parsers.linechar^1
8816 * parsers.newline^-1)
8817 parsers.sp = parsers.spacing^0
8818 parsers.spnl = parsers.optionalspace
8819 * (parsers.newline
8820 * parsers.optionalspace)^-1
8821 parsers.line = parsers.linechar^0 * parsers.newline
8822 parsers.nonemptyline = parsers.line - parsers.blankline
8823

```

### 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.

```

8824 parsers.unicode = {}
8825 parsers.unicode.preceding_punctuation = parsers.fail
8826 parsers.unicode.punctuation = parsers.fail

```

```

8827 parsers.unicode.alpha = parsers.fail
8828 parsers.unicode.numeric = parsers.fail
8829 parsers.unicode.word = parsers.fail
8830 parsers.unicode.preceding_whitespace = parsers.fail
8831 parsers.unicode.whitespace = parsers.fail
8832 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>40</sup>.

```

8833 local punctuation_of_length_n
8834 = unicode_data.categories.P[n]
8835 + unicode_data.categories.S[n]
8836 parsers.unicode.preceding_punctuation
8837 = parsers.unicode.preceding_punctuation
8838 + B(punctuation_of_length_n)
8839 parsers.unicode.punctuation
8840 = parsers.unicode.punctuation
8841 + punctuation_of_length_n

```

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

```

8842 local alpha_of_length_n = unicode_data.categories.L[n]
8843 parsers.unicode.alpha
8844 = parsers.unicode.alpha
8845 + alpha_of_length_n

```

For numeric characters, accept any characters from Unicode category N (number), similar to the character class ‘Unicode.

```

8846 local numeric_of_length_n = unicode_data.categories.N[n]
8847 parsers.unicode.numeric
8848 = parsers.unicode.numeric
8849 + 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 ‘

```

8850 local word_of_length_n
8851 = alpha_of_length_n
8852 + numeric_of_length_n
8853 + unicode_data.categories.Pc[n]
8854 parsers.unicode.word
8855 = parsers.unicode.word
8856 + 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.

```

8857 local whitespace_of_length_n = unicode_data.categories.Z[n]

```

<sup>40</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

8858 if n == 1 then
8859 whitespace_of_length_n
8860 = whitespace_of_length_n
8861 + R("\t\r")
8862 end
8863 parsers.unicode.preceding_whitespace
8864 = parsers.unicode.preceding_whitespace
8865 + B(whitespace_of_length_n)
8866 parsers.unicode.whitespace
8867 = parsers.unicode.whitespace
8868 + whitespace_of_length_n
8869 end

```

### 3.1.5.3 Parsers Used for Indentation

```

8870
8871 parsers.leader = parsers.space^-3
8872

```

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

```

8873 local function has_trail(indent_table)
8874 return indent_table ~= nil and
8875 indent_table.trail ~= nil and
8876 next(indent_table.trail) ~= nil
8877 end
8878

```

Check if indent table `indent_table` has any indents.

```

8879 local function has_indents(indent_table)
8880 return indent_table ~= nil and
8881 indent_table.indents ~= nil and
8882 next(indent_table.indents) ~= nil
8883 end
8884

```

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

```

8885 local function add_trail(indent_table, trail_info)
8886 indent_table.trail = trail_info
8887 return indent_table
8888 end
8889

```

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

```

8890 local function remove_trail(indent_table)
8891 indent_table.trail = nil
8892 return indent_table
8893 end
8894

```



Update the indent table `indent_table` by adding or removing a new indent `add`.

```
8895 local function update_indent_table(indent_table, new_indent, add)
8896 indent_table = remove_trail(indent_table)
8897
8898 if not has_indents(indent_table) then
8899 indent_table.indents = {}
8900 end
8901
8902
8903 if add then
8904 indent_table.indents[#indent_table.indents + 1] = new_indent
8905 else
8906 if indent_table.indents[#indent_table.indents].name
8907 == new_indent.name then
8908 indent_table.indents[#indent_table.indents] = nil
8909 end
8910 end
8911
8912 return indent_table
8913 end
8914
```

Remove an indent by its name `name`.

```
8915 local function remove_indent(name)
8916 local remove_indent_level =
8917 function(s, i, indent_table) -- luacheck: ignore s i
8918 indent_table = update_indent_table(indent_table, {name=name},
8919 false)
8920 return true, indent_table
8921 end
8922
8923 return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
8924 end
8925
```

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`.

```
8926 local function process_starter_spacing(indent, spacing,
8927 minimum, left_strip_length)
8928 left_strip_length = left_strip_length or 0
8929
8930 local count = 0
8931 local tab_value = 4 - (indent) % 4
8932
```

```

8933 local code_started, minimum_found = false, false
8934 local code_start, minimum_remainder = "", ""
8935
8936 local left_total_stripped = 0
8937 local full_remainder = ""
8938
8939 if spacing ~= nil then
8940 for i = 1, #spacing do
8941 local character = spacing:sub(i, i)
8942
8943 if character == "\t" then
8944 count = count + tab_value
8945 tab_value = 4
8946 elseif character == " " then
8947 count = count + 1
8948 tab_value = 4 - (1 - tab_value) % 4
8949 end
8950
8951 if (left_strip_length ~= 0) then
8952 local possible_to_strip = math.min(count, left_strip_length)
8953 count = count - possible_to_strip
8954 left_strip_length = left_strip_length - possible_to_strip
8955 left_total_stripped = left_total_stripped + possible_to_strip
8956 else
8957 full_remainder = full_remainder .. character
8958 end
8959
8960 if (minimum_found) then
8961 minimum_remainder = minimum_remainder .. character
8962 elseif (count >= minimum) then
8963 minimum_found = true
8964 minimum_remainder = minimum_remainder
8965 .. string.rep(" ", count - minimum)
8966 end
8967
8968 if (code_started) then
8969 code_start = code_start .. character
8970 elseif (count >= minimum + 4) then
8971 code_started = true
8972 code_start = code_start
8973 .. string.rep(" ", count - (minimum + 4))
8974 end
8975 end
8976 end
8977
8978 local remainder
8979 if (code_started) then

```

```

8980 remainder = code_start
8981 else
8982 remainder = string.rep(" ", count - minimum)
8983 end
8984
8985 local is_minimum = count >= minimum
8986 return {
8987 is_code = code_started,
8988 remainder = remainder,
8989 left_total_stripped = left_total_stripped,
8990 is_minimum = is_minimum,
8991 minimum_remainder = minimum_remainder,
8992 total_length = count,
8993 full_remainder = full_remainder
8994 }
8995 end
8996

```

Count the total width of all indents in the indent table `indent_table`.

```

8997 local function count_indent_tab_level(indent_table)
8998 local count = 0
8999 if not has_indents(indent_table) then
9000 return count
9001 end
9002
9003 for i=1, #indent_table.indents do
9004 count = count + indent_table.indents[i].length
9005 end
9006 return count
9007 end
9008

```

Count the total width of a delimiter `delimiter`.

```

9009 local function total_delimiter_length(delimiter)
9010 local count = 0
9011 if type(delimiter) == "string" then return #delimiter end
9012 for _, value in pairs(delimiter) do
9013 count = count + total_delimiter_length(value)
9014 end
9015 return count
9016 end
9017

```

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`.

```

9018 local function process_starter_indent(_, _, indent_table, starter,
9019 is_blank, indent_type, breakable)
9020 local last_trail = starter[1]

```

```

9021 local delimiter = starter[2]
9022 local raw_new_trail = starter[3]
9023
9024 if indent_type == "bq" and not breakable then
9025 indent_table.ignore_blockquote_blank = true
9026 end
9027
9028 if has_trail(indent_table) then
9029 local trail = indent_table.trail
9030 if trail.is_code then
9031 return false
9032 end
9033 last_trail = trail.remainder
9034 else
9035 local sp = process_starter_spacing(0, last_trail, 0, 0)
9036
9037 if sp.is_code then
9038 return false
9039 end
9040 last_trail = sp.remainder
9041 end
9042
9043 local preceding_indentation = count_indent_tab_level(indent_table) % 4
9044 local last_trail_length = #last_trail
9045 local delimiter_length = total_delimiter_length(delimiter)
9046
9047 local total_indent_level = preceding_indentation + last_trail_length
9048 + delimiter_length
9049
9050 local sp = {}
9051 if not is_blank then
9052 sp = process_starter_spacing(total_indent_level, raw_new_trail,
9053 0, 1)
9054 end
9055
9056 local del_trail_length = sp.left_total_stripped
9057 if is_blank then
9058 del_trail_length = 1
9059 elseif not sp.is_code then
9060 del_trail_length = del_trail_length + #sp.remainder
9061 end
9062
9063 local indent_length = last_trail_length + delimiter_length
9064 + del_trail_length
9065 local new_indent_info = {name=indent_type, length=indent_length}
9066
9067 indent_table = update_indent_table(indent_table, new_indent_info,

```

```

9068 true)
9069 indent_table = add_trail(indent_table,
9070 {is_code=sp.is_code,
9071 remainder=sp.remainder,
9072 total_length=sp.total_length,
9073 full_remainder=sp.full_remainder})
9074
9075 return true, indent_table
9076 end
9077

```

Return the pattern corresponding with the indent name `name`.

```

9078 local function decode_pattern(name)
9079 local delimiter = parsers.succeed
9080 if name == "bq" then
9081 delimiter = parsers.more
9082 end
9083
9084 return C(parsers.optionalspace) * C(delimiter)
9085 * C(parsers.optionalspace) * Cp()
9086 end
9087

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

9088 local function left_blank_starter(indent_table)
9089 local blank_starter_index
9090
9091 if not has_indents(indent_table) then
9092 return
9093 end
9094
9095 for i = #indent_table.indents,1,-1 do
9096 local value = indent_table.indents[i]
9097 if value.name == "li" then
9098 blank_starter_index = i
9099 else
9100 break
9101 end
9102 end
9103
9104 return blank_starter_index
9105 end
9106

```

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.

```
9107 local function traverse_indent(s, i, indent_table, is_optional,
9108 is_blank, current_line_indents)
9109 local new_index = i
9110
9111 local preceding_indentation = 0
9112 local current_trail = {}
9113
9114 local blank_starter = left_blank_starter(indent_table)
9115
9116 if current_line_indents == nil then
9117 current_line_indents = {}
9118 end
9119
9120 for index = 1, #indent_table.indents do
9121 local value = indent_table.indents[index]
9122 local pattern = decode_pattern(value.name)
9123
9124 -- match decoded pattern
9125 local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
9126 if new_indent_info == nil then
9127 local blankline_end = lpeg.match(
9128 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
9129 if is_optional or not indent_table.ignore_blockquote_blank
9130 or not blankline_end then
9131 return is_optional, new_index, current_trail,
9132 current_line_indents
9133 end
9134
9135 return traverse_indent(s, tonumber(blankline_end.pos),
9136 indent_table, is_optional, is_blank,
9137 current_line_indents)
9138 end
9139
9140 local raw_last_trail = new_indent_info[1]
9141 local delimiter = new_indent_info[2]
9142 local raw_new_trail = new_indent_info[3]
9143 local next_index = new_indent_info[4]
9144
9145 local space_only = delimiter == ""
9146
9147 -- check previous trail
9148 if not space_only and next(current_trail) == nil then
9149 local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
9150 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
9151 total_length=sp.total_length,
```

```

9152 full_remainder=sp.full_remainder}
9153 end
9154
9155 if next(current_trail) ~= nil then
9156 if not space_only and current_trail.is_code then
9157 return is_optional, new_index, current_trail,
9158 current_line_indents
9159 end
9160 if current_trail.internal_remainder ~= nil then
9161 raw_last_trail = current_trail.internal_remainder
9162 end
9163 end
9164
9165 local raw_last_trail_length = 0
9166 local delimiter_length = 0
9167
9168 if not space_only then
9169 delimiter_length = #delimiter
9170 raw_last_trail_length = #raw_last_trail
9171 end
9172
9173 local total_indent_level = preceding_indentation
9174 + raw_last_trail_length + delimiter_length
9175
9176 local spacing_to_process
9177 local minimum = 0
9178 local left_strip_length = 0
9179
9180 if not space_only then
9181 spacing_to_process = raw_new_trail
9182 left_strip_length = 1
9183 else
9184 spacing_to_process = raw_last_trail
9185 minimum = value.length
9186 end
9187
9188 local sp = process_starter_spacing(total_indent_level,
9189 spacing_to_process, minimum,
9190 left_strip_length)
9191
9192 if space_only and not sp.is_minimum then
9193 return is_optional or (is_blank and blank_starter <= index),
9194 new_index, current_trail, current_line_indents
9195 end
9196
9197 local indent_length = raw_last_trail_length + delimiter_length
9198 + sp.left_total_stripped

```

```

9199
9200 -- update info for the next pattern
9201 if not space_only then
9202 preceding_indentation = preceding_indentation + indent_length
9203 else
9204 preceding_indentation = preceding_indentation + value.length
9205 end
9206
9207 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
9208 internal_remainder=sp.minimum_remainder,
9209 total_length=sp.total_length,
9210 full_remainder=sp.full_remainder}
9211
9212 current_line_indents[#current_line_indents + 1] = new_indent_info
9213 new_index = next_index
9214 end
9215
9216 return true, new_index, current_trail, current_line_indents
9217 end
9218

```

Check if a code trail is expected.

```

9219 local function check_trail(expect_code, is_code)
9220 return (expect_code and is_code) or (not expect_code and not is_code)
9221 end
9222

```

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](#).

```

9223 local check_trail_joined =
9224 function(s, i, indent_table, -- luacheck: ignore s i
9225 spacing, expect_code, omit_remainder)
9226 local is_code
9227 local remainder
9228
9229 if has_trail(indent_table) then
9230 local trail = indent_table.trail
9231 is_code = trail.is_code
9232 if is_code then
9233 remainder = trail.remainder
9234 else
9235 remainder = trail.full_remainder
9236 end
9237 else
9238 local sp = process_starter_spacing(0, spacing, 0, 0)
9239 is_code = sp.is_code
9240 if is_code then

```



```

9241 remainder = sp.remainder
9242 else
9243 remainder = sp.full_remainder
9244 end
9245 end
9246
9247 local result = check_trail(expect_code, is_code)
9248 if omit_remainder then
9249 return result
9250 end
9251 return result, remainder
9252 end
9253

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```

9254 local check_trail_length =
9255 function(s, i, indent_table, -- luacheck: ignore s i
9256 spacing, min, max)
9257 local trail
9258
9259 if has_trail(indent_table) then
9260 trail = indent_table.trail
9261 else
9262 trail = process_starter_spacing(0, spacing, 0, 0)
9263 end
9264
9265 local total_length = trail.total_length
9266 if total_length == nil then
9267 return false
9268 end
9269
9270 return min <= total_length and total_length <= max
9271 end
9272

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

9273 local function check_continuation_indentation(s, i, indent_table,
9274 is_optional, is_blank)
9275 if not has_indents(indent_table) then
9276 return true
9277 end
9278
9279 local passes, new_index, current_trail, current_line_indents =
9280 traverse_indent(s, i, indent_table, is_optional, is_blank)
9281
9282 if passes then
9283 indent_table.current_line_indents = current_line_indents

```

```

9284 indent_table = add_trail(indent_table, current_trail)
9285 return new_index, indent_table
9286 end
9287 return false
9288 end
9289

```

Get name of the last indent from the `indent_table`.

```

9290 local function get_last_indent_name(indent_table)
9291 if has_indents(indent_table) then
9292 return indent_table.indents[#indent_table.indents].name
9293 end
9294 end
9295

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

9296 local function remove_remainder_if_blank(indent_table, remainder)
9297 if get_last_indent_name(indent_table) == "li" then
9298 return ""
9299 end
9300 return remainder
9301 end
9302

```

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.

```

9303 local check_trail_type =
9304 function(s, i, -- luacheck: ignore s i
9305 trail, spacing, trail_type)
9306 if trail == nil then
9307 trail = process_starter_spacing(0, spacing, 0, 0)
9308 end
9309
9310 if trail_type == "non-code" then
9311 return check_trail(false, trail.is_code)
9312 end
9313 if trail_type == "code" then
9314 return check_trail(true, trail.is_code)
9315 end
9316 if trail_type == "full-code" then
9317 if (trail.is_code) then
9318 return i, trail.remainder
9319 end
9320 return i, ""
9321 end
9322 if trail_type == "full-any" then
9323 return i, trail.internal_remainder
9324 end
9325 end
9326

```

```

9324 end
9325 end
9326

```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```

9327 local trail_freezing =
9328 function(s, i, -- luacheck: ignore s i
9329 indent_table, is_freezing)
9330 if is_freezing then
9331 if indent_table.is_trail_frozen then
9332 indent_table.trail = indent_table.frozen_trail
9333 else
9334 indent_table.frozen_trail = indent_table.trail
9335 indent_table.is_trail_frozen = true
9336 end
9337 else
9338 indent_table.frozen_trail = nil
9339 indent_table.is_trail_frozen = false
9340 end
9341 return true, indent_table
9342 end
9343

```

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`.

```

9344 local check_continuation_indentation_and_trail =
9345 function (s, i, indent_table, is_optional, is_blank, trail_type,
9346 reset_rem, omit_remainder)
9347 if not has_indents(indent_table) then
9348 local spacing, new_index = lpeg.match(C(parsers.spacechar^0)
9349 * Cp(), s, i)
9350 local result, remainder = check_trail_type(s, i,
9351 indent_table.trail, spacing, trail_type)
9352 if remainder == nil then
9353 if result then
9354 return new_index
9355 end
9356 return false
9357 end
9358 if result then
9359 return new_index, remainder
9360 end
9361 return false
9362 end
9363
9364 local passes, new_index, current_trail = traverse_indent(s, i,
9365 indent_table, is_optional, is_blank)

```

```

9366
9367 if passes then
9368 local spacing
9369 if current_trail == nil then
9370 local newer_spacing, newer_index = lpeg.match(
9371 C(parsers.spacechar^0) * Cp(), s, i)
9372 current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
9373 new_index = newer_index
9374 spacing = newer_spacing
9375 else
9376 spacing = current_trail.remainder
9377 end
9378 local result, remainder = check_trail_type(s, new_index,
9379 current_trail, spacing, trail_type)
9380 if remainder == nil or omit_remainder then
9381 if result then
9382 return new_index
9383 end
9384 return false
9385 end
9386
9387 if is_blank and reset_rem then
9388 remainder = remove_remainder_if_blank(indent_table, remainder)
9389 end
9390 if result then
9391 return new_index, remainder
9392 end
9393 return false
9394 end
9395 return false
9396 end
9397

```

The following patterns check whitespace indentation at the start of a block.

```

9398 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0)
9399 * Cc(false), check_trail_joined)
9400
9401 parsers.check_trail_no_rem = Cmt(Cb("indent_info")
9402 * C(parsers.spacechar^0) * Cc(false)
9403 * Cc(true), check_trail_joined)
9404
9405 parsers.check_code_trail = Cmt(Cb("indent_info")
9406 * C(parsers.spacechar^0)
9407 * Cc(true), check_trail_joined)
9408
9409 parsers.check_trail_length_range = function(min, max)
9410 return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
9411 * Cc(max), check_trail_length)

```

```

9412 end
9413
9414 parsers.check_trail_length = function(n)
9415 return parsers.check_trail_length_range(n, n)
9416 end
9417

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

9418 parsers.freeze_trail = Cg(Cmt(Cb("indent_info")
9419 * Cc(true), trail_freezing), "indent_info")
9420
9421 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
9422 trail_freezing), "indent_info")
9423

```

The following patterns check indentation in continuation lines as defined by the container start.

```

9424 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
9425 check_continuation_indentation)
9426
9427 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
9428 check_continuation_indentation)
9429
9430 parsers.check_minimal_blank_indent
9431 = Cmt(Cb("indent_info") * Cc(false)
9432 * Cc(true)
9433 , check_continuation_indentation)
9434

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

9435
9436 parsers.check_minimal_indent_and_trail =
9437 Cmt(Cb("indent_info")
9438 * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
9439 , check_continuation_indentation_and_trail)
9440
9441 parsers.check_minimal_indent_and_code_trail =
9442 Cmt(Cb("indent_info")
9443 * Cc(false) * Cc(false) * Cc("code") * Cc(false)
9444 , check_continuation_indentation_and_trail)
9445
9446 parsers.check_minimal_blank_indent_and_full_code_trail =
9447 Cmt(Cb("indent_info")
9448 * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
9449 , check_continuation_indentation_and_trail)
9450

```

```

9451 parsers.check_minimal_indent_and_any_trail =
9452 Cmt(Cb("indent_info")
9453 * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
9454 , check_continuation_indentation_and_trail)
9455
9456 parsers.check_minimal_blank_indent_and_any_trail =
9457 Cmt(Cb("indent_info")
9458 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
9459 , check_continuation_indentation_and_trail)
9460
9461 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
9462 Cmt(Cb("indent_info")
9463 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
9464 , check_continuation_indentation_and_trail)
9465
9466 parsers.check_optional_indent_and_any_trail =
9467 Cmt(Cb("indent_info")
9468 * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
9469 , check_continuation_indentation_and_trail)
9470
9471 parsers.check_optional_blank_indent_and_any_trail =
9472 Cmt(Cb("indent_info")
9473 * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
9474 , check_continuation_indentation_and_trail)
9475

```

The following patterns specify behaviour around newlines.

```

9476
9477 parsers.spnlc_noexc = parsers.optionalspace
9478 * (parsers.newline
9479 * parsers.check_minimal_indent_and_any_trail)^-1
9480
9481 parsers.spnlc = parsers.optionalspace
9482 * (V("EndlineNoSub"))^-1
9483
9484 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
9485 + parsers.spacechar^1
9486
9487 parsers.only_blank = parsers.spacechar^0
9488 * (parsers.newline + parsers.eof)
9489

```

The `parsers.commented\_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 8.

```

9490 parsers.commented_line_letter = parsers.linechar
9491 + parsers.newline
9492 - parsers.backslash
9493 - parsers.percent

```



```

9494 parsers.commented_line = Cg(Cc(""), "backslashes")
9495 * Cg(Cc(true), "pure_comment")
9496 * ((#(parsers.commented_line_letter
9497 - parsers.newline)
9498 * Cb("backslashes")
9499 * Cs(parsers.commented_line_letter
9500 - parsers.newline)^1 -- initial
9501 * Cg(Cc(""), "backslashes")
9502 * Cg(Cc(false), "pure_comment"))
9503 + #(parsers.backslash
9504 * (parsers.backslash + parsers.newline))
9505 * Cg((parsers.backslash -- even backslash
9506 * (parsers.backslash
9507 + #parsers.newline))^1, "backslashes")
9508 * Cg(Cc(false), "pure_comment")
9509 + (parsers.backslash
9510 * (#parsers.percent
9511 * Cb("backslashes")
9512 / function(backslashes)
9513 return string.rep("\\", #backslashes / 2)
9514 end
9515 * C(parsers.percent)
9516 + #parsers.commented_line_letter
9517 * Cb("backslashes")
9518 * Cc("\\")
9519 * C(parsers.commented_line_letter))
9520 * Cg(Cc(""), "backslashes")
9521 * Cg(Cc(false), "pure_comment"))^0
9522 * (#parsers.percent
9523 * Cb("backslashes")
9524 / function(backslashes)
9525 return string.rep("\\", #backslashes / 2)
9526 end
9527 * ((Cb("pure_comment")
9528 * parsers.percent -- comment
9529 * parsers.line
9530 * #parsers.blankline) -- blank line
9531 / function(is_pure)
9532 return is_pure and "" or "\n"
9533 end
9534 + parsers.percent -- comment
9535 * parsers.line
9536 * parsers.optionalspace) -- leading spaces
9537 + #(parsers.newline)
9538 * Cb("backslashes")
9539 * C(parsers.newline))
9540

```



```

9541 parsers.chunk = parsers.line * (parsers.optionallyindentedline
9542 - parsers.blankline)^0
9543
9544 parsers.attribute_key_char = parsers.unicode.alpha
9545 + parsers.unicode.numeric
9546 + S("_:.")
9547 parsers.attribute_raw_char = parsers.unicode.alpha
9548 + parsers.unicode.numeric
9549 + S("-_")
9550 parsers.attribute_key = (parsers.attribute_key_char
9551 - parsers.dash - parsers.digit)
9552 * parsers.attribute_key_char^0
9553 parsers.attribute_value = ((parsers.dquote / "\"")
9554 * (parsers.anyescaped - parsers.dquote)^0
9555 * (parsers.dquote / "\""))
9556 + ((parsers.squote / "\"")
9557 * (parsers.anyescaped - parsers.squote)^0
9558 * (parsers.squote / "\""))
9559 + (parsers.anyescaped
9560 - parsers.dquote
9561 - parsers.rbrace
9562 - parsers.space)^0
9563 parsers.attribute_identifier = parsers.attribute_key_char^1
9564 parsers.attribute_classname = parsers.unicode.alpha
9565 * parsers.attribute_key_char^0
9566 parsers.attribute_raw = parsers.attribute_raw_char^1
9567
9568 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
9569 + C(parsers.hash
9570 * parsers.attribute_identifier)
9571 + C(parsers.period
9572 * parsers.attribute_classname)
9573 + Cs(parsers.attribute_key
9574 * parsers.optionalspace
9575 * parsers.equal
9576 * parsers.optionalspace
9577 * parsers.attribute_value)
9578 parsers.attributes = parsers.lbrace
9579 * parsers.optionalspace
9580 * parsers.attribute
9581 * (parsers.spacechar^1
9582 * parsers.attribute)^0
9583 * parsers.optionalspace
9584 * parsers.rbrace
9585
9586 parsers.raw_attribute = parsers.lbrace
9587 * parsers.optionalspace

```

```

9588 * parsers.equal
9589 * C(parsers.attribute_raw)
9590 * parsers.optionalspace
9591 * parsers.rbrace
9592
9593 -- block followed by 0 or more optionally
9594 -- indented blocks with first line indented.
9595 parsers.indented_blocks = function(bl)
9596 return Cs(bl
9597 * (parsers.blankline~1
9598 * parsers.indent
9599 * -parsers.blankline
9600 * bl)^0
9601 * (parsers.blankline~1 + parsers.eof))
9602 end

```

### 3.1.5.4 Parsers Used for html Entities

```

9603 local function repeat_between(pattern, min, max)
9604 return -pattern^(max + 1) * pattern^min
9605 end
9606
9607 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
9608 * C(repeat_between(parsers.hexdigit, 1, 6))
9609 * parsers.semicolon
9610 parsers.decentity = parsers.ampersand * parsers.hash
9611 * C(repeat_between(parsers.digit, 1, 7))
9612 * parsers.semicolon
9613 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric~1)
9614 * parsers.semicolon
9615
9616 parsers.html_entities
9617 = parsers.hexentity / entities.hex_entity_with_x_char
9618 + parsers.decentity / entities.dec_entity
9619 + parsers.tagentity / entities.char_entity
9620 parsers.html_space_entity_char
9621 = Cmt(parsers.html_entities, function(_, i, entity)
9622 return entity == " " and i or nil
9623 end)

```

### 3.1.5.5 Parsers Used for Markdown Lists

```

9624 parsers.bullet = function(bullet_char, interrupting)
9625 local allowed_end
9626 if interrupting then
9627 allowed_end = C(parsers.spacechar~1) * #parsers.linechar
9628 else
9629 allowed_end = C(parsers.spacechar~1)

```

```

9630 + #(parsers.newline + parsers.eof)
9631 end
9632 return parsers.check_trail
9633 * Ct(C(bullet_char) * Cc(""))
9634 * allowed_end
9635 end
9636
9637 local function tickbox(interior)
9638 return parsers.optionalspace * parsers.lbracket
9639 * interior * parsers.rbracket * parsers.spacechar^1
9640 end
9641
9642 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
9643 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
9644 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
9645

```

### 3.1.5.6 Parsers Used for Markdown Code Spans

```

9646 parsers.openticks = Cg(parsers.backtick^1, "ticks")
9647
9648 local function captures_equal_length(_,i,a,b)
9649 return #a == #b and i
9650 end
9651
9652 parsers.closeticks = Cmt(C(parsers.backtick^1)
9653 * Cb("ticks"), captures_equal_length)
9654
9655 parsers.intickschar = (parsers.any - S("\n\r`"))
9656 + V("NoSoftLineBreakEndline")
9657 + (parsers.backtick^1 - parsers.closeticks)
9658
9659 local function process_inticks(s)
9660 s = s:gsub("\n", " ")
9661 s = s:gsub("^ (.*) $", "%1")
9662 return s
9663 end
9664
9665 parsers.inticks = parsers.openticks
9666 * C(parsers.space^0)
9667 * parsers.closeticks
9668 + parsers.openticks
9669 * Cs(Cs(parsers.intickschar^0) / process_inticks)
9670 * parsers.closeticks
9671

```

### 3.1.5.7 Parsers Used for html

```

9672 -- case-insensitive match (we assume s is lowercase)
9673 -- must be single byte encoding
9674 parsers.keyword_exact = function(s)
9675 local parser = P(0)
9676 for i=1,#s do
9677 local c = s:sub(i,i)
9678 local m = c .. upper(c)
9679 parser = parser * S(m)
9680 end
9681 return parser
9682 end
9683
9684 parsers.special_block_keyword =
9685 parsers.keyword_exact("pre") +
9686 parsers.keyword_exact("script") +
9687 parsers.keyword_exact("style") +
9688 parsers.keyword_exact("textarea")
9689
9690 parsers.block_keyword =
9691 parsers.keyword_exact("address") +
9692 parsers.keyword_exact("article") +
9693 parsers.keyword_exact("aside") +
9694 parsers.keyword_exact("base") +
9695 parsers.keyword_exact("basefont") +
9696 parsers.keyword_exact("blockquote") +
9697 parsers.keyword_exact("body") +
9698 parsers.keyword_exact("caption") +
9699 parsers.keyword_exact("center") +
9700 parsers.keyword_exact("col") +
9701 parsers.keyword_exact("colgroup") +
9702 parsers.keyword_exact("dd") +
9703 parsers.keyword_exact("details") +
9704 parsers.keyword_exact("dialog") +
9705 parsers.keyword_exact("dir") +
9706 parsers.keyword_exact("div") +
9707 parsers.keyword_exact("dl") +
9708 parsers.keyword_exact("dt") +
9709 parsers.keyword_exact("fieldset") +
9710 parsers.keyword_exact("figcaption") +
9711 parsers.keyword_exact("figure") +
9712 parsers.keyword_exact("footer") +
9713 parsers.keyword_exact("form") +
9714 parsers.keyword_exact("frame") +
9715 parsers.keyword_exact("frameset") +
9716 parsers.keyword_exact("h1") +
9717 parsers.keyword_exact("h2") +
9718 parsers.keyword_exact("h3") +

```

```

9719 parsers.keyword_exact("h4") +
9720 parsers.keyword_exact("h5") +
9721 parsers.keyword_exact("h6") +
9722 parsers.keyword_exact("head") +
9723 parsers.keyword_exact("header") +
9724 parsers.keyword_exact("hr") +
9725 parsers.keyword_exact("html") +
9726 parsers.keyword_exact("iframe") +
9727 parsers.keyword_exact("legend") +
9728 parsers.keyword_exact("li") +
9729 parsers.keyword_exact("link") +
9730 parsers.keyword_exact("main") +
9731 parsers.keyword_exact("menu") +
9732 parsers.keyword_exact("menuitem") +
9733 parsers.keyword_exact("nav") +
9734 parsers.keyword_exact("noframes") +
9735 parsers.keyword_exact("ol") +
9736 parsers.keyword_exact("optgroup") +
9737 parsers.keyword_exact("option") +
9738 parsers.keyword_exact("p") +
9739 parsers.keyword_exact("param") +
9740 parsers.keyword_exact("section") +
9741 parsers.keyword_exact("source") +
9742 parsers.keyword_exact("summary") +
9743 parsers.keyword_exact("table") +
9744 parsers.keyword_exact("tbody") +
9745 parsers.keyword_exact("td") +
9746 parsers.keyword_exact("tfoot") +
9747 parsers.keyword_exact("th") +
9748 parsers.keyword_exact("thead") +
9749 parsers.keyword_exact("title") +
9750 parsers.keyword_exact("tr") +
9751 parsers.keyword_exact("track") +
9752 parsers.keyword_exact("ul")
9753
9754 -- end conditions
9755 parsers.html_blankline_end_condition
9756 = parsers.linechar^0
9757 * (parsers.newline
9758 * (parsers.check_minimal_blank_indent_and_any_trail
9759 * #parsers.blankline
9760 + parsers.check_minimal_indent_and_any_trail)
9761 * parsers.linechar^1)^0
9762 * (parsers.newline^-1 / "")
9763
9764 local function remove_trailing_blank_lines(s)
9765 return s:gsub("[\n\r]+%s*$", "")

```

```

9766 end
9767
9768 parsers.html_until_end = function(end_marker)
9769 return (
9770 Cs(C(parsers.newline
9771 * (parsers.check_minimal_blank_indent_and_any_trail
9772 * #parsers.blankline
9773 + parsers.check_minimal_indent_and_any_trail)
9774 + parsers.linechar - end_marker)^0)
9775 * (C(parsers.linechar^0 * parsers.newline^-1)
9776 / remove_trailing_blank_lines)
9777)
9778 end
9779
9780 -- attributes
9781 parsers.html_attribute_spacing = parsers.optionalspace
9782 * V("NoSoftLineBreakEndline")
9783 * parsers.optionalspace
9784 + parsers.spacechar^1
9785
9786 parsers.html_attribute_name = (parsers.letter
9787 + parsers.colon
9788 + parsers.underscore)
9789 * (parsers.alphanumeric
9790 + parsers.colon
9791 + parsers.underscore
9792 + parsers.period
9793 + parsers.dash)^0
9794
9795 parsers.html_attribute_value = parsers.squote
9796 * (parsers.linechar - parsers.squote)^0
9797 * parsers.squote
9798 + parsers.dquote
9799 * (parsers.linechar - parsers.dquote)^0
9800 * parsers.dquote
9801 + (parsers.any
9802 - parsers.spacechar
9803 - parsers.newline
9804 - parsers.dquote
9805 - parsers.squote
9806 - parsers.backtick
9807 - parsers.equal
9808 - parsers.less
9809 - parsers.more)^1
9810
9811 parsers.html_inline_attribute_value = parsers.squote
9812 * (V("NoSoftLineBreakEndline")

```

```

9813 + parsers.any
9814 - parsers.blankline^2
9815 - parsers.squote)^0
9816 * parsers.squote
9817 + parsers.dquote
9818 * (V("NoSoftLineBreakEndline")
9819 + parsers.any
9820 - parsers.blankline^2
9821 - parsers.dquote)^0
9822 * parsers.dquote
9823 + (parsers.any
9824 - parsers.spacechar
9825 - parsers.newline
9826 - parsers.dquote
9827 - parsers.squote
9828 - parsers.backtick
9829 - parsers.equal
9830 - parsers.less
9831 - parsers.more)^1
9832
9833 parsers.html_attribute_value_specification
9834 = parsers.optionalspace
9835 * parsers.equal
9836 * parsers.optionalspace
9837 * parsers.html_attribute_value
9838
9839 parsers.html_spnl = parsers.optionalspace
9840 * (V("NoSoftLineBreakEndline")
9841 * parsers.optionalspace)^-1
9842
9843 parsers.html_inline_attribute_value_specification
9844 = parsers.html_spnl
9845 * parsers.equal
9846 * parsers.html_spnl
9847 * parsers.html_inline_attribute_value
9848
9849 parsers.html_attribute
9850 = parsers.html_attribute_spacing
9851 * parsers.html_attribute_name
9852 * parsers.html_inline_attribute_value_specification^-1
9853
9854 parsers.html_non_newline_attribute
9855 = parsers.spacechar^1
9856 * parsers.html_attribute_name
9857 * parsers.html_attribute_value_specification^-1
9858
9859 parsers.nested_breaking_blank = parsers.newline

```

```

9860 * parsers.check_minimal_blank_indent
9861 * parsers.blankline
9862
9863 parsers.html_comment_start = P("<!--")
9864 * parsers.optionalspace
9865
9866 parsers.html_comment_end = parsers.optionalspace
9867 * P("-->")
9868
9869 parsers.html_comment
9870 = parsers.html_comment_start
9871 * parsers.html_until_end(parsers.html_comment_end)
9872
9873 parsers.html_inline_comment = (parsers.html_comment_start / "")
9874 * -P(">") * -P("->")
9875 * Cs((parsers.any
9876 - parsers.nested_breaking_blank
9877 - parsers.html_comment_end)^0)
9878 * (parsers.html_comment_end / "")
9879
9880 parsers.html_cdatasection_start = P("<![CDATA[")
9881
9882 parsers.html_cdatasection_end = P("]]>")
9883
9884 parsers.html_cdatasection
9885 = Cs(parsers.html_cdatasection_start
9886 * parsers.html_until_end(parsers.html_cdatasection_end))
9887 / remove_trailing_blank_lines
9888
9889 parsers.html_inline_cdatasection
9890 = parsers.html_cdatasection_start
9891 * Cs(V("NoSoftLineBreakEndline") + parsers.any
9892 - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
9893 * parsers.html_cdatasection_end
9894
9895 parsers.html_declaration_start = P("<!") * parsers.letter
9896
9897 parsers.html_declaration_end = P(">")
9898
9899 parsers.html_declaration
9900 = Cs(parsers.html_declaration_start
9901 * parsers.html_until_end(parsers.html_declaration_end))
9902 / remove_trailing_blank_lines
9903
9904 parsers.html_inline_declaration
9905 = parsers.html_declaration_start
9906 * Cs(V("NoSoftLineBreakEndline") + parsers.any

```



```

9907 - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
9908 * parsers.html_declaration_end
9909
9910 parsers.html_instruction_start = P("<?")
9911
9912 parsers.html_instruction_end = P("?>")
9913
9914 parsers.html_instruction
9915 = Cs(parsers.html_instruction_start
9916 * parsers.html_until_end(parsers.html_instruction_end))
9917 / remove_trailing_blank_lines
9918
9919 parsers.html_inline_instruction = parsers.html_instruction_start
9920 * Cs(V("NoSoftLineBreakEndline")
9921 + parsers.any
9922 - parsers.nested_breaking_blank
9923 - parsers.html_instruction_end)^0
9924 * parsers.html_instruction_end
9925
9926 parsers.html_blankline = parsers.newline
9927 * parsers.optionalspace
9928 * parsers.newline
9929
9930 parsers.html_tag_start = parsers.less
9931
9932 parsers.html_tag_closing_start = parsers.less
9933 * parsers.slash
9934
9935 parsers.html_tag_end = parsers.html_spnl
9936 * parsers.more
9937
9938 parsers.html_empty_tag_end = parsers.html_spnl
9939 * parsers.slash
9940 * parsers.more
9941
9942 -- opening tags
9943 parsers.html_any_open_inline_tag = parsers.html_tag_start
9944 * parsers.keyword
9945 * parsers.html_attribute^0
9946 * parsers.html_tag_end
9947
9948 parsers.html_any_open_tag = parsers.html_tag_start
9949 * parsers.keyword
9950 * parsers.html_non_newline_attribute^0
9951 * parsers.html_tag_end
9952
9953 parsers.html_open_tag = parsers.html_tag_start

```

```

9954 * parsers.block_keyword
9955 * parsers.html_attribute^0
9956 * parsers.html_tag_end
9957
9958 parsers.html_open_special_tag = parsers.html_tag_start
9959 * parsers.special_block_keyword
9960 * parsers.html_attribute^0
9961 * parsers.html_tag_end
9962
9963 -- incomplete tags
9964 parsers.incomplete_tag_following = parsers.spacechar
9965 + parsers.more
9966 + parsers.slash * parsers.more
9967 + #(parsers.newline + parsers.eof)
9968
9969 parsers.incomplete_special_tag_following = parsers.spacechar
9970 + parsers.more
9971 + #(parsers.newline
9972 + parsers.eof)
9973
9974 parsers.html_incomplete_open_tag = parsers.html_tag_start
9975 * parsers.block_keyword
9976 * parsers.incomplete_tag_following
9977
9978 parsers.html_incomplete_open_special_tag
9979 = parsers.html_tag_start
9980 * parsers.special_block_keyword
9981 * parsers.incomplete_special_tag_following
9982
9983 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
9984 * parsers.block_keyword
9985 * parsers.incomplete_tag_following
9986
9987 parsers.html_incomplete_close_special_tag
9988 = parsers.html_tag_closing_start
9989 * parsers.special_block_keyword
9990 * parsers.incomplete_tag_following
9991
9992 -- closing tags
9993 parsers.html_close_tag = parsers.html_tag_closing_start
9994 * parsers.block_keyword
9995 * parsers.html_tag_end
9996
9997 parsers.html_any_close_tag = parsers.html_tag_closing_start
9998 * parsers.keyword
9999 * parsers.html_tag_end
10000

```

```

10001 parsers.html_close_special_tag = parsers.html_tag_closing_start
10002 * parsers.special_block_keyword
10003 * parsers.html_tag_end
10004
10005 -- empty tags
10006 parsers.html_any_empty_inline_tag = parsers.html_tag_start
10007 * parsers.keyword
10008 * parsers.html_attribute^0
10009 * parsers.html_empty_tag_end
10010
10011 parsers.html_any_empty_tag = parsers.html_tag_start
10012 * parsers.keyword
10013 * parsers.html_non_newline_attribute^0
10014 * parsers.optionalspace
10015 * parsers.slash
10016 * parsers.more
10017
10018 parsers.html_empty_tag = parsers.html_tag_start
10019 * parsers.block_keyword
10020 * parsers.html_attribute^0
10021 * parsers.html_empty_tag_end
10022
10023 parsers.html_empty_special_tag = parsers.html_tag_start
10024 * parsers.special_block_keyword
10025 * parsers.html_attribute^0
10026 * parsers.html_empty_tag_end
10027
10028 parsers.html_incomplete_blocks
10029 = parsers.html_incomplete_open_tag
10030 + parsers.html_incomplete_open_special_tag
10031 + parsers.html_incomplete_close_tag
10032
10033 -- parse special html blocks
10034 parsers.html_blankline_ending_special_block_opening
10035 = (parsers.html_close_special_tag
10036 + parsers.html_empty_special_tag)
10037 * #(parsers.optionalspace
10038 * (parsers.newline + parsers.eof))
10039
10040 parsers.html_blankline_ending_special_block
10041 = parsers.html_blankline_ending_special_block_opening
10042 * parsers.html_blankline_end_condition
10043
10044 parsers.html_special_block_opening
10045 = parsers.html_incomplete_open_special_tag
10046 - parsers.html_empty_special_tag
10047

```

```

10048 parsers.html_closing_special_block
10049 = Cs(parsers.html_special_block_opening
10050 * parsers.html_until_end(parsers.html_close_special_tag))
10051 / remove_trailing_blank_lines
10052
10053 parsers.html_special_block
10054 = parsers.html_blankline_ending_special_block
10055 + parsers.html_closing_special_block
10056
10057 -- parse html blocks
10058 parsers.html_block_opening = parsers.html_incomplete_open_tag
10059 + parsers.html_incomplete_close_tag
10060
10061 parsers.html_block = parsers.html_block_opening
10062 * parsers.html_blankline_end_condition
10063
10064 -- parse any html blocks
10065 parsers.html_any_block_opening
10066 = (parsers.html_any_open_tag
10067 + parsers.html_any_close_tag
10068 + parsers.html_any_empty_tag)
10069 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
10070
10071 parsers.html_any_block = parsers.html_any_block_opening
10072 * parsers.html_blankline_end_condition
10073
10074 parsers.html_inline_comment_full = parsers.html_comment_start
10075 * -P(">") * -P("->")
10076 * Cs((parsers.any - P("--")
10077 - parsers.nested_breaking_blank
10078 - parsers.html_comment_end)^0)
10079 * parsers.html_comment_end
10080
10081 parsers.html_inline_tags = parsers.html_inline_comment_full
10082 + parsers.html_any_empty_inline_tag
10083 + parsers.html_inline_instruction
10084 + parsers.html_inline_cdatasection
10085 + parsers.html_inline_declaration
10086 + parsers.html_any_open_inline_tag
10087 + parsers.html_any_close_tag
10088

```

### 3.1.5.8 Parsers Used for Markdown Tags and Links

```

10089 parsers.urlchar = parsers.anyescaped
10090 - parsers.newline
10091 - parsers.more

```

```

10092
10093 parsers.auto_link_scheme_part = parsers.alphanumeric
10094 + parsers.plus
10095 + parsers.period
10096 + parsers.dash
10097
10098 parsers.auto_link_scheme = parsers.letter
10099 * parsers.auto_link_scheme_part
10100 * parsers.auto_link_scheme_part~-30
10101
10102 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
10103 * (parsers.any - parsers.spacing
10104 - parsers.less - parsers.more)^0
10105
10106 parsers.printable_characters = S("!.#$_%&'*/+/?^_`{|}~-")
10107
10108 parsers.email_address_local_part_char = parsers.alphanumeric
10109 + parsers.printable_characters
10110
10111 parsers.email_address_local_part
10112 = parsers.email_address_local_part_char^1
10113
10114 parsers.email_address_dns_label = parsers.alphanumeric
10115 * (parsers.alphanumeric
10116 + parsers.dash)^-62
10117 * B(parsers.alphanumeric)
10118
10119 parsers.email_address_domain = parsers.email_address_dns_label
10120 * (parsers.period
10121 * parsers.email_address_dns_label)^0
10122
10123 parsers.email_address = parsers.email_address_local_part
10124 * parsers.at
10125 * parsers.email_address_domain
10126
10127 parsers.auto_link_url = parsers.less
10128 * C(parsers.absolute_uri)
10129 * parsers.more
10130
10131 parsers.auto_link_email = parsers.less
10132 * C(parsers.email_address)
10133 * parsers.more
10134
10135 parsers.auto_link_relative_reference = parsers.less
10136 * C(parsers.urlchar^1)
10137 * parsers.more
10138

```

```

10139 parsers.autolink = parsers.auto_link_url
10140 + parsers.auto_link_email
10141
10142 -- content in balanced brackets, parentheses, or quotes:
10143 parsers.bracketed = P{ parsers.lbracket
10144 * ((parsers.backslash / "\"" * parsers.rbracket
10145 + parsers.any - (parsers.lbracket
10146 + parsers.rbracket
10147 + parsers.blankline^2)
10148) + V(1))^0
10149 * parsers.rbracket }
10150
10151 parsers.inparens = P{ parsers.lparent
10152 * ((parsers.anyescaped - (parsers.lparent
10153 + parsers.rparent
10154 + parsers.blankline^2)
10155) + V(1))^0
10156 * parsers.rparent }
10157
10158 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
10159 * ((parsers.anyescaped - (parsers.squote
10160 + parsers.blankline^2)
10161) + V(1))^0
10162 * parsers.squote }
10163
10164 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
10165 * ((parsers.anyescaped - (parsers.dquote
10166 + parsers.blankline^2)
10167) + V(1))^0
10168 * parsers.dquote }
10169
10170 parsers.link_text = parsers.lbracket
10171 * Cs((parsers.alphanumeric^1
10172 + parsers.bracketed
10173 + parsers.inticks
10174 + parsers.autolink
10175 + V("InlineHtml")
10176 + (parsers.backslash * parsers.backslash)
10177 + (parsers.backslash
10178 * (parsers.lbracket
10179 + parsers.rbracket)
10180 + V("Space")
10181 + V("NoSoftLineBreakEndline")
10182 + (parsers.any
10183 - (parsers.newline
10184 + parsers.lbracket
10185 + parsers.rbracket

```

```

10186 + parsers.blankline^2))))^0)
10187 * parsers.rbracket
10188
10189 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
10190 * #((parsers.any
10191 - parsers.rbracket)^-999
10192 * parsers.rbracket)
10193 * Cs((parsers.alphanumeric^1
10194 + parsers.inticks
10195 + parsers.autolink
10196 + V("InlineHtml")
10197 + (parsers.backslash * parsers.backslash)
10198 + (parsers.backslash
10199 * (parsers.lbracket
10200 + parsers.rbracket)
10201 + V("Space")
10202 + V("NoSoftLineBreakEndline")
10203 + (parsers.any
10204 - (parsers.newline
10205 + parsers.lbracket
10206 + parsers.rbracket
10207 + parsers.blankline^2))))^1)
10208
10209 parsers.link_label = parsers.lbracket
10210 * parsers.link_label_body
10211 * parsers.rbracket
10212
10213 parsers.inparens_url = P{ parsers.lparent
10214 * ((parsers.anyescaped - (parsers.lparent
10215 + parsers.rparent
10216 + parsers.spacing)
10217) + V(1))^0
10218 * parsers.rparent }
10219
10220 -- url for markdown links, allowing nested brackets:
10221 parsers.url = parsers.less * Cs((parsers.anyescaped
10222 - parsers.newline
10223 - parsers.less
10224 - parsers.more)^0)
10225 * parsers.more
10226 + -parsers.less
10227 * Cs((parsers.inparens_url + (parsers.anyescaped
10228 - parsers.spacing
10229 - parsers.lparent
10230 - parsers.rparent))^1)
10231
10232 -- quoted text:

```

```

10233 parsers.title_s = parsers.squote
10234 * Cs((parsers.html_entities
10235 + V("Space")
10236 + V("NoSoftLineBreakEndline")
10237 + (parsers.anyescaped
10238 - parsers.newline
10239 - parsers.squote
10240 - parsers.blankline^2))^0)
10241 * parsers.squote
10242
10243 parsers.title_d = parsers.dquote
10244 * Cs((parsers.html_entities
10245 + V("Space")
10246 + V("NoSoftLineBreakEndline")
10247 + (parsers.anyescaped
10248 - parsers.newline
10249 - parsers.dquote
10250 - parsers.blankline^2))^0)
10251 * parsers.dquote
10252
10253 parsers.title_p = parsers.lparent
10254 * Cs((parsers.html_entities
10255 + V("Space")
10256 + V("NoSoftLineBreakEndline")
10257 + (parsers.anyescaped
10258 - parsers.newline
10259 - parsers.lparent
10260 - parsers.rparent
10261 - parsers.blankline^2))^0)
10262 * parsers.rparent
10263
10264 parsers.title =
10265 = parsers.title_d + parsers.title_s + parsers.title_p
10266
10267 parsers.optionaltitle =
10268 = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
10269

```

### 3.1.5.9 Helpers for Links and Link Reference Definitions

```

10270 -- parse a reference definition: [foo]: /bar "title"
10271 parsers.define_reference_parser = (parsers.check_trail / "")
10272 * parsers.link_label * parsers.colon
10273 * parsers.spnlc * parsers.url
10274 * (parsers.spnlc_sep * parsers.title
10275 * parsers.only_blank
10276 + Cc("") * parsers.only_blank)

```



### 3.1.5.10 Inline Elements

```
10277 parsers.Inline = V("Inline")
10278
10279 -- parse many p between starter and ender
10280 parsers.between = function(p, starter, ender)
10281 local ender2 = B(parsers.nonspacechar) * ender
10282 return (starter
10283 * #parsers.nonspacechar
10284 * Ct(p * (p - ender2)^0)
10285 * ender2)
10286 end
10287
```

### 3.1.5.11 Block Elements

```
10288 parsers.lineof = function(c)
10289 return (parsers.check_trail_no_rem
10290 * (P(c) * parsers.optionalspace)^3
10291 * (parsers.newline + parsers.eof))
10292 end
10293
10294 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
10295 + parsers.lineof(parsers.dash)
10296 + parsers.lineof(parsers.underscore)
```

### 3.1.5.12 Headings

```
10297 -- parse Atx heading start and return level
10298 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
10299 * -parsers.hash / length
10300
10301 -- parse setext header ending and return level
10302 parsers.heading_level
10303 = parsers.nonindentSPACE * parsers.equal^1
10304 * parsers.optionalspace * #parsers.newline * Cc(1)
10305 + parsers.nonindentSPACE * parsers.dash^1
10306 * parsers.optionalspace * #parsers.newline * Cc(2)
10307
10308 local function strip_atx_end(s)
10309 return s:gsub("%s+##%s*\n$", "")
10310 end
10311
10312 parsers.atx_heading = parsers.check_trail_no_rem
10313 * Cg(parsers.heading_start, "level")
10314 * (C(parsers.optionalspace
10315 * parsers.hash^0
10316 * parsers.optionalspace
10317 * parsers.newline)
```

```

10318 + parsers.spacechar^1
10319 * 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`.

```

10320 M.reader = {}
10321 function M.reader.new(writer, options)
10322 local self = {}

```

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

```

10323 self.writer = writer
10324 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.

```

10325 self.parsers = {}
10326 (function(parsers)
10327 setmetatable(self.parsers, {
10328 __index = function (_, key)
10329 return parsers[key]
10330 end
10331 })
10332 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.

```

10333 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.

```

10334 function self.normalize_tag(tag)
10335 tag = util.rope_to_string(tag)

```

```

10336 tag = tag:gsub("[\n\r\t]+", " ")
10337 tag = tag:gsub("^ ", ""):gsub(" $", "")
10338 local form = nil
10339 if options.unicodeNormalization then
10340 form = options.unicodeNormalizationForm
10341 end
10342 tag = util.casefold(tag, form)
10343 return tag
10344 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.

```

10345 local function iterlines(s, f)
10346 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
10347 return util.rope_to_string(rope)
10348 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.

```

10349 if options.preserveTabs then
10350 self.expandtabs = function(s) return s end
10351 else
10352 self.expandtabs = function(s)
10353 if s:find("\t") then
10354 return iterlines(s, util.expand_tabs_in_line)
10355 else
10356 return s
10357 end
10358 end
10359 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.

```

10360 self.parser_functions = {}
10361 self.create_parser = function(name, grammar, toplevel)
10362 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.

```

10363 if toplevel and options.stripIndent then
10364 local min_prefix_length, min_prefix = nil, ''
10365 str = iterlines(str, function(line)
10366 if lpeg.match(parsers.nonemptyline, line) == nil then
10367 return line
10368 end
10369 line = util.expand_tabs_in_line(line)
10370 local prefix = lpeg.match(C(parsers.optionalspace), line)
10371 local prefix_length = #prefix
10372 local is_shorter = min_prefix_length == nil
10373 if not is_shorter then
10374 is_shorter = prefix_length < min_prefix_length
10375 end
10376 if is_shorter then
10377 min_prefix_length, min_prefix = prefix_length, prefix
10378 end
10379 return line
10380 end)
10381 str = str:gsub('^' .. min_prefix, '')
10382 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.

```

10383 if toplevel and (options.texComments or options.hybrid) then
10384 str = lpeg.match(Ct(parsers.commented_line^1), str)
10385 str = util.rope_to_string(str)
10386 end
10387 local res = lpeg.match(grammar(), str)
10388 if res == nil then
10389 return writer.error(
10390 format("Parser `%s` failed to process the input text.", name),
10391 format("Here are the first 20 characters of the remaining "
10392 .. "unprocessed text: `%s`.", str:sub(1,20))
10393)
10394 else
10395 return res
10396 end
10397 end
10398 end
10399
10400 self.create_parser("parse_blocks",
10401 function()
10402 return parsers.blocks
10403 end, true)
10404
10405 self.create_parser("parse_blocks_nested",

```

```

10406 function()
10407 return parsers.blocks_nested
10408 end, false)
10409
10410 self.create_parser("parse_inlines",
10411 function()
10412 return parsers.inlines
10413 end, false)
10414
10415 self.create_parser("parse_inlines_no_inline_note",
10416 function()
10417 return parsers.inlines_no_inline_note
10418 end, false)
10419
10420 self.create_parser("parse_inlines_no_html",
10421 function()
10422 return parsers.inlines_no_html
10423 end, false)
10424
10425 self.create_parser("parse_inlines_nbsp",
10426 function()
10427 return parsers.inlines_nbsp
10428 end, false)
10429
10430 self.create_parser("parse_inlines_no_link_or_emphasis",
10431 function()
10432 return parsers.inlines_no_link_or_emphasis
10433 end, false)
10434
10435 self.create_parser("parse_inlines_identity",
10436 function()
10437 return parsers.inlines_identity
10438 end, false)
10439
10440 self.create_parser("parse_inlines_string",
10441 function()
10442 return parsers.inlines_string
10443 end, false)
10444
10445 self.create_parser("parse_inlines_math",
10446 function()
10447 return parsers.inlines_math
10448 end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

10449 parsers.minimally_indented_blankline
10450 = parsers.check_minimal_indent * (parsers.blankline / "")
10451
10452 parsers.minimally_indented_block
10453 = parsers.check_minimal_indent * V("Block")
10454
10455 parsers.minimally_indented_block_or_paragraph
10456 = parsers.check_minimal_indent * V("BlockOrParagraph")
10457
10458 parsers.minimally_indented_paragraph
10459 = parsers.check_minimal_indent * V("Paragraph")
10460
10461 parsers.minimally_indented_plain
10462 = parsers.check_minimal_indent * V("Plain")
10463
10464 parsers.minimally_indented_par_or_plain
10465 = parsers.minimally_indented_paragraph
10466 + parsers.minimally_indented_plain
10467
10468 parsers.minimally_indented_par_or_plain_no_blank
10469 = parsers.minimally_indented_par_or_plain
10470 - parsers.minimally_indented_blankline
10471
10472 parsers.minimally_indented_ref
10473 = parsers.check_minimal_indent * V("Reference")
10474
10475 parsers.minimally_indented_blank
10476 = parsers.check_minimal_indent * V("Blank")
10477
10478 parsers.conditionally_indented_blankline
10479 = parsers.check_minimal_blank_indent * (parsers.blankline / "")
10480
10481 parsers.minimally_indented_ref_or_block
10482 = parsers.minimally_indented_ref
10483 + parsers.minimally_indented_block
10484 - parsers.minimally_indented_blankline
10485
10486 parsers.minimally_indented_ref_or_block_or_par
10487 = parsers.minimally_indented_ref
10488 + parsers.minimally_indented_block_or_paragraph
10489 - parsers.minimally_indented_blankline
10490

```

The following pattern parses the properly indented content that follows the initial container start.

```

10491
10492 function parsers.separator_loop(separated_block, paragraph,

```

```

10493 block_separator, paragraph_separator)
10494 return separated_block
10495 + block_separator
10496 * paragraph
10497 * separated_block
10498 + paragraph_separator
10499 * paragraph
10500 end
10501
10502 function parsers.create_loop_body_pair(separated_block, paragraph,
10503 block_separator,
10504 paragraph_separator)
10505 return {
10506 block = parsers.separator_loop(separated_block, paragraph,
10507 block_separator, block_separator),
10508 par = parsers.separator_loop(separated_block, paragraph,
10509 block_separator, paragraph_separator)
10510 }
10511 end
10512
10513 parsers.block_sep_group = function(blank)
10514 return blank^0 * parsers.eof
10515 + (blank^2 / writer.paragraphsep
10516 + blank^0 / writer.interblocksep
10517)
10518 end
10519
10520 parsers.par_sep_group = function(blank)
10521 return blank^0 * parsers.eof
10522 + blank^0 / writer.paragraphsep
10523 end
10524
10525 parsers.sep_group_no_output = function(blank)
10526 return blank^0 * parsers.eof
10527 + blank^0
10528 end
10529
10530 parsers.content_blank = parsers.minimally_indented_blankline
10531
10532 parsers.ref_or_block_separated
10533 = parsers.sep_group_no_output(parsers.content_blank)
10534 * (parsers.minimally_indented_ref
10535 - parsers.content_blank)
10536 + parsers.block_sep_group(parsers.content_blank)
10537 * (parsers.minimally_indented_block
10538 - parsers.content_blank)
10539

```

```

10540 parsers.loop_body_pair =
10541 parsers.create_loop_body_pair(
10542 parsers.ref_or_block_separated,
10543 parsers.minimally_indented_par_or_plain_no_blank,
10544 parsers.block_sep_group(parsers.content_blank),
10545 parsers.par_sep_group(parsers.content_blank))
10546
10547 parsers.content_loop = (V("Block")
10548 * parsers.loop_body_pair.block^0
10549 + (V("Paragraph") + V("Plain"))
10550 * parsers.ref_or_block_separated
10551 * parsers.loop_body_pair.block^0
10552 + (V("Paragraph") + V("Plain"))
10553 * parsers.loop_body_pair.par^0
10554 * parsers.content_blank^0
10555
10556 parsers.indented_content = function()
10557 return Ct((V("Reference") + (parsers.blankline / ""))
10558 * parsers.content_blank^0
10559 * parsers.check_minimal_indent
10560 * parsers.content_loop
10561 + (V("Reference") + (parsers.blankline / ""))
10562 * parsers.content_blank^0
10563 + parsers.content_loop)
10564 end
10565
10566 parsers.add_indent = function(pattern, name, breakable)
10567 return Cg(Cmt(Cb("indent_info")
10568 * Ct(pattern)
10569 * (#parsers.linechar -- check if starter is blank
10570 * Cc(false) + Cc(true))
10571 * Cc(name)
10572 * Cc(breakable),
10573 process_starter_indent), "indent_info")
10574 end
10575

```

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

```

10576 if options.hashEnumerators then
10577 parsers.dig = parsers.digit + parsers.hash
10578 else
10579 parsers.dig = parsers.digit
10580 end
10581
10582 parsers.enumerator = function(delimiter_type, interrupting)
10583 local delimiter_range

```



```

10584 local allowed_end
10585 if interrupting then
10586 delimiter_range = P("1")
10587 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10588 else
10589 delimiter_range = parsers.dig * parsers.dig^-8
10590 allowed_end = C(parsers.spacechar^1)
10591 + #(parsers.newline + parsers.eof)
10592 end
10593
10594 return parsers.check_trail
10595 * Ct(C(delimiter_range) * C(delimiter_type))
10596 * allowed_end
10597 end
10598
10599 parsers.starter = parsers.bullet(parsers.dash)
10600 + parsers.bullet(parsers.asterisk)
10601 + parsers.bullet(parsers.plus)
10602 + parsers.enumerator(parsers.period)
10603 + parsers.enumerator(parsers.rparent)
10604

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

10605 parsers.blockquote_start
10606 = parsers.check_trail
10607 * C(parsers.more)
10608 * C(parsers.spacechar^0)
10609
10610 parsers.blockquote_body
10611 = parsers.add_indent(parsers.blockquote_start, "bq", true)
10612 * parsers.indented_content()
10613 * remove_indent("bq")
10614
10615 if not options.breakableBlockquotes then
10616 parsers.blockquote_body
10617 = parsers.add_indent(parsers.blockquote_start, "bq", false)
10618 * parsers.indented_content()
10619 * remove_indent("bq")
10620 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.

```

10621 local function parse_content_part(content_part)
10622 local rope = util.rope_to_string(content_part)
10623 local parsed

```

```

10624 = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
10625 parsed.indent_info = nil
10626 return parsed
10627 end
10628

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10629 local collect_emphasis_content =
10630 function(t, opening_index, closing_index)
10631 local content = {}
10632
10633 local content_part = {}
10634 for i = opening_index, closing_index do
10635 local value = t[i]
10636
10637 if value.rendered ~= nil then
10638 content[#content + 1] = parse_content_part(content_part)
10639 content_part = {}
10640 content[#content + 1] = value.rendered
10641 value.rendered = nil
10642 else
10643 if value.warning ~= nil then
10644 if next(content_part) ~= nil then
10645 content[#content + 1] = parse_content_part(content_part)
10646 end
10647 content_part = {}
10648
10649 content[#content + 1] = value.warning
10650 value.warning = nil
10651 end
10652 if value.type == "delimiter"
10653 and value.element == "emphasis" then
10654 if value.is_active then
10655 content_part[#content_part + 1]
10656 = string.rep(value.character, value.current_count)
10657 end
10658 else
10659 content_part[#content_part + 1] = value.content
10660 end
10661 value.content = ''
10662 value.is_active = false
10663 end
10664 end
10665
10666 if next(content_part) ~= nil then
10667 content[#content + 1] = parse_content_part(content_part)
10668 end
10669 end
10670

```

```

10668 end
10669
10670 return content
10671 end
10672

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

10673 local function fill_emph(t, opening_index, closing_index)
10674 local content
10675 = collect_emphasis_content(t, opening_index + 1,
10676 closing_index - 1)
10677 t[opening_index + 1].is_active = true
10678 t[opening_index + 1].rendered = writer.emphasis(content)
10679 end
10680

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

10681 local function fill_strong(t, opening_index, closing_index)
10682 local content
10683 = collect_emphasis_content(t, opening_index + 1,
10684 closing_index - 1)
10685 t[opening_index + 1].is_active = true
10686 t[opening_index + 1].rendered = writer.strong(content)
10687 end
10688

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

10689 local function breaks_three_rule(opening_delimiter, closing_delimiter)
10690 return (opening_delimiter.is_closing
10691 or closing_delimiter.is_opening)
10692 and ((opening_delimiter.original_count
10693 + closing_delimiter.original_count) % 3 == 0)
10694 and (opening_delimiter.original_count % 3 ~= 0
10695 or closing_delimiter.original_count % 3 ~= 0)
10696 end
10697

```

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`.

```

10698 local find_emphasis_opener = function(t, bottom_index, latest_index,
10699 character, closing_delimiter)
10700 for i = latest_index, bottom_index, -1 do
10701 local value = t[i]
10702 if value.is_active and

```

```

10703 value.is_opening and
10704 value.type == "delimiter" and
10705 value.element == "emphasis" and
10706 (value.character == character) and
10707 (value.current_count > 0) then
10708 if not breaks_three_rule(value, closing_delimiter) then
10709 return i
10710 end
10711 end
10712 end
10713 end
10714

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

10715 local function process_emphasis(t, opening_index, closing_index)
10716 for i = opening_index, closing_index do
10717 local value = t[i]
10718 if value.type == "delimiter" and value.element == "emphasis" then
10719 local delimiter_length = string.len(value.content)
10720 value.character = string.sub(value.content, 1, 1)
10721 value.current_count = delimiter_length
10722 value.original_count = delimiter_length
10723 end
10724 end
10725
10726 local openers_bottom = {
10727 ['*'] = {
10728 [true] = {opening_index, opening_index, opening_index},
10729 [false] = {opening_index, opening_index, opening_index}
10730 },
10731 ['_'] = {
10732 [true] = {opening_index, opening_index, opening_index},
10733 [false] = {opening_index, opening_index, opening_index}
10734 }
10735 }
10736
10737 local current_position = opening_index
10738 local max_position = closing_index
10739
10740 while current_position <= max_position do
10741 local value = t[current_position]
10742
10743 if value.type ~= "delimiter" or
10744 value.element ~= "emphasis" or
10745 not value.is_active or
10746 not value.is_closing or

```

```

10747 (value.current_count <= 0) then
10748 current_position = current_position + 1
10749 goto continue
10750 end
10751
10752 local character = value.character
10753 local is_opening = value.is_opening
10754 local closing_length_modulo_three = value.original_count % 3
10755
10756 local current_openers_bottom
10757 = openers_bottom[character][is_opening]
10758 [closing_length_modulo_three + 1]
10759
10760 local opener_position
10761 = find_emphasis_opener(t, current_openers_bottom,
10762 current_position - 1, character, value)
10763
10764 if (opener_position == nil) then
10765 openers_bottom[character][is_opening]
10766 [closing_length_modulo_three + 1]
10767 = current_position
10768 current_position = current_position + 1
10769 goto continue
10770 end
10771
10772 local opening_delimiter = t[opener_position]
10773
10774 local current_opening_count = opening_delimiter.current_count
10775 local current_closing_count = t[current_position].current_count
10776
10777 if (current_opening_count >= 2)
10778 and (current_closing_count >= 2) then
10779 opening_delimiter.current_count = current_opening_count - 2
10780 t[current_position].current_count = current_closing_count - 2
10781 fill_strong(t, opener_position, current_position)
10782 else
10783 opening_delimiter.current_count = current_opening_count - 1
10784 t[current_position].current_count = current_closing_count - 1
10785 fill_emph(t, opener_position, current_position)
10786 end
10787
10788 ::continue::
10789 end
10790 end
10791
10792 parsers.delimiter_run = function(character)
10793 return (B(parsers.backslash * character) + -B(character))

```

```

10794 * character^1
10795 * -#character
10796 end
10797
10798 parsers.left_flanking_delimiter_run = function(character)
10799 return (B(parsers.any)
10800 * (parsers.unicode.preceding_punctuation
10801 + parsers.unicode.preceding_whitespace)
10802 + -B(parsers.any))
10803 * parsers.delimiter_run(character)
10804 * #parsers.unicode.punctuation
10805 + parsers.delimiter_run(character)
10806 * -(#parsers.unicode.punctuation
10807 + #parsers.unicode.whitespace
10808 + parsers.eof)
10809 end
10810
10811 parsers.right_flanking_delimiter_run = function(character)
10812 return parsers.unicode.preceding_punctuation
10813 * parsers.delimiter_run(character)
10814 * (#parsers.unicode.punctuation
10815 + #parsers.unicode.whitespace
10816 + parsers.eof)
10817 + (B(parsers.any)
10818 * -(parsers.unicode.preceding_punctuation
10819 + parsers.unicode.preceding_whitespace))
10820 * parsers.delimiter_run(character)
10821 end
10822
10823 if options.underscores then
10824 parsers.emph_start
10825 = parsers.left_flanking_delimiter_run(parsers.asterisk)
10826 + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
10827 + (parsers.unicode.preceding_punctuation
10828 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
10829 * parsers.left_flanking_delimiter_run(parsers.underscore)
10830
10831 parsers.emph_end
10832 = parsers.right_flanking_delimiter_run(parsers.asterisk)
10833 + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
10834 + #(parsers.left_flanking_delimiter_run(parsers.underscore)
10835 * #parsers.unicode.punctuation))
10836 * parsers.right_flanking_delimiter_run(parsers.underscore)
10837 else
10838 parsers.emph_start
10839 = parsers.left_flanking_delimiter_run(parsers.asterisk)
10840

```

```

10841 parsers.emph_end
10842 = parsers.right_flanking_delimiter_run(parsers.asterisk)
10843 end
10844
10845 parsers.emph_capturing_open_and_close
10846 = #parsers.emph_start * #parsers.emph_end
10847 * Ct(Cg(Cc("delimiter"), "type")
10848 * Cg(Cc("emphasis"), "element")
10849 * Cg(C(parsers.emph_start), "content")
10850 * Cg(Cc(true), "is_opening")
10851 * Cg(Cc(true), "is_closing"))
10852
10853 parsers.emph_capturing_open = Ct(Cg(Cc("delimiter"), "type")
10854 * Cg(Cc("emphasis"), "element")
10855 * Cg(C(parsers.emph_start), "content")
10856 * Cg(Cc(true), "is_opening")
10857 * Cg(Cc(false), "is_closing"))
10858
10859 parsers.emph_capturing_close = Ct(Cg(Cc("delimiter"), "type")
10860 * Cg(Cc("emphasis"), "element")
10861 * Cg(C(parsers.emph_end), "content")
10862 * Cg(Cc(false), "is_opening")
10863 * Cg(Cc(true), "is_closing"))
10864
10865 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
10866 + parsers.emph_capturing_open
10867 + parsers.emph_capturing_close
10868
10869 parsers.emph_open = parsers.emph_capturing_open_and_close
10870 + parsers.emph_capturing_open
10871
10872 parsers.emph_close = parsers.emph_capturing_open_and_close
10873 + parsers.emph_capturing_close
10874

```

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

```

10875 -- List of references defined in the document
10876 local references
10877
10878 -- List of note references defined in the document
10879 parsers.rawnotes = {}
10880

```

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.

```

10881 function self.register_link(_, tag, url, title,

```

```

10882 attributes)
10883 local normalized_tag = self.normalize_tag(tag)
10884 if references[normalized_tag] == nil then
10885 references[normalized_tag] = {
10886 url = url,
10887 title = title,
10888 attributes = attributes
10889 }
10890 return ""
10891 else
10892 local text
10893 = string.format('Multiply defined link reference "%s"', tag)
10894 local more = string.format("Look for the text `[%s]: ...`.", tag)
10895 return writer.warning(text, more)
10896 end
10897 end
10898

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

10899 function self.lookup_reference(tag)
10900 return references[self.normalize_tag(tag)]
10901 end
10902

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

10903 function self.lookup_note_reference(tag)
10904 return parsers.rawnotes[self.normalize_tag(tag)]
10905 end
10906
10907 parsers.title_s_direct_ref = parsers.squote
10908 * Cs((parsers.html_entities
10909 + (parsers.anyescaped
10910 - parsers.squote
10911 - parsers.blankline^2))^0)
10912 * parsers.squote
10913
10914 parsers.title_d_direct_ref = parsers.dquote
10915 * Cs((parsers.html_entities
10916 + (parsers.anyescaped
10917 - parsers.dquote
10918 - parsers.blankline^2))^0)
10919 * parsers.dquote
10920
10921 parsers.title_p_direct_ref = parsers.lparent
10922 * Cs((parsers.html_entities
10923 + (parsers.anyescaped
10924 - parsers.lparent

```



```

10925 - parsers.rparent
10926 - parsers.blankline^2))^0)
10927 * parsers.rparent
10928
10929 parsers.title_direct_ref = parsers.title_s_direct_ref
10930 + parsers.title_d_direct_ref
10931 + parsers.title_p_direct_ref
10932
10933 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
10934 * Cg(parsers.url + Cc(""), "url")
10935 * parsers.spnl
10936 * Cg(parsers.title_direct_ref
10937 + Cc(""), "title")
10938 * parsers.spnl * parsers.rparent
10939
10940 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
10941 * Cg(parsers.url + Cc(""), "url")
10942 * parsers.spnlc
10943 * Cg(parsers.title + Cc(""), "title")
10944 * parsers.spnlc * parsers.rparent
10945
10946 parsers.empty_link = parsers.lbracket
10947 * parsers.rbracket
10948
10949 parsers.inline_link = parsers.link_text
10950 * parsers.inline_direct_ref
10951
10952 parsers.full_link = parsers.link_text
10953 * parsers.link_label
10954
10955 parsers.shortcut_link = parsers.link_label
10956 * -(parsers.empty_link + parsers.link_label)
10957
10958 parsers.collapsed_link = parsers.link_label
10959 * parsers.empty_link
10960
10961 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
10962 * Cg(Cc("inline"), "link_type")
10963 + #(parsers.exclamation * parsers.full_link)
10964 * Cg(Cc("full"), "link_type")
10965 + #(parsers.exclamation
10966 * parsers.collapsed_link)
10967 * Cg(Cc("collapsed"), "link_type")
10968 + #(parsers.exclamation * parsers.shortcut_link)
10969 * Cg(Cc("shortcut"), "link_type")
10970 + #(parsers.exclamation * parsers.empty_link)
10971 * Cg(Cc("empty"), "link_type")

```

```

10972
10973 parsers.link_opening = #parsers.inline_link
10974 * Cg(Cc("inline"), "link_type")
10975 + #parsers.full_link
10976 * Cg(Cc("full"), "link_type")
10977 + #parsers.collapsed_link
10978 * Cg(Cc("collapsed"), "link_type")
10979 + #parsers.shortcut_link
10980 * Cg(Cc("shortcut"), "link_type")
10981 + #parsers.empty_link
10982 * Cg(Cc("empty_link"), "link_type")
10983 + #parsers.link_text
10984 * Cg(Cc("link_text"), "link_type")
10985
10986 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
10987 * Cg(Cc("note_inline"), "link_type")
10988
10989 parsers.raw_note_opening = #(parsers.lbracket
10990 * parsers.circumflex
10991 * parsers.link_label_body
10992 * parsers.rbracket)
10993 * Cg(Cc("raw_note"), "link_type")
10994
10995 local inline_note_element = Cg(Cc("note"), "element")
10996 * parsers.note_opening
10997 * Cg(parsers.circumflex
10998 * parsers.lbracket, "content")
10999
11000 local image_element = Cg(Cc("image"), "element")
11001 * parsers.image_opening
11002 * Cg(parsers.exclamation
11003 * parsers.lbracket, "content")
11004
11005 local note_element = Cg(Cc("note"), "element")
11006 * parsers.raw_note_opening
11007 * Cg(parsers.lbracket
11008 * parsers.circumflex, "content")
11009
11010 local link_element = Cg(Cc("link"), "element")
11011 * parsers.link_opening
11012 * Cg(parsers.lbracket, "content")
11013
11014 local opening_elements = parsers.fail
11015
11016 if options.inlineNotes then
11017 opening_elements = opening_elements + inline_note_element
11018 end

```

```

11019
11020 opening_elements = opening_elements + image_element
11021
11022 if options.notes then
11023 opening_elements = opening_elements + note_element
11024 end
11025
11026 opening_elements = opening_elements + link_element
11027
11028 parsers.link_image_opening = Ct(Cg(Cc("delimiter"), "type")
11029 * Cg(Cc(true), "is_opening")
11030 * Cg(Cc(false), "is_closing")
11031 * opening_elements)
11032
11033 parsers.link_image_closing = Ct(Cg(Cc("delimiter"), "type")
11034 * Cg(Cc("link"), "element")
11035 * Cg(Cc(false), "is_opening")
11036 * Cg(Cc(true), "is_closing")
11037 * (Cg(Cc(true), "is_direct")
11038 * Cg(parsers.rbracket
11039 * #parsers.inline_direct_ref,
11040 "content")
11041 + Cg(Cc(false), "is_direct")
11042 * Cg(parsers.rbracket, "content")))
11043
11044 parsers.link_image_open_or_close = parsers.link_image_opening
11045 + parsers.link_image_closing
11046
11047 if options.html then
11048 parsers.link_emph_precedence = parsers.inticks
11049 + parsers.autolink
11050 + parsers.html_inline_tags
11051 else
11052 parsers.link_emph_precedence = parsers.inticks
11053 + parsers.autolink
11054 end
11055
11056 parsers.link_and_emph_endline = parsers.newline
11057 * ((parsers.check_minimal_indent
11058 * -V("EndlineExceptions")
11059 + parsers.check_optional_indent
11060 * -V("EndlineExceptions")
11061 * -V("ListStarter")) / "")
11062 * parsers.spacechar^0 / "\n"
11063
11064 parsers.link_and_emph_content
11065 = Ct(Cg(Cc("content"), "type")

```

```

11066 * Cg(Cs((parsers.link_emph_precedence
11067 + parsers.backslash * parsers.linechar
11068 + parsers.link_and_emph_endline
11069 + (parsers.linechar
11070 - parsers.blankline^2
11071 - parsers.link_image_open_or_close
11072 - parsers.emph_open_or_close))^0), "content"))
11073
11074 parsers.link_and_emph_table
11075 = (parsers.link_image_opening + parsers.emph_open)
11076 * parsers.link_and_emph_content
11077 * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
11078 * parsers.link_and_emph_content)^1
11079

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

11080 local function collect_link_content(t, opening_index, closing_index)
11081 local content = {}
11082 for i = opening_index, closing_index do
11083 content[#content + 1] = t[i].content
11084 end
11085 return util.rope_to_string(content)
11086 end
11087

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

11088 local function find_link_opener(t, bottom_index, latest_index)
11089 for i = latest_index, bottom_index, -1 do
11090 local value = t[i]
11091 if value.type == "delimiter" and
11092 value.is_opening and
11093 (value.element == "link"
11094 or value.element == "image"
11095 or value.element == "note")
11096 and not value.removed then
11097 if value.is_active then
11098 return i
11099 end
11100 value.removed = true
11101 end
11102 end
11103 end
11104 end
11105

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

11106 local function find_next_link_closing_index(t, latest_index)
11107 for i = latest_index, #t do
11108 local value = t[i]
11109 if value.is_closing and
11110 value.element == "link" and
11111 not value.removed then
11112 return i
11113 end
11114 end
11115 end
11116

```

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.

```

11117 local function disable_previous_link_openers(t, opening_index)
11118 if t[opening_index].element == "image" then
11119 return
11120 end
11121
11122 for i = opening_index, 1, -1 do
11123 local value = t[i]
11124 if value.is_active and
11125 value.type == "delimiter" and
11126 value.is_opening and
11127 value.element == "link" then
11128 value.is_active = false
11129 end
11130 end
11131 end
11132

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

11133 local function disable_range(t, opening_index, closing_index)
11134 for i = opening_index, closing_index do
11135 local value = t[i]
11136 if value.is_active then
11137 value.is_active = false
11138 if value.type == "delimiter" then
11139 value.removed = true
11140 end
11141 end
11142 end
11143 end
11144

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
11145 local delete_parsed_content_in_range =
11146 function(t, opening_index, closing_index)
11147 for i = opening_index, closing_index do
11148 t[i].rendered = nil
11149 end
11150 end
11151
```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
11152 local function empty_content_in_range(t, opening_index, closing_index)
11153 for i = opening_index, closing_index do
11154 t[i].content = ''
11155 end
11156 end
11157
```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```
11158 local function join_attributes(reference_attributes, own_attributes)
11159 local merged_attributes = {}
11160 for _, attribute in ipairs(reference_attributes or {}) do
11161 table.insert(merged_attributes, attribute)
11162 end
11163 for _, attribute in ipairs(own_attributes or {}) do
11164 table.insert(merged_attributes, attribute)
11165 end
11166 if next(merged_attributes) == nil then
11167 merged_attributes = nil
11168 end
11169 return merged_attributes
11170 end
11171
```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```
11172 local render_link_or_image =
11173 function(t, opening_index, closing_index, content_end_index,
11174 reference)
11175 process_emphasis(t, opening_index, content_end_index)
11176 local mapped = collect_emphasis_content(t, opening_index + 1,
11177 content_end_index - 1)
11178
11179 local rendered = {}
11180 if (t[opening_index].element == "link") then
11181 rendered = writer.link(mapped, reference.url,
```

```

11182 reference.title, reference.attributes)
11183 end
11184
11185 if (t[opening_index].element == "image") then
11186 rendered = writer.image(mapped, reference.url,
11187 self.parser_functions.parse_inlines_string(reference.title),
11188 reference.attributes)
11189 end
11190
11191 if (t[opening_index].element == "note") then
11192 if (t[opening_index].link_type == "note_inline") then
11193 rendered = writer.note(mapped)
11194 end
11195 if (t[opening_index].link_type == "raw_note") then
11196 rendered = writer.note(reference)
11197 end
11198 end
11199
11200 t[opening_index].rendered = rendered
11201 delete_parsed_content_in_range(t, opening_index + 1,
11202 closing_index)
11203 empty_content_in_range(t, opening_index, closing_index)
11204 disable_previous_link_openers(t, opening_index)
11205 disable_range(t, opening_index, closing_index)
11206 end
11207

```

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.

```

11208 local resolve_inline_following_content =
11209 function(t, closing_index, match_reference, match_link_attributes)
11210 local content = ""
11211 for i = closing_index + 1, #t do
11212 content = content .. t[i].content
11213 end
11214
11215 local matching_content = parsers.succeed
11216
11217 if match_reference then
11218 matching_content = matching_content
11219 * parsers.inline_direct_ref_inside
11220 end
11221
11222 if match_link_attributes then
11223 matching_content = matching_content
11224 * Cg(Ct(parsers.attributes^-1), "attributes")

```

```

11225 end
11226
11227 local matched = lpeg.match(Ct(matching_content
11228 * Cg(Cp(), "end_position")), content)
11229
11230 local matched_count = matched.end_position - 1
11231 for i = closing_index + 1, #t do
11232 local value = t[i]
11233
11234 local chars_left = matched_count
11235 matched_count = matched_count - #value.content
11236
11237 if matched_count <= 0 then
11238 value.content = value.content:sub(chars_left + 1)
11239 break
11240 end
11241
11242 value.content = ''
11243 value.is_active = false
11244 end
11245
11246 local attributes = matched.attributes
11247 if attributes == nil or next(attributes) == nil then
11248 attributes = nil
11249 end
11250
11251 return {
11252 url = matched.url or "",
11253 title = matched.title or "",
11254 attributes = attributes
11255 }
11256 end
11257

```

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.

```

11258 local function resolve_inline_link(t, opening_index, closing_index)
11259 local inline_content
11260 = resolve_inline_following_content(t, closing_index, true,
11261 t.match_link_attributes)
11262 render_link_or_image(t, opening_index, closing_index,
11263 closing_index, inline_content)
11264 end
11265

```



Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
11266 local resolve_note_inline_link =
11267 function(t, opening_index, closing_index)
11268 local inline_content
11269 = resolve_inline_following_content(t, closing_index,
11270 false, false)
11271 render_link_or_image(t, opening_index, closing_index,
11272 closing_index, inline_content)
11273 end
11274
```

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.

```
11275 local function resolve_shortcut_link(t, opening_index, closing_index)
11276 local content
11277 = collect_link_content(t, opening_index + 1, closing_index - 1)
11278 local r = self.lookup_reference(content)
11279
11280 if r then
11281 local inline_content
11282 = resolve_inline_following_content(t, closing_index, false,
11283 t.match_link_attributes)
11284 r.attributes
11285 = join_attributes(r.attributes, inline_content.attributes)
11286 render_link_or_image(t, opening_index, closing_index,
11287 closing_index, r)
11288 end
11289 end
11290
```

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.

```
11291 local function resolve_raw_note_link(t, opening_index, closing_index)
11292 local content
11293 = collect_link_content(t, opening_index + 1, closing_index - 1)
11294 local r = self.lookup_note_reference(content)
11295
11296 if r then
11297 local parsed_ref = self.parser_functions.parse_blocks_nested(r)
11298 render_link_or_image(t, opening_index, closing_index,
11299 closing_index, parsed_ref)
11300 end
11301 end
11302
```

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.

```

11303 local function resolve_full_link(t, opening_index, closing_index)
11304 local next_link_closing_index
11305 = find_next_link_closing_index(t, closing_index + 4)
11306 local next_link_content
11307 = collect_link_content(t, closing_index + 3,
11308 next_link_closing_index - 1)
11309 local r = self.lookup_reference(next_link_content)
11310
11311 if r then
11312 local inline_content
11313 = resolve_inline_following_content(t, next_link_closing_index,
11314 false,
11315 t.match_link_attributes)
11316 r.attributes
11317 = join_attributes(r.attributes, inline_content.attributes)
11318 render_link_or_image(t, opening_index, next_link_closing_index,
11319 closing_index, r)
11320 else
11321 local text = string.format('Undefined link reference "%s"',
11322 next_link_content)
11323 local more = string.format("Look for the text `[...] [%s]`.",
11324 next_link_content)
11325 t[opening_index].warning = writer.warning(text, more)
11326 end
11327 end
11328
```

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.

```

11329 local function resolve_collapsed_link(t, opening_index, closing_index)
11330 local next_link_closing_index
11331 = find_next_link_closing_index(t, closing_index + 4)
11332 local content
11333 = collect_link_content(t, opening_index + 1, closing_index - 1)
11334 local r = self.lookup_reference(content)
11335
11336 if r then
11337 local inline_content
11338 = resolve_inline_following_content(t, closing_index, false,
11339 t.match_link_attributes)
11340 r.attributes
11341 = join_attributes(r.attributes, inline_content.attributes)
11342 render_link_or_image(t, opening_index, next_link_closing_index,
11343 closing_index, r)
11344 end
11345
```

```

11344 else
11345 local text = string.format('Undefined link reference "%s"',
11346 content)
11347 local more = string.format("Look for the text `[%s] []`.",
11348 content)
11349 t[opening_index].warning = writer.warning(text, more)
11350 end
11351 end
11352

```

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.

```

11353 local function process_links_and_emphasis(t)
11354 for _,value in ipairs(t) do
11355 value.is_active = true
11356 end
11357
11358 for i,value in ipairs(t) do
11359 if not value.is_closing
11360 or value.type ~= "delimiter"
11361 or not (value.element == "link"
11362 or value.element == "image"
11363 or value.element == "note")
11364 or value.removed then
11365 goto continue
11366 end
11367
11368 local opener_position = find_link_opener(t, 1, i - 1)
11369 if (opener_position == nil) then
11370 goto continue
11371 end
11372
11373 local opening_delimiter = t[opener_position]
11374 opening_delimiter.removed = true
11375
11376 local link_type = opening_delimiter.link_type
11377
11378 if (link_type == "inline") then
11379 resolve_inline_link(t, opener_position, i)
11380 end
11381 if (link_type == "shortcut") then
11382 resolve_shortcut_link(t, opener_position, i)
11383 end
11384 if (link_type == "full") then
11385 resolve_full_link(t, opener_position, i)

```

```

11386 end
11387 if (link_type == "collapsed") then
11388 resolve_collapsed_link(t, opener_position, i)
11389 end
11390 if (link_type == "note_inline") then
11391 resolve_note_inline_link(t, opener_position, i)
11392 end
11393 if (link_type == "raw_note") then
11394 resolve_raw_note_link(t, opener_position, i)
11395 end
11396
11397 ::continue::
11398 end
11399
11400 t[#t].content = t[#t].content:gsub("%s*$", "")
11401
11402 process_emphasis(t, 1, #t)
11403 local final_result = collect_emphasis_content(t, 1, #t)
11404 return final_result
11405 end
11406
11407 function self.defer_link_and_emphasis_processing(delimiter_table)
11408 return writer.defer_call(function()
11409 return process_links_and_emphasis(delimiter_table)
11410 end)
11411 end
11412

```

### 3.1.6.8 Inline Elements (local)

```

11413 parsers.Str = (parsers.normalchar
11414 * (parsers.normalchar + parsers.at)^0)
11415 / writer.string
11416
11417 parsers.Symbol = (parsers.backtick^1 + V("SpecialChar"))
11418 / writer.string
11419
11420 parsers.Ellipsis = P("...") / writer.ellipsis
11421
11422 parsers.Smart = parsers.Ellipsis
11423
11424 parsers.Code = parsers.inticks / writer.code
11425
11426 if options.blankBeforeBlockquote then
11427 parsers.bqstart = parsers.fail
11428 else
11429 parsers.bqstart = parsers.blockquote_start

```

```

11430 end
11431
11432 if options.blankBeforeHeading then
11433 parsers.headerstart = parsers.fail
11434 else
11435 parsers.headerstart = parsers.atx_heading
11436 end
11437
11438 if options.blankBeforeList then
11439 parsers.interrupting_bullets = parsers.fail
11440 parsers.interrupting_enumerators = parsers.fail
11441 else
11442 parsers.interrupting_bullets
11443 = parsers.bullet(parsers.dash, true)
11444 + parsers.bullet(parsers.asterisk, true)
11445 + parsers.bullet(parsers.plus, true)
11446
11447 parsers.interrupting_enumerators
11448 = parsers.enumerator(parsers.period, true)
11449 + parsers.enumerator(parsers.rparent, true)
11450 end
11451
11452 if options.html then
11453 parsers.html_interrupting
11454 = parsers.check_trail
11455 * (parsers.html_incomplete_open_tag
11456 + parsers.html_incomplete_close_tag
11457 + parsers.html_incomplete_open_special_tag
11458 + parsers.html_comment_start
11459 + parsers.html_cdatasection_start
11460 + parsers.html_declaration_start
11461 + parsers.html_instruction_start
11462 - parsers.html_close_special_tag
11463 - parsers.html_empty_special_tag)
11464 else
11465 parsers.html_interrupting = parsers.fail
11466 end
11467
11468 if options.blankBeforeHtmlBlock then
11469 parsers.html_interrupting = parsers.fail
11470 end
11471
11472 parsers.ListStarter = parsers.starter
11473
11474 parsers.EndlineExceptions
11475 = parsers.blankline -- paragraph break
11476 + parsers.eof -- end of document

```

```

11477 + parsers.bqstart
11478 + parsers.thematic_break_lines
11479 + parsers.interrupting_bullets
11480 + parsers.interrupting_enumerators
11481 + parsers.headerstart
11482 + parsers.html_interrupting
11483
11484 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
11485
11486 parsers.endline = parsers.newline
11487 * (parsers.check_minimal_indent
11488 * -V("EndlineExceptions")
11489 + parsers.check_optional_indent
11490 * -V("EndlineExceptions")
11491 * -V("ListStarter")) / function(_) return end
11492 * parsers.spacechar^0
11493
11494 parsers.Endline = parsers.endline
11495 / writer.soft_line_break
11496
11497 parsers.EndlineNoSub = parsers.endline
11498
11499 parsers.NoSoftLineBreakEndline
11500 = parsers.newline
11501 * (parsers.check_minimal_indent
11502 * -V("NoSoftLineBreakEndlineExceptions")
11503 + parsers.check_optional_indent
11504 * -V("NoSoftLineBreakEndlineExceptions")
11505 * -V("ListStarter"))
11506 * parsers.spacechar^0
11507 / writer.space
11508
11509 parsers.EndlineBreak = parsers.backslash * parsers.endline
11510 / writer.hard_line_break
11511
11512 parsers.OptionalIndent
11513 = parsers.spacechar^1 / writer.space
11514
11515 parsers.Space = parsers.spacechar^2 * parsers.endline
11516 / writer.hard_line_break
11517 + parsers.spacechar^1
11518 * parsers.endline^-1
11519 * parsers.eof / self.expandtabs
11520 + parsers.spacechar^1 * parsers.endline
11521 / writer.soft_line_break
11522 + parsers.spacechar^1
11523 * -parsers.newline / self.expandtabs

```

```

11524 + parsers.spacechar^1
11525 + (parsers.spacechar
11526 + parsers.html_space_entity_char)^1
11527 * parsers.endline / writer.soft_line_break
11528
11529 parsers.NonbreakingEndline
11530 = parsers.endline
11531 / writer.nbsp
11532
11533 parsers.NonbreakingSpace
11534 = parsers.spacechar^2 * parsers.endline
11535 / writer.nbsp
11536 + parsers.spacechar^1
11537 * parsers.endline^-1 * parsers.eof / ""
11538 + parsers.spacechar^1 * parsers.endline
11539 * parsers.optionalspace
11540 / writer.nbsp
11541 + parsers.spacechar^1 * parsers.optionalspace
11542 / writer.nbsp
11543

```

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.

```

11544 function self.auto_link_url(url, attributes)
11545 return writer.link(writer.escape(url),
11546 url, nil, attributes)
11547 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.

```

11548 function self.auto_link_email(email, attributes)
11549 return writer.link(writer.escape(email),
11550 "mailto:".email,
11551 nil, attributes)
11552 end
11553
11554 parsers.AutoLinkUrl = parsers.auto_link_url
11555 / self.auto_link_url
11556
11557 parsers.AutoLinkEmail
11558 = parsers.auto_link_email
11559 / self.auto_link_email
11560
11561 parsers.AutoLinkRelativeReference
11562 = parsers.auto_link_relative_reference

```

```

11563 / self.auto_link_url
11564
11565 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
11566 / self.defer_link_and_emphasis_processing
11567
11568 parsers.EscapedChar = parsers.backslash
11569 * C(parsers.escapable) / writer.string
11570
11571 if options.htmlOutput == "commonmark" then
11572 parsers.InlineHtml
11573 = Cs(parsers.html_inline_comment)
11574 / self.parser_functions.parse_inlines_no_html
11575 / writer.inline_html_comment
11576 + Cs(parsers.html_any_empty_inline_tag)
11577 / writer.inline_html_empty_tag
11578 + Cs(parsers.html_inline_instruction)
11579 / writer.inline_html_processing_instruction
11580 + Cs(parsers.html_inline_cdatasection)
11581 / writer.inline_html_cdata_section
11582 + Cs(parsers.html_inline_declaration)
11583 / writer.inline_html_declaration
11584 + Cs(parsers.html_any_open_inline_tag)
11585 / writer.inline_html_open_tag
11586 + Cs(parsers.html_any_close_tag)
11587 / writer.inline_html_close_tag
11588 else
11589 parsers.InlineHtml = Cs(parsers.html_inline_comment)
11590 / self.parser_functions.parse_inlines_no_html
11591 / writer.inline_html_comment
11592 + Cs(parsers.html_any_empty_inline_tag
11593 + parsers.html_inline_instruction
11594 + parsers.html_inline_cdatasection
11595 + parsers.html_inline_declaration
11596 + parsers.html_any_open_inline_tag
11597 + parsers.html_any_close_tag)
11598 / writer.inline_html_tag
11599 end
11600
11601 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

11602 if options.htmlOutput == "commonmark" then
11603 parsers.DisplayHtml
11604 = parsers.check_trail
11605 * (parsers.html_comment
11606 / self.parser_functions.parse_blocks_nested

```



```

11607 / writer.block_html_comment)
11608 + Cs(parsers.check_trail * parsers.html_special_block)
11609 / writer.block_html_cdata_element
11610 + Cs(parsers.check_trail * parsers.html_block)
11611 / writer.block_html_pdata_element
11612 + Cs(parsers.check_trail * parsers.html_any_block)
11613 / writer.block_html_standalone_tag
11614 + Cs(parsers.check_trail * parsers.html_instruction)
11615 / writer.block_html_processing_instruction
11616 + Cs(parsers.check_trail * parsers.html_cdatasection)
11617 / writer.block_html_cdata_section
11618 + Cs(parsers.check_trail * parsers.html_declaration)
11619 / writer.block_html_declaration
11620 else
11621 parsers.DisplayHtml = Cs(parsers.check_trail
11622 * (parsers.html_comment
11623 + parsers.html_special_block
11624 + parsers.html_block
11625 + parsers.html_any_block
11626 + parsers.html_instruction
11627 + parsers.html_cdatasection
11628 + parsers.html_declaration))
11629 / writer.block_html_element
11630 end
11631
11632 parsers.indented_non_blank_line = parsers.indentedline
11633 - parsers.blankline
11634
11635 parsers.Verbatim
11636 = Cs(parsers.check_code_trail
11637 * (parsers.line - parsers.blankline)
11638 * ((parsers.check_minimal_blank_indent_and_full_code_trail
11639 * parsers.blankline)^0
11640 * ((parsers.check_minimal_indent / "")
11641 * parsers.check_code_trail
11642 * (parsers.line - parsers.blankline))^1)^0)
11643 / self.expandtabs / writer.verbatim
11644
11645 parsers.Blockquote = parsers.blockquote_body
11646 / writer.blockquote
11647
11648 parsers.ThematicBreak = parsers.thematic_break_lines
11649 / writer.thematic_break
11650
11651 parsers.Reference = parsers.define_reference_parser
11652 / self.register_link
11653

```

```

11654 parsers.Paragraph = parsers.freeze_trail
11655 * (Ct((parsers.Inline)^1)
11656 * (parsers.newline + parsers.eof)
11657 * parsers.unfreeze_trail
11658 / writer.paragraph)
11659
11660 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
11661 / writer.plain

```

### 3.1.6.10 Lists (local)

```

11662
11663 if options.taskLists then
11664 parsers.tickbox = (parsers.ticked_box
11665 + parsers.halfticked_box
11666 + parsers.unticked_box
11667) / writer.tickbox
11668 else
11669 parsers.tickbox = parsers.fail
11670 end
11671
11672 parsers.list_blank = parsers.conditionally_indented_blankline
11673
11674 parsers.ref_or_block_list_separated
11675 = parsers.sep_group_no_output(parsers.list_blank)
11676 * parsers.minimally_indented_ref
11677 + parsers.block_sep_group(parsers.list_blank)
11678 * parsers.minimally_indented_block
11679
11680 parsers.ref_or_block_non_separated
11681 = parsers.minimally_indented_ref
11682 + (parsers.succeed / writer.interblocksep)
11683 * parsers.minimally_indented_block
11684 - parsers.minimally_indented_blankline
11685
11686 parsers.tight_list_loop_body_pair =
11687 parsers.create_loop_body_pair(
11688 parsers.ref_or_block_non_separated,
11689 parsers.minimally_indented_par_or_plain_no_blank,
11690 (parsers.succeed / writer.interblocksep),
11691 (parsers.succeed / writer.paragraphsep))
11692
11693 parsers.loose_list_loop_body_pair =
11694 parsers.create_loop_body_pair(
11695 parsers.ref_or_block_list_separated,
11696 parsers.minimally_indented_par_or_plain,
11697 parsers.block_sep_group(parsers.list_blank),

```

```

11698 parsers.par_sep_group(parsers.list_blank))
11699
11700 parsers.tight_list_content_loop
11701 = V("Block")
11702 * parsers.tight_list_loop_body_pair.block^0
11703 + (V("Paragraph") + V("Plain"))
11704 * parsers.ref_or_block_non_separated
11705 * parsers.tight_list_loop_body_pair.block^0
11706 + (V("Paragraph") + V("Plain"))
11707 * parsers.tight_list_loop_body_pair.par^0
11708
11709 parsers.loose_list_content_loop
11710 = V("Block")
11711 * parsers.loose_list_loop_body_pair.block^0
11712 + (V("Paragraph") + V("Plain"))
11713 * parsers.ref_or_block_list_separated
11714 * parsers.loose_list_loop_body_pair.block^0
11715 + (V("Paragraph") + V("Plain"))
11716 * parsers.loose_list_loop_body_pair.par^0
11717
11718 parsers.list_item_tightness_condition
11719 = -#(parsers.list_blank^0
11720 * parsers.minimally_indented_ref_or_block_or_par)
11721 * remove_indent("li")
11722 + remove_indent("li")
11723 * parsers.fail
11724
11725 parsers.indented_content_tight
11726 = Ct((parsers.blankline / "")
11727 * #parsers.list_blank
11728 * remove_indent("li")
11729 + ((V("Reference") + (parsers.blankline / ""))
11730 * parsers.check_minimal_indent
11731 * parsers.tight_list_content_loop
11732 + (V("Reference") + (parsers.blankline / ""))
11733 + (parsers.tickbox~-1 / writer.escape)
11734 * parsers.tight_list_content_loop
11735)
11736 * parsers.list_item_tightness_condition)
11737
11738 parsers.indented_content_loose
11739 = Ct((parsers.blankline / "")
11740 * #parsers.list_blank
11741 + ((V("Reference") + (parsers.blankline / ""))
11742 * parsers.check_minimal_indent
11743 * parsers.loose_list_content_loop
11744 + (V("Reference") + (parsers.blankline / ""))

```

```

11745 + (parsers.tickbox~-1 / writer.escape)
11746 * parsers.loose_list_content_loop))
11747
11748 parsers.TightListItem = function(starter)
11749 return -parsers.ThematicBreak
11750 * parsers.add_indent(starter, "li")
11751 * parsers.indented_content_tight
11752 end
11753
11754 parsers.LooseListItem = function(starter)
11755 return -parsers.ThematicBreak
11756 * parsers.add_indent(starter, "li")
11757 * parsers.indented_content_loose
11758 * remove_indent("li")
11759 end
11760
11761 parsers.BulletListOfType = function(bullet_type)
11762 local bullet = parsers.bullet(bullet_type)
11763 return (Ct(parsers.TightListItem(bullet)
11764 * ((parsers.check_minimal_indent / "")
11765 * parsers.TightListItem(bullet)
11766)^0
11767)
11768 * Cc(true)
11769 * -#((parsers.list_blank^0 / "")
11770 * parsers.check_minimal_indent
11771 * (bullet - parsers.ThematicBreak)
11772)
11773 + Ct(parsers.LooseListItem(bullet)
11774 * ((parsers.list_blank^0 / "")
11775 * (parsers.check_minimal_indent / "")
11776 * parsers.LooseListItem(bullet)
11777)^0
11778)
11779 * Cc(false)
11780) / writer.bulletlist
11781 end
11782
11783 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
11784 + parsers.BulletListOfType(parsers.asterisk)
11785 + parsers.BulletListOfType(parsers.plus)
11786
11787 local function ordered_list(items,tight,starter)
11788 local startnum = starter[2][1]
11789 if options.startNumber then
11790 startnum = tonumber(startnum) or 1 -- fallback for '#'
11791 if startnum ~= nil then

```

```

11792 startnum = math.floor(startnum)
11793 end
11794 else
11795 startnum = nil
11796 end
11797 return writer.orderedlist(items,tight,startnum)
11798 end
11799
11800 parsers.OrderedListOfType = function(delimiter_type)
11801 local enumerator = parsers.enumerator(delimiter_type)
11802 return Cg(enumerator, "listtype")
11803 * (Ct(parsers.TightListItem(Cb("listtype"))
11804 * ((parsers.check_minimal_indent / "")
11805 * parsers.TightListItem(enumerator))^0)
11806 * Cc(true)
11807 * -#((parsers.list_blank^0 / "")
11808 * parsers.check_minimal_indent * enumerator)
11809 + Ct(parsers.LooseListItem(Cb("listtype"))
11810 * ((parsers.list_blank^0 / "")
11811 * (parsers.check_minimal_indent / "")
11812 * parsers.LooseListItem(enumerator))^0)
11813 * Cc(false)
11814) * Ct(Cb("listtype")) / ordered_list
11815 end
11816
11817 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
11818 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

11819 parsers.Blank = parsers.blankline / ""
11820 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

11821 function parsers.parse_heading_text(s)
11822 local inlines = self.parser_functions.parse_inlines(s)
11823 local flatten_inlines = self.writer.flatten_inlines
11824 self.writer.flatten_inlines = true
11825 local flat_text = self.parser_functions.parse_inlines(s)
11826 flat_text = util.rope_to_string(flat_text)
11827 self.writer.flatten_inlines = flatten_inlines
11828 return {flat_text, inlines}
11829 end
11830
11831 -- parse atx header
11832 parsers.AtxHeading = parsers.check_trail_no_rem
11833 * Cg(parsers.heading_start, "level")

```

```

11834 * ((C(parsers.optionalspace
11835 * parsers.hash^0
11836 * parsers.optionalspace
11837 * parsers.newline)
11838 + parsers.spacechar^1
11839 * C(parsers.line))
11840 / strip_atx_end
11841 / parsers.parse_heading_text)
11842 * Cb("level")
11843 / writer.heading
11844
11845 parsers.heading_line = parsers.linechar^1
11846 - parsers.thematic_break_lines
11847
11848 parsers.heading_text = parsers.heading_line
11849 * ((V("Endline") / "\n")
11850 * (parsers.heading_line
11851 - parsers.heading_level))^0
11852 * parsers.newline^-1
11853
11854 parsers.SettextHeading = parsers.freeze_trail
11855 * parsers.check_trail_no_rem
11856 * #(parsers.heading_text
11857 * parsers.check_minimal_indent
11858 * parsers.check_trail
11859 * parsers.heading_level)
11860 * Cs(parsers.heading_text)
11861 / parsers.parse_heading_text
11862 * parsers.check_minimal_indent_and_trail
11863 * parsers.heading_level
11864 * parsers.newline
11865 * parsers.unfreeze_trail
11866 / writer.heading
11867
11868 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.

```

11869 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.

```

11870 local walkable_syntax = (function(global_walkable_syntax)
11871 local local_walkable_syntax = {}
11872 for lhs, rule in pairs(global_walkable_syntax) do
11873 local_walkable_syntax[lhs] = util.table_copy(rule)
11874 end
11875 return local_walkable_syntax
11876 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.

```

11877 local current_extension_name = nil
11878 self.insert_pattern = function(selector, pattern, pattern_name)
11879 assert(pattern_name == nil or type(pattern_name) == "string")
11880 local _, _, lhs, pos, rhs
11881 = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
11882 assert(lhs ~= nil,
11883 [[Expected selector in form]]
11884 .. [["LHS (before|after|instead of) RHS", not "]]
11885 .. selector .. [["]]))
11886 assert(walkable_syntax[lhs] ~= nil,
11887 [[Rule]] .. lhs
11888 .. [[-> ... does not exist in markdown grammar]])
11889 assert(pos == "before" or pos == "after" or pos == "instead of",
11890 [[Expected positional specifier "before", "after",]]
11891 .. [[or "instead of", not "]]
11892 .. pos .. [["]]))
11893 local rule = walkable_syntax[lhs]
11894 local index = nil
11895 for current_index, current_rhs in ipairs(rule) do
11896 if type(current_rhs) == "string" and current_rhs == rhs then
11897 index = current_index
11898 if pos == "after" then
11899 index = index + 1
11900 end
11901 break
11902 end
11903 end
11904 assert(index ~= nil,
11905 [[Rule]] .. lhs .. [[->]] .. rhs
11906 .. [[does not exist in markdown grammar]])
11907 local accountable_pattern
11908 if current_extension_name then
11909 accountable_pattern
11910 = {pattern, current_extension_name, pattern_name}

```

```

11911 else
11912 assert(type(pattern) == "string",
11913 [[reader->insert_pattern() was called outside]]
11914 .. [[an extension with]]
11915 .. [[a PEG pattern instead of a rule name]])
11916 accountable_pattern = pattern
11917 end
11918 if pos == "instead of" then
11919 rule[index] = accountable_pattern
11920 else
11921 table.insert(rule, index, accountable_pattern)
11922 end
11923 -- TODO: Remove all occurrences of `pattern` after `index`
11924 -- to improve speed?
11925 end
11926 if options.htmlOverLinks then
11927 self.insert_pattern("Inline before AutoLinkUrl", "InlineHtml")
11928 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.

```

11929 local syntax =
11930 { "Blocks",
11931
11932 Blocks = V("InitializeState")
11933 * V("ExpectedJekyllData")
11934 * 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.

```

11935 * (V("Block")
11936 * (V("Blank")^0 * parsers.eof
11937 + (V("Blank")^2 / writer.paragraphsep
11938 + V("Blank")^0 / writer.interblocksep
11939)
11940)
11941 + (V("Paragraph") + V("Plain"))
11942 * (V("Blank")^0 * parsers.eof
11943 + (V("Blank")^2 / writer.paragraphsep
11944 + V("Blank")^0 / writer.interblocksep
11945)
11946)
11947 * V("Block")
11948 * (V("Blank")^0 * parsers.eof
11949 + (V("Blank")^2 / writer.paragraphsep
11950 + V("Blank")^0 / writer.interblocksep
11951)

```



```

11952)
11953 + (V("Paragraph") + V("Plain"))
11954 * (V("Blank")^0 * parsers.eof
11955 + V("Blank")^0 / writer.paragraphsep
11956)
11957)^0,
11958
11959 ExpectedJekyllData = parsers.succeed,
11960
11961 Blank = parsers.Blank,
11962 Reference = parsers.Reference,
11963
11964 Blockquote = parsers.Blockquote,
11965 Verbatim = parsers.Verbatim,
11966 ThematicBreak = parsers.ThematicBreak,
11967 BulletList = parsers.BulletList,
11968 OrderedList = parsers.OrderedList,
11969 DisplayHtml = parsers.DisplayHtml,
11970 Heading = parsers.Heading,
11971 Paragraph = parsers.Paragraph,
11972 Plain = parsers.Plain,
11973
11974 ListStarter = parsers.ListStarter,
11975 EndlineExceptions = parsers.EndlineExceptions,
11976 NoSoftLineBreakEndlineExceptions
11977 = parsers.NoSoftLineBreakEndlineExceptions,
11978
11979 Str = parsers.Str,
11980 Space = parsers.Space,
11981 OptionalIndent = parsers.OptionalIndent,
11982 Endline = parsers.Endline,
11983 EndlineNoSub = parsers.EndlineNoSub,
11984 NoSoftLineBreakEndline
11985 = parsers.NoSoftLineBreakEndline,
11986 EndlineBreak = parsers.EndlineBreak,
11987 LinkAndEmph = parsers.LinkAndEmph,
11988 Code = parsers.Code,
11989 AutoLinkUrl = parsers.AutoLinkUrl,
11990 AutoLinkEmail = parsers.AutoLinkEmail,
11991 AutoLinkRelativeReference
11992 = parsers.AutoLinkRelativeReference,
11993 InlineHtml = parsers.InlineHtml,
11994 HtmlEntity = parsers.HtmlEntity,
11995 EscapedChar = parsers.EscapedChar,
11996 Smart = parsers.Smart,
11997 Symbol = parsers.Symbol,
11998 SpecialChar = parsers.fail,

```

```

11999 InitializeState = parsers.succeed,
12000 }

```

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]`.

```

12001 self.update_rule = function(rule_name, get_pattern)
12002 assert(current_extension_name ~= nil)
12003 assert(syntax[rule_name] ~= nil,
12004 [[Rule]] .. rule_name
12005 .. [[-> ... does not exist in markdown grammar]])
12006 local previous_pattern
12007 local extension_name
12008 if walkable_syntax[rule_name] then
12009 local previous_accountable_pattern
12010 = walkable_syntax[rule_name][1]
12011 previous_pattern = previous_accountable_pattern[1]
12012 extension_name
12013 = previous_accountable_pattern[2]
12014 .. ", " .. current_extension_name
12015 else
12016 previous_pattern = nil
12017 extension_name = current_extension_name
12018 end
12019 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

```

```

12020 if type(get_pattern) == "function" then
12021 pattern = get_pattern(previous_pattern)
12022 else
12023 assert(previous_pattern == nil,
12024 [[Rule]] .. rule_name ..
12025 [[has already been updated by]] .. extension_name)
12026 pattern = get_pattern
12027 end

```

```

12028 local accountable_pattern = { pattern, extension_name, rule_name }
12029 walkable_syntax[rule_name] = { accountable_pattern }
12030 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.

```

12031 local special_characters = {}
12032 self.add_special_character = function(c)
12033 table.insert(special_characters, c)
12034 syntax.SpecialChar = S(table.concat(special_characters, ""))
12035 end
12036
12037 self.add_special_character("*")
12038 self.add_special_character("[")
12039 self.add_special_character("]")
12040 self.add_special_character("<")
12041 self.add_special_character("!")
12042 self.add_special_character("\\")

```

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

```

12043 self.initialize_named_group = function(name, value)
12044 local pattern = Ct("")
12045 if value ~= nil then
12046 pattern = pattern / value
12047 end
12048 syntax.InitializeState = syntax.InitializeState
12049 * Cg(pattern, name)
12050 end

```

Add a named group for indentation.

```

12051 self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

12052 for _, extension in ipairs(extensions) do
12053 current_extension_name = extension.name
12054 extension.extend_writer(writer)
12055 extension.extend_reader(self)
12056 end
12057 current_extension_name = nil

```

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

```

12058 if options.debugExtensions then
12059 local sorted_lhs = {}
12060 for lhs, _ in pairs(walkable_syntax) do
12061 table.insert(sorted_lhs, lhs)

```

```

12062 end
12063 table.sort(sorted_lhs)
12064
12065 local output_lines = {"{"}
12066 for lhs_index, lhs in ipairs(sorted_lhs) do
12067 local encoded_lhs = util.encode_json_string(lhs)
12068 table.insert(output_lines, [[]] .. encoded_lhs .. [[:]])
12069 local rule = walkable_syntax[lhs]
12070 for rhs_index, rhs in ipairs(rule) do
12071 local human_readable_rhs
12072 if type(rhs) == "string" then
12073 human_readable_rhs = rhs
12074 else
12075 local pattern_name
12076 if rhs[3] then
12077 pattern_name = rhs[3]
12078 else
12079 pattern_name = "Anonymous Pattern"
12080 end
12081 local extension_name = rhs[2]
12082 human_readable_rhs = pattern_name .. [[(]]
12083 .. extension_name .. [[)]]
12084 end
12085 local encoded_rhs
12086 = util.encode_json_string(human_readable_rhs)
12087 local output_line = [[]] .. encoded_rhs
12088 if rhs_index < #rule then
12089 output_line = output_line .. ", "
12090 end
12091 table.insert(output_lines, output_line)
12092 end
12093 local output_line = "]"
12094 if lhs_index < #sorted_lhs then
12095 output_line = output_line .. ", "
12096 end
12097 table.insert(output_lines, output_line)
12098 end
12099 table.insert(output_lines, "}")
12100
12101 local output = table.concat(output_lines, "\n")
12102 local output_filename = options.debugExtensionsFileName
12103 local output_file = assert(io.open(output_filename, "w"),
12104 [[Could not open file]] .. output_filename
12105 .. [[for writing]])
12106 assert(output_file:write(output))
12107 assert(output_file:close())
12108 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`.

```
12109 for lhs, rule in pairs(walkable_syntax) do
12110 syntax[lhs] = parsers.fail
12111 for _, rhs in ipairs(rule) do
12112 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.

```
12113 if type(rhs) == "string" then
12114 pattern = V(rhs)
12115 else
12116 pattern = rhs[1]
12117 if type(pattern) == "string" then
12118 pattern = V(pattern)
12119 end
12120 end
12121 syntax[lhs] = syntax[lhs] + pattern
12122 end
12123 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.

```
12124 if options.underscores then
12125 self.add_special_character("_")
12126 end
12127
12128 if not options.codeSpans then
12129 syntax.Code = parsers.fail
12130 else
12131 self.add_special_character("`")
12132 end
12133
12134 if not options.html then
12135 syntax.DisplayHtml = parsers.fail
12136 syntax.InlineHtml = parsers.fail
12137 syntax.HtmlEntity = parsers.fail
12138 else
12139 self.add_special_character("&")
12140 end
12141
12142 if options.preserveTabs then
12143 options.stripIndent = false
```

```

12144 end
12145
12146 if not options.smartEllipses then
12147 syntax.Smart = parsers.fail
12148 else
12149 self.add_special_character(".")
12150 end
12151
12152 if not options.relativeReferences then
12153 syntax.AutoLinkRelativeReference = parsers.fail
12154 end
12155
12156 if options.contentLevel == "inline" then
12157 syntax[1] = "Inlines"
12158 syntax.Inlines = V("InitializeState")
12159 * parsers.Inline^0
12160 * (parsers.spacing^0
12161 * parsers.eof / "")
12162 syntax.Space = parsers.Space + parsers.blankline / writer.space
12163 end
12164
12165 local blocks_nested_t = util.table_copy(syntax)
12166 blocks_nested_t.ExpectedJekyllData = parsers.succeed
12167 parsers.blocks_nested = Ct(blocks_nested_t)
12168
12169 parsers.blocks = Ct(syntax)
12170
12171 local inlines_t = util.table_copy(syntax)
12172 inlines_t[1] = "Inlines"
12173 inlines_t.Inlines = V("InitializeState")
12174 * parsers.Inline^0
12175 * (parsers.spacing^0
12176 * parsers.eof / "")
12177 parsers.inlines = Ct(inlines_t)
12178
12179 local inlines_no_inline_note_t = util.table_copy(inlines_t)
12180 inlines_no_inline_note_t.InlineNote = parsers.fail
12181 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
12182
12183 local inlines_no_html_t = util.table_copy(inlines_t)
12184 inlines_no_html_t.DisplayHtml = parsers.fail
12185 inlines_no_html_t.InlineHtml = parsers.fail
12186 inlines_no_html_t.HtmlEntity = parsers.fail
12187 parsers.inlines_no_html = Ct(inlines_no_html_t)
12188
12189 local inlines_nbsp_t = util.table_copy(inlines_t)
12190 inlines_nbsp_t.Endline = parsers.NonbreakingEndline

```

```

12191 inlines_nbsp_t.Space = parsers.NonbreakingSpace
12192 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
12193
12194 local inlines_identity_t = util.table_copy(inlines_t)
12195 inlines_identity_t.Str = parsers.any / function(s) return s end
12196 parsers.inlines_identity = Ct(inlines_identity_t)
12197
12198 local inlines_string_t = util.table_copy(inlines_t)
12199 inlines_string_t.Str = parsers.any / writer.string
12200 parsers.inlines_string = Ct(inlines_string_t)
12201
12202 local inlines_math_t = util.table_copy(inlines_t)
12203 inlines_math_t.Str = parsers.any / writer.math
12204 parsers.inlines_math = Ct(inlines_math_t)
12205
12206 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
12207 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
12208 inlines_no_link_or_emphasis_t.EndlineExceptions
12209 = parsers.EndlineExceptions - parsers.eof
12210 parsers.inlines_no_link_or_emphasis
12211 = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain TeX output and returns it..

```

12212 return function(input)

```

Normalize the input.

```

12213 if options.unicodeNormalization then
12214 local form = options.unicodeNormalizationForm
12215 input = util.normalize(input, form)
12216 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.

```

12217 input = input:gsub("\r\n?", "\n")
12218 if input:sub(-1) ~= "\n" then
12219 input = input .. "\n"
12220 end

```

Clear the table of references.

```

12221 references = {}
12222 local document = self.parser_functions.parse_blocks(input)
12223 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`.

```

12224 local undosep_start, undosep_end
12225 local potential_secend_start, secend_start

```

```

12226 local potential_sep_start, sep_start
12227 while true do
12228 -- find a `writer->undosep`
12229 undosep_start, undosep_end
12230 = output:find(writer.undosep_text, 1, true)
12231 if undosep_start == nil then break end
12232 -- skip any preceding section ends
12233 secend_start = undosep_start
12234 while true do
12235 potential_secend_start = secend_start - #writer.secend_text
12236 if potential_secend_start < 1
12237 or output:sub(potential_secend_start,
12238 secend_start - 1) ~= writer.secend_text
12239 then
12240 break
12241 end
12242 secend_start = potential_secend_start
12243 end
12244 -- find an immediately preceding
12245 -- block element / paragraph separator
12246 sep_start = secend_start
12247 potential_sep_start = sep_start - #writer.interblocksep_text
12248 if potential_sep_start >= 1
12249 and output:sub(potential_sep_start,
12250 sep_start - 1) == writer.interblocksep_text
12251 then
12252 sep_start = potential_sep_start
12253 else
12254 potential_sep_start = sep_start - #writer.paragraphsep_text
12255 if potential_sep_start >= 1
12256 and output:sub(potential_sep_start,
12257 sep_start - 1) == writer.paragraphsep_text
12258 then
12259 sep_start = potential_sep_start
12260 end
12261 end
12262 -- remove `writer->undosep` and immediately preceding
12263 -- block element / paragraph separator
12264 output = output:sub(1, sep_start - 1)
12265 .. output:sub(secend_start, undosep_start - 1)
12266 .. output:sub(undosep_end + 1)
12267 end
12268 return output
12269 end
12270 end
12271 return self
12272 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.

```
12273 M.extensions = {}
```

#### 3.1.7.1 Acronyms

The `extensions.acronyms` function recognizes acronyms, initialisms, and other all-caps sequences in markdown text.

```
12274 M.extensions.acronyms = function(acronyms)
12275 assert(#acronyms > 0)
12276 return {
12277 name = "built-in acronyms syntax extension",
12278 extend_writer = function(self)
```

Define `writer->acronym` as a function that will transform an input acronym, initialism, or another all-caps sequence to the output format.

```
12279 function self.acronym(s)
12280 if self.flatten_inlines then return s end
12281 return {"\\markdownRendererAcronym{" ,s,""} }
12282 end
12283 end, extend_reader = function(self)
12284 local parsers = self.parsers
12285 local options = self.options
12286 local writer = self.writer
```

In order to minimize the size and speed of the parser, we will first construct prefix trees of all registered acronyms, initialisms, and other all-caps sequences.

```
12287 local prefix_tree = {_type = "intermediate"}
12288 for _, acronym in ipairs(acronyms) do
12289 local node = prefix_tree
```

Normalize each acronym, initialism, or another all-caps sequence.

```
12290 if options.unicodeNormalization then
12291 local form = options.unicodeNormalizationForm
12292 acronym = util.normalize(acronym, form)
12293 end
12294 for i = 1, #acronym do
12295 local acronym_byte = acronym:sub(i, i)
12296 if i < #acronym then
12297 if node[acronym_byte] == nil then
12298 node[acronym_byte] = {_type = "intermediate"}
12299 end
12300 node = node[acronym_byte]
```

```

12301 else
12302 table.insert(node, {acronym_byte, _type = "leaf"})
12303 end
12304 end
12305 end

```

Next, we will construct a parser out of the prefix tree.

```

12306 local subparsers = {}
12307 depth_first_search(prefix_tree, "", function(node, path)
12308 if node._type == "leaf" then
12309 local acronym_byte = table.unpack(node)
12310 if subparsers[path] == nil then
12311 subparsers[path] = parsers.fail
12312 end
12313 subparsers[path] = subparsers[path] + P(acronym_byte)
12314 end
12315 end, function(_, path)
12316 if #path > 0 then
12317 local byte = path:sub(#path, #path)
12318 local parent_path = path:sub(1, #path-1)
12319 local prefix = P(byte)
12320 local suffix = prefix * subparsers[path]
12321 if subparsers[parent_path] == nil then
12322 subparsers[parent_path] = parsers.fail
12323 end
12324 subparsers[parent_path] = subparsers[parent_path] + suffix
12325 end
12326 end)
12327 assert(subparsers[""] ~= nil)

```

Only match acronyms at word boundaries.

```

12328 local Acronym = (parsers.unicode.punctuation
12329 - V("SpecialChar")
12330 + parsers.unicode.whitespace
12331 - V("EndlineExceptions"))^0
12332 / writer.string
12333 * (
12334 subparsers[""]
12335 * #(-parsers.normalchar
12336 + parsers.unicode.punctuation
12337 + parsers.unicode.whitespace
12338 + parsers.eof)
12339 / writer.acronym)
12340 self.insert_pattern("Inline before Str", Acronym, "Acronym")
12341 end
12342 }
12343 end
12344 %

```

```

12345 ##### Bracketed Spans
12346 %
12347 % The \luamdef{extensions.bracketed_spans} function implements the Pandoc
12348 % bracketed span syntax extension.
12349 %
12350 % \end{markdown}
12351 % \begin{macrocode}
12352 M.extensions.bracketed_spans = function()
12353 return {
12354 name = "built-in bracketed_spans syntax extension",
12355 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.

```

12356 function self.span(s, attr)
12357 if self.flatten_inlines then return s end
12358 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
12359 self.attributes(attr),
12360 "\\markdownRendererBracketedSpan{" ,s,"}",
12361 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"
12362 end
12363 end, extend_reader = function(self)
12364 local parsers = self.parsers
12365 local writer = self.writer
12366
12367 local span_label = parsers.lbracket
12368 * (Cs((parsers.alphanumeric~1
12369 + parsers.inticks
12370 + parsers.autolink
12371 + V("InlineHtml")
12372 + (parsers.backslash * parsers.backslash)
12373 + (parsers.backslash
12374 * (parsers.lbracket + parsers.rbracket)
12375 + V("Space") + V("Endline")
12376 + (parsers.any
12377 - (parsers.newline
12378 + parsers.lbracket
12379 + parsers.rbracket
12380 + parsers.blankline~2))))~1)
12381 / self.parser_functions.parse_inlines)
12382 * parsers.rbracket
12383
12384 local Span = span_label
12385 * Ct(parsers.attributes)
12386 / writer.span
12387
12388 self.insert_pattern("Inline before LinkAndEmph",

```

```

12389 Span, "Span")
12390 end
12391 }
12392 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.

```

12393 M.extensions.citations = function(citation_nbsps)
12394 return {
12395 name = "built-in citations syntax extension",
12396 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.

```

12397 function self.citations(text_cites, cites)
12398 local buffer = {}
12399 if self.flatten_inlines then
12400 for _,cite in ipairs(cites) do
12401 if cite.prenote then
12402 table.insert(buffer, {cite.prenote, " "})
12403 end
12404 table.insert(buffer, cite.name)
12405 if cite.postnote then
12406 table.insert(buffer, {" ", cite.postnote})
12407 end
12408 end
12409 else
12410 table.insert(buffer,
12411 {"\\markdownRenderer",
12412 text_cites and "TextCite" or "Cite",

```

```

12413 "{", #cites, "}")
12414 for _,cite in ipairs(cites) do
12415 table.insert(buffer,
12416 {cite.suppress_author and "-" or "+", "{",
12417 cite.prenote or "", "}{" ,
12418 cite.postnote or "", "}{" , cite.name, "}")
12419 end
12420 end
12421 return buffer
12422 end
12423 end, extend_reader = function(self)
12424 local parsers = self.parsers
12425 local writer = self.writer
12426
12427 local citation_chars
12428 = parsers.alphanumeric
12429 + S("#$%&-+<>~/_")
12430
12431 local citation_name
12432 = Cs(parsers.dash~1) * parsers.at
12433 * Cs(citation_chars
12434 * (((citation_chars
12435 + parsers.internal_punctuation
12436 - parsers.comma - parsers.semicolon)
12437 * -#((parsers.internal_punctuation
12438 - parsers.comma
12439 - parsers.semicolon)^0
12440 * -(citation_chars
12441 + parsers.internal_punctuation
12442 - parsers.comma
12443 - parsers.semicolon)))^0
12444 * citation_chars)^-1)
12445
12446 local citation_body_prenote
12447 = Cs((parsers.alphanumeric^1
12448 + parsers.bracketed
12449 + parsers.inticks
12450 + parsers.autolink
12451 + V("InlineHtml")
12452 + V("Space") + V("EndlineNoSub")
12453 + (parsers.anyescaped
12454 - (parsers.newline
12455 + parsers.rbracket
12456 + parsers.blankline^2))
12457 - (parsers.spnl
12458 * parsers.dash~1
12459 * parsers.at))^1)

```

```

12460
12461 local citation_body_postnote
12462 = Cs((parsers.alphanumeric^1
12463 + parsers.bracketed
12464 + parsers.inticks
12465 + parsers.autolink
12466 + V("InlineHtml")
12467 + V("Space") + V("EndlineNoSub")
12468 + (parsers.anyescaped
12469 - (parsers.newline
12470 + parsers.rbracket
12471 + parsers.semicolon
12472 + parsers.blankline^2))
12473 - (parsers.spnl * parsers.rbracket))^1)
12474
12475 local citation_body_chunk
12476 = (citation_body_prenote
12477 * parsers.spnlc_sep
12478 + Cc("")
12479 * parsers.spnlc
12480)
12481 * citation_name
12482 * (parsers.internal_punctuation
12483 - parsers.semicolon)^-1
12484 * (parsers.spnlc / function(_) return end
12485 * citation_body_postnote
12486 + Cc("")
12487 * parsers.spnlc
12488)
12489
12490 local citation_body
12491 = citation_body_chunk
12492 * (parsers.semicolon
12493 * parsers.spnlc
12494 * citation_body_chunk
12495)^0
12496
12497 local citation_headless_body_postnote
12498 = Cs((parsers.alphanumeric^1
12499 + parsers.bracketed
12500 + parsers.inticks
12501 + parsers.autolink
12502 + V("InlineHtml")
12503 + V("Space") + V("Endline")
12504 + (parsers.anyescaped
12505 - (parsers.newline
12506 + parsers.rbracket

```

```

12507 + parsers.at
12508 + parsers.semicolon + parsers.blankline^2))
12509 - (parsers.spnl * parsers.rbracket))^0)
12510
12511 local citation_headless_body
12512 = citation_headless_body_postnote
12513 * (parsers.semicolon
12514 * parsers.spnlc
12515 * citation_body_chunk
12516)^0
12517
12518 local citations
12519 = function(text_cites, raw_cites)
12520 local function normalize(str)
12521 if str == "" then
12522 str = nil
12523 else
12524 str = (citation_nbsps and
12525 self.parser_functions.parse_inlines_nbsp or
12526 self.parser_functions.parse_inlines)(str)
12527 end
12528 return str
12529 end
12530
12531 local cites = {}
12532 for i = 1,#raw_cites,4 do
12533 cites[#cites+1] = {
12534 prenote = normalize(raw_cites[i]),
12535 suppress_author = raw_cites[i+1] == "-",
12536 name = writer.identifier(raw_cites[i+2]),
12537 postnote = normalize(raw_cites[i+3]),
12538 }
12539 end
12540 return writer.citations(text_cites, cites)
12541 end
12542
12543 local TextCitations
12544 = Ct((parsers.spnlc
12545 * Cc("")
12546 * citation_name
12547 * ((parsers.spnlc
12548 * parsers.lbracket
12549 * citation_headless_body
12550 * parsers.rbracket) + Cc("")))^1)
12551 / function(raw_cites)
12552 return citations(true, raw_cites)
12553 end

```

```

12554
12555 local ParenthesizedCitations
12556 = Ct((parsers.spnlc
12557 * parsers.lbracket
12558 * citation_body
12559 * parsers.rbracket)^1)
12560 / function(raw_cites)
12561 return citations(false, raw_cites)
12562 end
12563
12564 local Citations = TextCitations + ParenthesizedCitations
12565
12566 self.insert_pattern("Inline before LinkAndEmph",
12567 Citations, "Citations")
12568
12569 self.add_special_character("@")
12570 self.add_special_character("-")
12571 end
12572 }
12573 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.

```

12574 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.

```

12575 local languages_json = (function()
12576 local base, prev, curr
12577 for _, pathname in ipairs(util.find_files(language_map)) do
12578 local file = io.open(pathname, "r")
12579 if not file then goto continue end
12580 local input = assert(file:read("*a"))
12581 assert(file:close())
12582 local json = input:gsub('"[^\\n]-":', '[%1]=')
12583 curr = load("_ENV = {}; return "..json")()
12584 if type(curr) == "table" then
12585 if base == nil then
12586 base = curr
12587 else
12588 setmetatable(prev, { __index = curr })
12589 end

```



```

12590 prev = curr
12591 end
12592 ::continue::
12593 end
12594 return base or {}
12595 end()()
12596
12597 return {
12598 name = "built-in content_blocks syntax extension",
12599 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.

```

12600 function self.contentblock(src,suf,type,tit)
12601 if not self.is_writing then return "" end
12602 src = src.." "..suf
12603 suf = suf:lower()
12604 if type == "onlineimage" then
12605 return {"\\markdownRendererContentBlockOnlineImage{" ,suf,""},
12606 {"",self.string(src),""},
12607 {"",self.uri(src),""},
12608 {"",self.string(tit or ""),"}}"
12609 elseif languages_json[suf] then
12610 return {"\\markdownRendererContentBlockCode{" ,suf,""},
12611 {"",self.string(languages_json[suf]),""},
12612 {"",self.string(src),""},
12613 {"",self.uri(src),""},
12614 {"",self.string(tit or ""),"}}"
12615 else
12616 return {"\\markdownRendererContentBlock{" ,suf,""},
12617 {"",self.string(src),""},
12618 {"",self.uri(src),""},
12619 {"",self.string(tit or ""),"}}"
12620 end
12621 end
12622 end, extend_reader = function(self)
12623 local parsers = self.parsers
12624 local writer = self.writer
12625
12626 local contentblock_tail
12627 = parsers.optionaltitle
12628 * (parsers.newline + parsers.eof)
12629
12630 -- case insensitive online image suffix:
12631 local onlineimagesuffix

```

```

12632 = (function(...)
12633 local parser = nil
12634 for _, suffix in ipairs({...}) do
12635 local pattern=nil
12636 for i=1,#suffix do
12637 local char=suffix:sub(i,i)
12638 char = S(char:lower()..char:upper())
12639 if pattern == nil then
12640 pattern = char
12641 else
12642 pattern = pattern * char
12643 end
12644 end
12645 if parser == nil then
12646 parser = pattern
12647 else
12648 parser = parser + pattern
12649 end
12650 end
12651 return parser
12652 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
12653
12654 -- online image url for iA Writer content blocks with
12655 -- mandatory suffix, allowing nested brackets:
12656 local onlineimageurl
12657 = (parsers.less
12658 * Cs((parsers.anyescaped
12659 - parsers.more
12660 - parsers.spacing
12661 - #(parsers.period
12662 * onlineimagesuffix
12663 * parsers.more
12664 * contentblock_tail))^0)
12665 * parsers.period
12666 * Cs(onlineimagesuffix)
12667 * parsers.more
12668 + (Cs((parsers.inparens
12669 + (parsers.anyescaped
12670 - parsers.spacing
12671 - parsers.rparent
12672 - #(parsers.period
12673 * onlineimagesuffix
12674 * contentblock_tail))))^0)
12675 * parsers.period
12676 * Cs(onlineimagesuffix))
12677) * Cc("onlineimage")
12678

```

```

12679 -- filename for iA Writer content blocks with mandatory suffix:
12680 local localfilepath
12681 = parsers.slash
12682 * Cs((parsers.anyescaped
12683 - parsers.tab
12684 - parsers.newline
12685 - #(parsers.period
12686 * parsers.alphanumeric^1
12687 * contentblock_tail))^1)
12688 * parsers.period
12689 * Cs(parsers.alphanumeric^1)
12690 * Cc("localfile")
12691
12692 local ContentBlock
12693 = parsers.check_trail_no_rem
12694 * (localfilepath + onlineimageurl)
12695 * contentblock_tail
12696 / writer.contentblock
12697
12698 self.insert_pattern("Block before Blockquote",
12699 ContentBlock, "ContentBlock")
12700 end
12701 }
12702 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.

```

12703 M.extensions.definition_lists = function(tight_lists)
12704 return {
12705 name = "built-in definition_lists syntax extension",
12706 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.

```

12707 local function dlistem(term, defs)
12708 local retVal = {"\\markdownRendererDlItem{",term,""}
12709 for _, def in ipairs(defs) do
12710 retVal[#retVal+1]
12711 = {"\\markdownRendererDlDefinitionBegin ",def,
12712 "\\markdownRendererDlDefinitionEnd "}
12713 end
12714 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "

```

```

12715 return retVal
12716 end
12717
12718 function self.definitionlist(items,tight)
12719 if not self.is_writing then return "" end
12720 local buffer = {}
12721 for _,item in ipairs(items) do
12722 buffer[#buffer + 1] = dlititem(item.term, item.definitions)
12723 end
12724 if tight and tight_lists then
12725 return {"\\markdownRendererDlBeginTight\n", buffer,
12726 "\n\\markdownRendererDlEndTight"}
12727 else
12728 return {"\\markdownRendererDlBegin\n", buffer,
12729 "\n\\markdownRendererDlEnd"}
12730 end
12731 end
12732 end, extend_reader = function(self)
12733 local parsers = self.parsers
12734 local writer = self.writer
12735
12736 local defstartchar = S("~:")
12737
12738 local defstart
12739 = parsers.check_trail_length(0) * defstartchar
12740 * #parsers.spacing
12741 * (parsers.tab + parsers.space^-3)
12742 + parsers.check_trail_length(1)
12743 * defstartchar * #parsers.spacing
12744 * (parsers.tab + parsers.space^-2)
12745 + parsers.check_trail_length(2)
12746 * defstartchar * #parsers.spacing
12747 * (parsers.tab + parsers.space^-1)
12748 + parsers.check_trail_length(3)
12749 * defstartchar * #parsers.spacing
12750
12751 local indented_line
12752 = (parsers.check_minimal_indent / "")
12753 * parsers.check_code_trail * parsers.line
12754
12755 local blank
12756 = parsers.check_minimal_blank_indent_and_any_trail
12757 * parsers.optionalspace * parsers.newline
12758
12759 local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
12760
12761 local indented_blocks = function(bl)

```

```

12762 return Cs(bl
12763 * (blank^1 * (parsers.check_minimal_indent / ""))
12764 * parsers.check_code_trail * -parsers.blankline * bl)^0
12765 * (blank^1 + parsers.eof))
12766 end
12767
12768 local function definition_list_item(term, defs, _)
12769 return { term = self.parser_functions.parse_inlines(term),
12770 definitions = defs }
12771 end
12772
12773 local DefinitionListItemLoose
12774 = C(parsers.line) * blank^0
12775 * Ct((parsers.check_minimal_indent * (defstart
12776 * indented_blocks(dlchunk)
12777 / self.parser_functions.parse_blocks_nested))^1)
12778 * Cc(false) / definition_list_item
12779
12780 local DefinitionListItemTight
12781 = C(parsers.line)
12782 * Ct((parsers.check_minimal_indent * (defstart * dlchunk
12783 / self.parser_functions.parse_blocks_nested))^1)
12784 * Cc(true) / definition_list_item
12785
12786 local DefinitionList
12787 = (Ct(DefinitionListItemLoose^1) * Cc(false)
12788 + Ct(DefinitionListItemTight^1)
12789 * (blank^0
12790 * -DefinitionListItemLoose * Cc(true))
12791) / writer.definitionlist
12792
12793 self.insert_pattern("Block after Heading",
12794 DefinitionList, "DefinitionList")
12795 end
12796 }
12797 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

12798 M.extensions.fancy_lists = function()
12799 return {
12800 name = "built-in fancy_lists syntax extension",
12801 extend_writer = function(self)
12802 local options = self.options
12803

```

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.

```

12804 function self.fancylist(items,tight,startnum,numstyle,numdelim)
12805 if not self.is_writing then return "" end
12806 local buffer = {}
12807 local num = startnum
12808 for _,item in ipairs(items) do
12809 if item ~= "" then
12810 buffer[#buffer + 1] = self.fancyitem(item,num)
12811 end
12812 if num ~= nil and item ~= "" then
12813 num = num + 1
12814 end
12815 end
12816 local contents = util.intersperse(buffer,"\n")
12817 if tight and options.tightLists then
12818 return {"\\markdownRendererFancy01BeginTight{",
12819 numstyle,"}{",numdelim,"}",contents,
12820 "\\n\\markdownRendererFancy01EndTight "}
12821 else
12822 return {"\\markdownRendererFancy01Begin{",
12823 numstyle,"}{",numdelim,"}",contents,
12824 "\\n\\markdownRendererFancy01End "}
12825 end
12826 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.

```

12827 function self.fancyitem(s,num)
12828 if num ~= nil then
12829 return {"\\markdownRendererFancyOliItemWithNumber{" ,num,"} ",s,
12830 "\\markdownRendererFancyOliItemEnd "}
12831 else
12832 return {"\\markdownRendererFancyOliItem ",s,
12833 "\\markdownRendererFancyOliItemEnd "}
12834 end
12835 end
12836 end, extend_reader = function(self)
12837 local parsers = self.parsers
12838 local options = self.options
12839 local writer = self.writer
12840
12841 local function combine_markers_and_delims(markers, delims)
12842 local markers_table = {}
12843 for _,marker in ipairs(markers) do
12844 local start_marker
12845 local continuation_marker
12846 if type(marker) == "table" then
12847 start_marker = marker[1]
12848 continuation_marker = marker[2]
12849 else
12850 start_marker = marker
12851 continuation_marker = marker
12852 end
12853 for _,delim in ipairs(delims) do
12854 table.insert(markers_table,
12855 {start_marker, continuation_marker, delim})
12856 end
12857 end
12858 return markers_table
12859 end
12860
12861 local function join_table_with_func(func, markers_table)
12862 local pattern = func(table.unpack(markers_table[1]))
12863 for i = 2, #markers_table do
12864 pattern = pattern + func(table.unpack(markers_table[i]))
12865 end
12866 return pattern
12867 end
12868
12869 local lowercase_letter_marker = R("az")
12870 local uppercase_letter_marker = R("AZ")
12871
12872 local roman_marker = function(chars)
12873 local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])

```

```

12874 local l, x, v, i
12875 = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
12876 return m^-3
12877 * (c*m + c*d + d^-1 * c^-3)
12878 * (x*c + x*l + l^-1 * x^-3)
12879 * (i*x + i*v + v^-1 * i^-3)
12880 end
12881
12882 local lowercase_roman_marker
12883 = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
12884 local uppercase_roman_marker
12885 = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
12886
12887 local lowercase_opening_roman_marker = P("i")
12888 local uppercase_opening_roman_marker = P("I")
12889
12890 local digit_marker = parsers.dig * parsers.dig^-8
12891
12892 local markers = {
12893 {lowercase_opening_roman_marker, lowercase_roman_marker},
12894 {uppercase_opening_roman_marker, uppercase_roman_marker},
12895 lowercase_letter_marker,
12896 uppercase_letter_marker,
12897 lowercase_roman_marker,
12898 uppercase_roman_marker,
12899 digit_marker
12900 }
12901
12902 local delims = {
12903 parsers.period,
12904 parsers.rparent
12905 }
12906
12907 local markers_table = combine_markers_and_delims(markers, delims)
12908
12909 local function enumerator(start_marker, _,
12910 delimiter_type, interrupting)
12911 local delimiter_range
12912 local allowed_end
12913 if interrupting then
12914 delimiter_range = P("1")
12915 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
12916 else
12917 delimiter_range = start_marker
12918 allowed_end = C(parsers.spacechar^1)
12919 + #(parsers.newline + parsers.eof)
12920 end

```



```

12921
12922 return parsers.check_trail
12923 * Ct(C(delimiter_range) * C(delimiter_type))
12924 * allowed_end
12925 end
12926
12927 local starter = join_table_with_func(enumerator, markers_table)
12928
12929 local TightListItem = function(starter)
12930 return parsers.add_indent(starter, "li")
12931 * parsers.indented_content_tight
12932 end
12933
12934 local LooseListItem = function(starter)
12935 return parsers.add_indent(starter, "li")
12936 * parsers.indented_content_loose
12937 * remove_indent("li")
12938 end
12939
12940 local function roman2number(roman)
12941 local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
12942 ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
12943 local numeral = 0
12944
12945 local i = 1
12946 local len = string.len(roman)
12947 while i < len do
12948 local z1, z2 = romans[string.sub(roman, i, i)],
12949 romans[string.sub(roman, i+1, i+1)]
12950 if z1 < z2 then
12951 numeral = numeral + (z2 - z1)
12952 i = i + 2
12953 else
12954 numeral = numeral + z1
12955 i = i + 1
12956 end
12957 end
12958 if i <= len then
12959 numeral = numeral + romans[string.sub(roman,i,i)]
12960 end
12961 return numeral
12962 end
12963
12964 local function sniffstyle(numstr, delimend)
12965 local numdelim
12966 if delimend == ")" then
12967 numdelim = "OneParen"

```

```

12968 elseif delimend == "." then
12969 numdelim = "Period"
12970 else
12971 numdelim = "Default"
12972 end
12973
12974 local num
12975 num = numstr:match("^([I])$")
12976 if num then
12977 return roman2number(num), "UpperRoman", numdelim
12978 end
12979 num = numstr:match("^([i])$")
12980 if num then
12981 return roman2number(string.upper(num)), "LowerRoman", numdelim
12982 end
12983 num = numstr:match("^([A-Z])$")
12984 if num then
12985 return string.byte(num) - string.byte("A") + 1,
12986 "UpperAlpha", numdelim
12987 end
12988 num = numstr:match("^([a-z])$")
12989 if num then
12990 return string.byte(num) - string.byte("a") + 1,
12991 "LowerAlpha", numdelim
12992 end
12993 num = numstr:match("^([IVXLCDM]+)")
12994 if num then
12995 return roman2number(num), "UpperRoman", numdelim
12996 end
12997 num = numstr:match("^([ivxlcdm]+)")
12998 if num then
12999 return roman2number(string.upper(num)), "LowerRoman", numdelim
13000 end
13001 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
13002 end
13003
13004 local function fancylist(items,tight,start)
13005 local startnum, numstyle, numdelim
13006 = sniffstyle(start[2][1], start[2][2])
13007 return writer.fancylist(items,tight,
13008 options.startNumber and startnum or 1,
13009 numstyle or "Decimal",
13010 numdelim or "Default")
13011 end
13012
13013 local FancyListOfType
13014 = function(start_marker, continuation_marker, delimiter_type)

```

```

13015 local enumerator_start
13016 = enumerator(start_marker, continuation_marker,
13017 delimiter_type)
13018 local enumerator_cont
13019 = enumerator(continuation_marker, continuation_marker,
13020 delimiter_type)
13021 return Cg(enumerator_start, "listtype")
13022 * (Ct(TightListItem(Cb("listtype"))
13023 * ((parsers.check_minimal_indent / "")
13024 * TightListItem(enumerator_cont))^0)
13025 * Cc(true)
13026 * -#((parsers.conditionally_indented_blankline^0 / "")
13027 * parsers.check_minimal_indent * enumerator_cont)
13028 + Ct(LooseListItem(Cb("listtype"))
13029 * ((parsers.conditionally_indented_blankline^0 / "")
13030 * (parsers.check_minimal_indent / "")
13031 * LooseListItem(enumerator_cont))^0)
13032 * Cc(false)
13033) * Ct(Cb("listtype")) / fancylist
13034 end
13035
13036 local FancyList
13037 = join_table_with_func(FancyListOfType, markers_table)
13038
13039 local ListStarter = starter
13040
13041 self.update_rule("OrderedList", FancyList)
13042 self.update_rule("ListStarter", ListStarter)
13043 end
13044 }
13045 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.

```

13046 M.extensions.fenced_code = function(blank_before_code_fence,
13047 allow_attributes,
13048 allow_raw_blocks)
13049 return {

```

```

13050 name = "built-in fenced_code syntax extension",
13051 extend_writer = function(self)
13052 local options = self.options
13053

```

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.

```

13054 function self.fencedCode(s, i, attr)
13055 if not self.is_writing then return "" end
13056 s = s:gsub("\n$", "")
13057 local buf = {}
13058 if attr ~= nil then
13059 table.insert(buf,
13060 {"\\markdownRendererFencedCodeAttributeContextBegin",
13061 self.attributes(attr)})
13062 end
13063 local name = util.cache_verbatim(options.cacheDir, s)
13064 table.insert(buf,
13065 {"\\markdownRendererInputFencedCode{",
13066 name,"}{",self.string(i),"}{",self.infostring(i),""}})
13067 if attr ~= nil then
13068 table.insert(buf,
13069 "\\markdownRendererFencedCodeAttributeContextEnd{")
13070 end
13071 return buf
13072 end
13073

```

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

```

13074 if allow_raw_blocks then
13075 function self.rawBlock(s, attr)
13076 if not self.is_writing then return "" end
13077 s = s:gsub("\n$", "")
13078 local name = util.cache_verbatim(options.cacheDir, s)
13079 return {"\\markdownRendererInputRawBlock{",
13080 name,"}{", self.string(attr),""}}
13081 end
13082 end
13083 end, extend_reader = function(self)
13084 local parsers = self.parsers
13085 local writer = self.writer
13086
13087 local function captures_geq_length(_,i,a,b)
13088 return #a >= #b and i
13089 end
13090
13091 local function strip_enclosing_whitespaces(str)

```

```

13092 return str:gsub("^%s*(.)%s*$", "%1")
13093 end
13094
13095 local tilde_infostring = Cs(Cs((V("HtmlEntity")
13096 + parsers.anyescaped
13097 - parsers.newline)^0)
13098 / strip_enclosing_whitespaces)
13099
13100 local backtick_infostring
13101 = Cs(Cs((V("HtmlEntity")
13102 + (-#(parsers.backslash * parsers.backtick)
13103 * parsers.anyescaped)
13104 - parsers.newline
13105 - parsers.backtick)^0)
13106 / strip_enclosing_whitespaces)
13107
13108 local fenceindent
13109
13110 local function has_trail(indent_table)
13111 return indent_table ~= nil and
13112 indent_table.trail ~= nil and
13113 next(indent_table.trail) ~= nil
13114 end
13115
13116 local function has_indents(indent_table)
13117 return indent_table ~= nil and
13118 indent_table.indents ~= nil and
13119 next(indent_table.indents) ~= nil
13120 end
13121
13122 local function get_last_indent_name(indent_table)
13123 if has_indents(indent_table) then
13124 return indent_table.indents[#indent_table.indents].name
13125 end
13126 end
13127
13128 local count_fenced_start_indent =
13129 function(_, _, indent_table, trail)
13130 local last_indent_name = get_last_indent_name(indent_table)
13131 fenceindent = 0
13132 if last_indent_name ~= "li" then
13133 fenceindent = #trail
13134 end
13135 return true
13136 end
13137
13138 local fencehead = function(char, infostring)

```

```

13139 return Cmt(Cb("indent_info")
13140 * parsers.check_trail, count_fenced_start_indent)
13141 * Cg(char^3, "fencelength")
13142 * parsers.optionalspace
13143 * infostring
13144 * (parsers.newline + parsers.eof)
13145 end
13146
13147 local fencetail = function(char)
13148 return parsers.check_trail_no_rem
13149 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
13150 * parsers.optionalspace * (parsers.newline + parsers.eof)
13151 + parsers.eof
13152 end
13153
13154 local process_fenced_line =
13155 function(s, i, -- luacheck: ignore s i
13156 indent_table, line_content, is_blank)
13157 local remainder = ""
13158 if has_trail(indent_table) then
13159 remainder = indent_table.trail.internal_remainder
13160 end
13161
13162 if is_blank
13163 and get_last_indent_name(indent_table) == "li" then
13164 remainder = ""
13165 end
13166
13167 local str = remainder .. line_content
13168 local index = 1
13169 local remaining = fenceindent
13170
13171 while true do
13172 local c = str:sub(index, index)
13173 if c == " " and remaining > 0 then
13174 remaining = remaining - 1
13175 index = index + 1
13176 elseif c == "\t" and remaining > 3 then
13177 remaining = remaining - 4
13178 index = index + 1
13179 else
13180 break
13181 end
13182 end
13183
13184 return true, str:sub(index)
13185 end

```

```

13186
13187 local fencedline = function(char)
13188 return Cmt(Cb("indent_info")
13189 * C(parsers.line - fencetail(char))
13190 * Cc(false), process_fenced_line)
13191 end
13192
13193 local blankfencedline
13194 = Cmt(Cb("indent_info")
13195 * C(parsers.blankline)
13196 * Cc(true), process_fenced_line)
13197
13198 local TildeFencedCode
13199 = fencehead(parsers.tilde, tilde_infostring)
13200 * Cs((parsers.check_minimal_blank_indent / "")
13201 * blankfencedline
13202 + (parsers.check_minimal_indent / "")
13203 * fencedline(parsers.tilde))^0)
13204 * ((parsers.check_minimal_indent / "")
13205 * fencetail(parsers.tilde) + parsers.succeed)
13206
13207 local BacktickFencedCode
13208 = fencehead(parsers.backtick, backtick_infostring)
13209 * Cs((parsers.check_minimal_blank_indent / "")
13210 * blankfencedline
13211 + (parsers.check_minimal_indent / "")
13212 * fencedline(parsers.backtick))^0)
13213 * ((parsers.check_minimal_indent / "")
13214 * fencetail(parsers.backtick) + parsers.succeed)
13215
13216 local infostring_with_attributes
13217 = Ct(C((parsers.linechar
13218 - (parsers.optionalspace
13219 * parsers.attributes))^0)
13220 * parsers.optionalspace
13221 * Ct(parsers.attributes))
13222
13223 local FencedCode
13224 = ((TildeFencedCode + BacktickFencedCode)
13225 / function(infostring, code)
13226 local expanded_code = self.expandtabs(code)
13227
13228 if allow_raw_blocks then
13229 local raw_attr = lpeg.match(parsers.raw_attribute,
13230 infostring)
13231
13232 if raw_attr then
13233 return writer.rawBlock(expanded_code, raw_attr)

```

```

13233 end
13234 end
13235
13236 local attr = nil
13237 if allow_attributes then
13238 local match = lpeg.match(fostring_with_attributes,
13239 fostring)
13240 if match then
13241 fostring, attr = table.unpack(match)
13242 end
13243 end
13244 return writer.fencedCode(expanded_code, fostring, attr)
13245 end)
13246
13247 self.insert_pattern("Block after Verbatim",
13248 FencedCode, "FencedCode")
13249
13250 local fencestart
13251 if blank_before_code_fence then
13252 fencestart = parsers.fail
13253 else
13254 fencestart = fencehead(parsers.backtick, backtick_fostring)
13255 + fencehead(parsers.tilde, tilde_fostring)
13256 end
13257
13258 self.update_rule("EndlineExceptions", function(previous_pattern)
13259 if previous_pattern == nil then
13260 previous_pattern = parsers.EndlineExceptions
13261 end
13262 return previous_pattern + fencestart
13263 end)
13264
13265 self.add_special_character("`")
13266 self.add_special_character("~")
13267 end
13268 }
13269 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.

```

13270 M.extensions.fenced_divs = function(blank_before_div_fence)
13271 return {
13272 name = "built-in fenced_divs syntax extension",

```



```
13273 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.

```
13274 function self.div_begin(attributes)
13275 local start_output
13276 = {"\\markdownRendererFencedDivAttributeContextBegin\\n",
13277 self.attributes(attributes)}
13278 local end_output
13279 = {"\\markdownRendererFencedDivAttributeContextEnd{}}"
13280 return self.push_attributes(
13281 "div", attributes, start_output, end_output)
13282 end
```

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

```
13283 function self.div_end()
13284 return self.pop_attributes("div")
13285 end
13286 end, extend_reader = function(self)
13287 local parsers = self.parsers
13288 local writer = self.writer
```

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

```
13289 local fenced_div_infostring
13290 = Ct(parsers.attributes)
13291 + (
13292 C(parsers.attribute_classname
13293 - parsers.colon)^1
13294 / function (infostring)
13295 return {"." .. infostring}
13296 end
13297)
13298
13299 local fenced_div_begin = parsers.nonindentSPACE
13300 * parsers.colon^3
13301 * parsers.optionalspace
13302 * fenced_div_infostring
13303 * (parsers.spacechar^1
13304 * parsers.colon^1)^0
13305 * parsers.optionalspace
13306 * (parsers.newline + parsers.eof)
13307
13308 local fenced_div_end = parsers.nonindentSPACE
13309 * parsers.colon^3
13310 * parsers.optionalspace
13311 * (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.

```

13312 self.initialize_named_group("fenced_div_level", "0")
13313 self.initialize_named_group("fenced_div_num_opening_indents")
13314
13315 local function increment_div_level()
13316 local push_indent_table =
13317 function(s, i, indent_table, -- luacheck: ignore s i
13318 fenced_div_num_opening_indents, fenced_div_level)
13319 fenced_div_level = tonumber(fenced_div_level) + 1
13320 local num_opening_indents = 0
13321 if indent_table.indents ~= nil then
13322 num_opening_indents = #indent_table.indents
13323 end
13324 fenced_div_num_opening_indents[fenced_div_level]
13325 = num_opening_indents
13326 return true, fenced_div_num_opening_indents
13327 end
13328
13329 local increment_level =
13330 function(s, i, fenced_div_level) -- luacheck: ignore s i
13331 fenced_div_level = tonumber(fenced_div_level) + 1
13332 return true, tostring(fenced_div_level)
13333 end
13334
13335 return Cg(Cmt(Cb("indent_info")
13336 * Cb("fenced_div_num_opening_indents")
13337 * Cb("fenced_div_level"), push_indent_table)
13338 , "fenced_div_num_opening_indents")
13339 * Cg(Cmt(Cb("fenced_div_level"), increment_level)
13340 , "fenced_div_level")
13341 end
13342
13343 local function decrement_div_level()
13344 local pop_indent_table =
13345 function(s, i, -- luacheck: ignore s i
13346 fenced_div_indent_table, fenced_div_level)
13347 fenced_div_level = tonumber(fenced_div_level)
13348 fenced_div_indent_table[fenced_div_level] = nil
13349 return true, tostring(fenced_div_level - 1)
13350 end
13351
```

```

13352 return Cg(Cmt(Cb("fenced_div_num_opening_indents")
13353 * Cb("fenced_div_level"), pop_indent_table)
13354 , "fenced_div_level")
13355 end
13356
13357
13358 local non_fenced_div_block
13359 = parsers.check_minimal_indent * V("Block")
13360 - parsers.check_minimal_indent_and_trail * fenced_div_end
13361
13362 local non_fenced_div_paragraph
13363 = parsers.check_minimal_indent * V("Paragraph")
13364 - parsers.check_minimal_indent_and_trail * fenced_div_end
13365
13366 local blank = parsers.minimally_indented_blank
13367
13368 local block_separated = parsers.block_sep_group(blank)
13369 * non_fenced_div_block
13370
13371 local loop_body_pair
13372 = parsers.create_loop_body_pair(block_separated,
13373 non_fenced_div_paragraph,
13374 parsers.block_sep_group(blank),
13375 parsers.par_sep_group(blank))
13376
13377 local content_loop = (non_fenced_div_block
13378 * loop_body_pair.block^0
13379 + non_fenced_div_paragraph
13380 * block_separated
13381 * loop_body_pair.block^0
13382 + non_fenced_div_paragraph
13383 * loop_body_pair.par^0)
13384 * blank^0
13385
13386 local FencedDiv = fenced_div_begin
13387 / writer.div_begin
13388 * increment_div_level()
13389 * parsers.skipblanklines
13390 * Ct(content_loop)
13391 * parsers.minimally_indented_blank^0
13392 * parsers.check_minimal_indent_and_trail
13393 * fenced_div_end
13394 * decrement_div_level()
13395 * (Cc("") / writer.div_end)
13396
13397 self.insert_pattern("Block after Verbatim",
13398 FencedDiv, "FencedDiv")

```

```

13399
13400 self.add_special_character(":")
13401

```

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.

```

13402 local function is_inside_div()
13403 local check_div_level =
13404 function(s, i, fenced_div_level) -- luacheck: ignore s i
13405 fenced_div_level = tonumber(fenced_div_level)
13406 return fenced_div_level > 0
13407 end
13408
13409 return Cmt(Cb("fenced_div_level"), check_div_level)
13410 end
13411
13412 local function check_indent()
13413 local compare_indent =
13414 function(s, i, indent_table, -- luacheck: ignore s i
13415 fenced_div_num_opening_indents, fenced_div_level)
13416 fenced_div_level = tonumber(fenced_div_level)
13417 local num_current_indents
13418 = (indent_table.current_line_indents ~= nil and
13419 #indent_table.current_line_indents) or 0
13420 local num_opening_indents
13421 = fenced_div_num_opening_indents[fenced_div_level]
13422 return num_current_indents == num_opening_indents
13423 end
13424
13425 return Cmt(Cb("indent_info")
13426 * Cb("fenced_div_num_opening_indents")
13427 * Cb("fenced_div_level"), compare_indent)
13428 end
13429
13430 local fencestart = is_inside_div()
13431 * fenced_div_end
13432 * check_indent()
13433
13434 if not blank_before_div_fence then
13435 self.update_rule("EndlineExceptions", function(previous_pattern)
13436 if previous_pattern == nil then
13437 previous_pattern = parsers.EndlineExceptions
13438 end
13439 return previous_pattern + fencestart
13440 end)
13441 end

```

```

13442 end
13443 }
13444 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

13445 M.extensions.header_attributes = function()
13446 return {
13447 name = "built-in header_attributes syntax extension",
13448 extend_writer = function()
13449 end, extend_reader = function(self)
13450 local parsers = self.parsers
13451 local writer = self.writer
13452
13453 local function strip_atx_end(s)
13454 return s:gsub("%s+##%s*$", "")
13455 end
13456
13457 local AtxHeading = Cg(parsers.heading_start, "level")
13458 * parsers.optionalspace
13459 * (C(((parsers.linechar
13460 - (parsers.attributes
13461 * parsers.optionalspace
13462 * parsers.newline))
13463 * (parsers.linechar
13464 - parsers.lbrace)^0)^1)
13465 / strip_atx_end
13466 / parsers.parse_heading_text)
13467 * Cg(Ct(parsers.newline
13468 + (parsers.attributes
13469 * parsers.optionalspace
13470 * parsers.newline)), "attributes")
13471 * Cb("level")
13472 * Cb("attributes")
13473 / writer.heading
13474
13475 local function strip_trailing_spaces(s)
13476 return s:gsub("%s*$", "")
13477 end
13478
13479 local heading_line = (parsers.linechar
13480 - (parsers.attributes
13481 * parsers.optionalspace
13482 * parsers.newline))^1
13483 - parsers.thematic_break_lines

```

```

13484
13485 local heading_text
13486 = heading_line
13487 * ((V("Endline") / "\n")
13488 * (heading_line - parsers.heading_level))^0
13489 * parsers.newline^-1
13490
13491 local SettextHeading
13492 = parsers.freeze_trail * parsers.check_trail_no_rem
13493 * #(heading_text
13494 * (parsers.attributes
13495 * parsers.optionalspace
13496 * parsers.newline)^-1
13497 * parsers.check_minimal_indent
13498 * parsers.check_trail
13499 * parsers.heading_level)
13500 * Cs(heading_text) / strip_trailing_spaces
13501 / parsers.parse_heading_text
13502 * Cg(Ct((parsers.attributes
13503 * parsers.optionalspace
13504 * parsers.newline)^-1), "attributes")
13505 * parsers.check_minimal_indent_and_trail * parsers.heading_level
13506 * Cb("attributes")
13507 * parsers.newline
13508 * parsers.unfreeze_trail
13509 / writer.heading
13510
13511 local Heading = AtxHeading + SettextHeading
13512 self.update_rule("Heading", Heading)
13513 end
13514 }
13515 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc in-line code attribute syntax extension.

```

13516 M.extensions.inline_code_attributes = function()
13517 return {
13518 name = "built-in inline_code_attributes syntax extension",
13519 extend_writer = function()
13520 end, extend_reader = function(self)
13521 local writer = self.writer
13522
13523 local CodeWithAttributes = parsers.inticks
13524 * Ct(parsers.attributes)
13525 / writer.code

```

```

13526
13527 self.insert_pattern("Inline before Code",
13528 CodeWithAttributes,
13529 "CodeWithAttributes")
13530 end
13531 }
13532 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

13533 M.extensions.line_blocks = function()
13534 return {
13535 name = "built-in line_blocks syntax extension",
13536 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.

```

13537 function self.lineblock(lines)
13538 if not self.is_writing then return "" end
13539 local buffer = {}
13540 for i = 1, #lines - 1 do
13541 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
13542 end
13543 buffer[#buffer + 1] = lines[#lines]
13544
13545 return {"\\markdownRendererLineBlockBegin\\n"
13546 ,buffer,
13547 "\\n\\markdownRendererLineBlockEnd "}
13548 end
13549 end, extend_reader = function(self)
13550 local parsers = self.parsers
13551 local writer = self.writer
13552
13553 local LineBlock
13554 = Ct((Cs(((parsers.pipe * parsers.space) / ""
13555 * ((parsers.space)/entities.char_entity("nbsp"))^0
13556 * parsers.linechar^0 * (parsers.newline/""))
13557 * (-parsers.pipe
13558 * (parsers.space^1/" ")
13559 * parsers.linechar^1
13560 * (parsers.newline/"")
13561)^0)
13562 / self.parser_functions.parse_inlines)^1)
13563 / writer.lineblock
13564
13565 self.insert_pattern("Block after Blockquote",

```

```

13566 LineBlock, "LineBlock")
13567 end
13568 }
13569 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

13570 M.extensions.mark = function()
13571 return {
13572 name = "built-in mark syntax extension",
13573 extend_writer = function(self)

```

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

```

13574 function self.mark(s)
13575 if self.flatten_inlines then return s end
13576 return {"\\markdownRendererMark{" , s, "}"}
13577 end
13578 end, extend_reader = function(self)
13579 local parsers = self.parsers
13580 local writer = self.writer
13581
13582 local doubleequals = P("==")
13583
13584 local Mark
13585 = parsers.between(V("Inline"), doubleequals, doubleequals)
13586 / function (inlines) return writer.mark(inlines) end
13587
13588 self.add_special_character("=")
13589 self.insert_pattern("Inline before LinkAndEmph",
13590 Mark, "Mark")
13591 end
13592 }
13593 end

```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

13594 M.extensions.link_attributes = function()
13595 return {
13596 name = "built-in link_attributes syntax extension",
13597 extend_writer = function()
13598 end, extend_reader = function(self)
13599 local parsers = self.parsers
13600 local options = self.options
13601

```



The following patterns define link reference definitions with attributes.

```
13602 local define_reference_parser
13603 = (parsers.check_trail / "")
13604 * parsers.link_label
13605 * parsers.colon
13606 * parsers.spnlc * parsers.url
13607 * (parsers.spnlc_sep * parsers.title
13608 * (parsers.spnlc * Ct(parsers.attributes))
13609 * parsers.only_blank
13610 + parsers.spnlc_sep * parsers.title * parsers.only_blank
13611 + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
13612 * parsers.only_blank
13613 + Cc("") * parsers.only_blank)
13614
13615 local ReferenceWithAttributes = define_reference_parser
13616 / self.register_link
13617
13618 self.update_rule("Reference", ReferenceWithAttributes)
13619
```

The following patterns define direct and indirect links with attributes.

```
13620
13621 local LinkWithAttributesAndEmph
13622 = Ct(parsers.link_and_emph_table * Cg(Cc(true),
13623 "match_link_attributes"))
13624 / self.defer_link_and_emphasis_processing
13625
13626 self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
13627
```

The following patterns define autolinks with attributes.

```
13628 local AutoLinkUrlWithAttributes
13629 = parsers.auto_link_url
13630 * Ct(parsers.attributes)
13631 / self.auto_link_url
13632
13633 self.insert_pattern("Inline before AutoLinkUrl",
13634 AutoLinkUrlWithAttributes,
13635 "AutoLinkUrlWithAttributes")
13636
13637 local AutoLinkEmailWithAttributes
13638 = parsers.auto_link_email
13639 * Ct(parsers.attributes)
13640 / self.auto_link_email
13641
13642 self.insert_pattern("Inline before AutoLinkEmail",
13643 AutoLinkEmailWithAttributes,
13644 "AutoLinkEmailWithAttributes")
```

```

13645
13646 if options.relativeReferences then
13647
13648 local AutoLinkRelativeReferenceWithAttributes
13649 = parsers.auto_link_relative_reference
13650 * Ct(parsers.attributes)
13651 / self.auto_link_url
13652
13653 self.insert_pattern(
13654 "Inline before AutoLinkRelativeReference",
13655 AutoLinkRelativeReferenceWithAttributes,
13656 "AutoLinkRelativeReferenceWithAttributes")
13657
13658 end
13659
13660 end
13661 }
13662 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.

```

13663 M.extensions.notes = function(notes, inline_notes)
13664 assert(notes or inline_notes)
13665 return {
13666 name = "built-in notes syntax extension",
13667 extend_writer = function(self)

```

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

```

13668 function self.note(s)
13669 if self.flatten_inlines then return "" end
13670 return {"\\markdownRendererNote{",s,""}
13671 end
13672 end, extend_reader = function(self)
13673 local parsers = self.parsers
13674 local writer = self.writer
13675
13676 local rawnotes = parsers.rawnotes
13677
13678 if inline_notes then
13679 local InlineNote
13680 = parsers.circumflex
13681 * (parsers.link_label

```

```

13682 / self.parser_functions.parse_inlines_no_inline_note)
13683 / writer.note
13684
13685 self.insert_pattern("Inline after LinkAndEmph",
13686 InlineNote, "InlineNote")
13687 end
13688 if notes then
13689 local function strip_first_char(s)
13690 return s:sub(2)
13691 end
13692
13693 local RawNoteRef
13694 = #(parsers.lbracket * parsers.circumflex)
13695 * parsers.link_label / strip_first_char
13696
13697 -- like indirect_link
13698 local function lookup_note(ref)
13699 return writer.defer_call(function()
13700 local found = rawnotes[self.normalize_tag(ref)]
13701 if found then
13702 return writer.note(
13703 self.parser_functions.parse_blocks_nested(found))
13704 else
13705 local text = string.format(
13706 'Undefined note reference "%s"', ref)
13707 local more = string.format(
13708 "Look for the text `[~%s]`.", ref)
13709 return {writer.warning(text, more), "[" ,
13710 self.parser_functions.parse_inlines("^" .. ref), "]" }
13711 end
13712 end)
13713 end
13714
13715 local function register_note(ref, rawnote)
13716 local normalized_tag = self.normalize_tag(ref)
13717 if rawnotes[normalized_tag] == nil then
13718 rawnotes[normalized_tag] = rawnote
13719 return ""
13720 else
13721 local text
13722 = string.format('Multiply defined note reference "%s"',
13723 ref)
13724 local more
13725 = string.format("Look for the text `[~%s]: ...`.", ref)
13726 return writer.warning(text, more)
13727 end
13728 end

```

```

13729
13730 local NoteRef = RawNoteRef / lookup_note
13731
13732 local optionally_indented_line
13733 = parsers.check_optional_indent_and_any_trail * parsers.line
13734
13735 local blank
13736 = parsers.check_optional_blank_indent_and_any_trail
13737 * parsers.optionalspace * parsers.newline
13738
13739 local chunk
13740 = Cs(parsers.line
13741 * (optionally_indented_line - blank)^0)
13742
13743 local indented_blocks = function(bl)
13744 return Cs(bl
13745 * (blank^1 * (parsers.check_optional_indent / "")
13746 * parsers.check_code_trail
13747 * -parsers.blankline * bl)^0)
13748 end
13749
13750 local NoteBlock
13751 = parsers.check_trail_no_rem
13752 * RawNoteRef * parsers.colon
13753 * parsers.spnlc * indented_blocks(chunk)
13754 / register_note
13755
13756 self.update_rule("Reference", function(previous_pattern)
13757 if previous_pattern == nil then
13758 previous_pattern = parsers.Reference
13759 end
13760 return NoteBlock + previous_pattern
13761 end)
13762
13763 self.insert_pattern("Inline before LinkAndEmph",
13764 NoteRef, "NoteRef")
13765 end
13766
13767 self.add_special_character("^")
13768 end
13769 }
13770 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.

```

13771 M.extensions.pipe_tables = function(table_captions, table_attributes)
13772
13773 local function make_pipe_table_rectangular(rows)
13774 local num_columns = #rows[2]
13775 local rectangular_rows = {}
13776 for i = 1, #rows do
13777 local row = rows[i]
13778 local rectangular_row = {}
13779 for j = 1, num_columns do
13780 rectangular_row[j] = row[j] or ""
13781 end
13782 table.insert(rectangular_rows, rectangular_row)
13783 end
13784 return rectangular_rows
13785 end
13786
13787 local function pipe_table_row(allow_empty_first_column
13788 , nonempty_column
13789 , column_separator
13790 , column)
13791 local row_beginning
13792 if allow_empty_first_column then
13793 row_beginning = -- empty first column
13794 #(parsers.spacechar^4
13795 * column_separator)
13796 * parsers.optionalspace
13797 * column
13798 * parsers.optionalspace
13799 -- non-empty first column
13800 + parsers.nonindentSPACE
13801 * nonempty_column^-1
13802 * parsers.optionalspace
13803 else
13804 row_beginning = parsers.nonindentSPACE
13805 * nonempty_column^-1
13806 * parsers.optionalspace
13807 end
13808
13809 return Ct(row_beginning
13810 * (-- single column with no leading pipes
13811 #(column_separator
13812 * parsers.optionalspace

```

```

13813 * parsers.newline)
13814 * column_separator
13815 * parsers.optionalspace
13816 -- single column with leading pipes or
13817 -- more than a single column
13818 + (column_separator
13819 * parsers.optionalspace
13820 * column
13821 * parsers.optionalspace)^1
13822 * (column_separator
13823 * parsers.optionalspace)^-1))
13824 end
13825
13826 return {
13827 name = "built-in pipe_tables syntax extension",
13828 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.

```

13829 function self.table(rows, caption, attributes)
13830 if not self.is_writing then return "" end
13831 local buffer = {}
13832 if attributes ~= nil then
13833 table.insert(buffer,
13834 "\\markdownRendererTableAttributeContextBegin\n")
13835 table.insert(buffer, self.attributes(attributes))
13836 end
13837 table.insert(buffer,
13838 {"\\markdownRendererTable{",
13839 caption or "", "}{" , #rows - 1, "}{" ,
13840 #rows[1], "}"})
13841 local temp = rows[2] -- put alignments on the first row
13842 rows[2] = rows[1]
13843 rows[1] = temp
13844 for i, row in ipairs(rows) do
13845 table.insert(buffer, "{")
13846 for _, column in ipairs(row) do
13847 if i > 1 then -- do not use braces for alignments
13848 table.insert(buffer, "{")
13849 end
13850 table.insert(buffer, column)
13851 if i > 1 then
13852 table.insert(buffer, "}")
13853 end
13854 end
13855 table.insert(buffer, "}")
13856 end

```

```

13857 if attributes ~= nil then
13858 table.insert(buffer,
13859 "\\markdownRendererTableAttributeContextEnd{}")
13860 end
13861 return buffer
13862 end
13863 end, extend_reader = function(self)
13864 local parsers = self.parsers
13865 local writer = self.writer
13866
13867 local table_hline_separator = parsers.pipe + parsers.plus
13868
13869 local table_hline_column = (parsers.dash
13870 - #(parsers.dash
13871 * (parsers.spacechar
13872 + table_hline_separator
13873 + parsers.newline)))^1
13874 * (parsers.colon * Cc("r")
13875 + parsers.dash * Cc("d"))
13876 + parsers.colon
13877 * (parsers.dash
13878 - #(parsers.dash
13879 * (parsers.spacechar
13880 + table_hline_separator
13881 + parsers.newline)))^1
13882 * (parsers.colon * Cc("c")
13883 + parsers.dash * Cc("l"))
13884
13885 local table_hline = pipe_table_row(false
13886 , table_hline_column
13887 , table_hline_separator
13888 , table_hline_column)
13889
13890 local table_caption_beginning
13891 = (parsers.check_minimal_blank_indent_and_any_trail_no_rem
13892 * parsers.optionalspace * parsers.newline)^0
13893 * parsers.check_minimal_indent_and_trail
13894 * (P("Table")^-1 * parsers.colon)
13895 * parsers.optionalspace
13896
13897 local function strip_trailing_spaces(s)
13898 return s:gsub("%s*$", "")
13899 end
13900
13901 local table_row
13902 = pipe_table_row(true
13903 , (C((parsers.linechar - parsers.pipe)^1)

```

```

13904 / strip_trailing_spaces
13905 / self.parser_functions.parse_inlines)
13906 , parsers.pipe
13907 , (C((parsers.linechar - parsers.pipe)^0)
13908 / strip_trailing_spaces
13909 / self.parser_functions.parse_inlines))
13910
13911 local table_caption
13912 if table_captions then
13913 table_caption = #table_caption_beginning
13914 * table_caption_beginning
13915 if table_attributes then
13916 table_caption = table_caption
13917 * (C((((parsers.linechar
13918 - (parsers.attributes
13919 * parsers.optionalspace
13920 * parsers.newline
13921 * -(parsers.optionalspace
13922 * parsers.linechar)))
13923 + (parsers.newline
13924 * #(parsers.optionalspace
13925 * parsers.linechar)
13926 * C(parsers.optionalspace)
13927 / writer.space))
13928 * (parsers.linechar
13929 - parsers.lbrace)^0)^1)
13930 / strip_trailing_spaces
13931 / self.parser_functions.parse_inlines)
13932 * (parsers.newline
13933 + (Ct(parsers.attributes)
13934 * parsers.optionalspace
13935 * parsers.newline))
13936 else
13937 table_caption = table_caption
13938 * C((parsers.linechar
13939 + (parsers.newline
13940 * #(parsers.optionalspace
13941 * parsers.linechar)
13942 * C(parsers.optionalspace)
13943 / writer.space))^1)
13944 / self.parser_functions.parse_inlines
13945 * parsers.newline
13946 end
13947 else
13948 table_caption = parsers.fail
13949 end
13950

```



```

13951 local PipeTable
13952 = Ct(table_row * parsers.newline
13953 * (parsers.check_minimal_indent_and_trail / {})
13954 * table_hline * parsers.newline
13955 * ((parsers.check_minimal_indent / {})
13956 * table_row * parsers.newline)^0)
13957 / make_pipe_table_rectangular
13958 * table_caption^-1
13959 / writer.table
13960
13961 self.insert_pattern("Block after Blockquote",
13962 PipeTable, "PipeTable")
13963 end
13964 }
13965 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

13966 M.extensions.raw_inline = function()
13967 return {
13968 name = "built-in raw_inline syntax extension",
13969 extend_writer = function(self)
13970 local options = self.options
13971

```

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

```

13972 function self.rawInline(s, attr)
13973 if not self.is_writing then return "" end
13974 if self.flatten_inlines then return s end
13975 local name = util.cache_verbatim(options.cacheDir, s)
13976 return {"\\markdownRendererInputRawInline{",
13977 name,"}{" , self.string(attr),"}" }
13978 end
13979 end, extend_reader = function(self)
13980 local writer = self.writer
13981
13982 local RawInline = parsers.inticks
13983 * parsers.raw_attribute
13984 / writer.rawInline
13985
13986 self.insert_pattern("Inline before Code",
13987 RawInline, "RawInline")
13988 end
13989 }
13990 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
13991 M.extensions.strike_through = function()
13992 return {
13993 name = "built-in strike_through syntax extension",
13994 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.

```
13995 function self.strike_through(s)
13996 if self.flatten_inlines then return s end
13997 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
13998 end
13999 end, extend_reader = function(self)
14000 local parsers = self.parsers
14001 local writer = self.writer
14002
14003 local StrikeThrough = (
14004 parsers.between(parsers.Inline, parsers.doubletildes,
14005 parsers.doubletildes)
14006) / writer.strike_through
14007
14008 self.insert_pattern("Inline after LinkAndEmph",
14009 StrikeThrough, "StrikeThrough")
14010
14011 self.add_special_character("~")
14012 end
14013 }
14014 end
```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
14015 M.extensions.subscripts = function()
14016 return {
14017 name = "built-in subscripts syntax extension",
14018 extend_writer = function(self)
```

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

```
14019 function self.subscript(s, parsed)
14020 if self.flatten_inlines then return s end
14021 return {"\\markdownRendererSubscript{" ,parsed,"}"}
14022 end
14023 end, extend_reader = function(self)
```

```

14024 local parsers = self.parsers
14025 local writer = self.writer
14026
14027 local Subscript = (
14028 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
14029) / function(s)
14030 return writer.subscript(
14031 s, self.parser_functions.parse_inlines_identity(s[1])
14032)
14033 end
14034
14035 self.insert_pattern("Inline after LinkAndEmph",
14036 Subscript, "Subscript")
14037
14038 self.add_special_character("~")
14039 end
14040 }
14041 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

14042 M.extensions.superscripts = function()
14043 return {
14044 name = "built-in superscripts syntax extension",
14045 extend_writer = function(self)

```

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

```

14046 function self.superscript(s, parsed)
14047 if self.flatten_inlines then return s end
14048 return {"\\markdownRendererSuperscript{" ,parsed,""} }
14049 end
14050 end, extend_reader = function(self)
14051 local parsers = self.parsers
14052 local writer = self.writer
14053
14054 local Superscript = (
14055 parsers.between(parsers.Str, parsers.circumflex,
14056 parsers.circumflex)
14057) / function(s)
14058 return writer.superscript(
14059 s, self.parser_functions.parse_inlines_identity(s[1])
14060)
14061 end
14062
14063 self.insert_pattern("Inline after LinkAndEmph",

```

```

14064 Superscript, "Superscript")
14065
14066 self.add_special_character("^")
14067 end
14068 }
14069 end

```

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

14070 M.extensions.tex_math = function(tex_math_dollars,
14071 tex_math_single_backslash,
14072 tex_math_double_backslash)
14073 return {
14074 name = "built-in tex_math syntax extension",
14075 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.

```

14076 function self.display_math(s, parsed)
14077 if self.flatten_inlines then return s end
14078 return {"\\markdownRendererDisplayMath{",parsed,""}
14079 end

```

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

```

14080 function self.inline_math(s, parsed)
14081 if self.flatten_inlines then return s end
14082 return {"\\markdownRendererInlineMath{",parsed,""}
14083 end
14084 end, extend_reader = function(self)
14085 local parsers = self.parsers
14086 local writer = self.writer
14087
14088 local function between(p, starter, ender)
14089 return (starter * Cs(p * (p - ender)^0) * ender)
14090 end
14091
14092 local function strip_preceding_spaces(str)
14093 return str:gsub("^%s*(.-)$", "%1")
14094 end
14095
14096 local allowed_before_closing
14097 = B(parsers.backslash * parsers.any
14098 + parsers.any * (parsers.any - parsers.backslash))
14099
14100 local allowed_before_closing_no_space

```

```

14101 = B(parsers.backslash * parsers.any
14102 + parsers.any * (parsers.nonspacechar - parsers.backslash))
14103

```

The following patterns implement the Pandoc dollar math syntax extension.

```

14104 local dollar_math_content
14105 = (parsers.newline * (parsers.check_optional_indent / "")
14106 + parsers.backslash~1
14107 * parsers.linechar)
14108 - parsers.blankline~2
14109 - parsers.dollar
14110
14111 local inline_math_opening_dollars = parsers.dollar
14112 * #(parsers.nonspacechar)
14113
14114 local inline_math_closing_dollars
14115 = allowed_before_closing_no_space
14116 * parsers.dollar
14117 * -#(parsers.digit)
14118
14119 local inline_math_dollars = between(Cs(dollar_math_content),
14120 inline_math_opening_dollars,
14121 inline_math_closing_dollars)
14122
14123 local display_math_opening_dollars = parsers.dollar
14124 * parsers.dollar
14125
14126 local display_math_closing_dollars = parsers.dollar
14127 * parsers.dollar
14128
14129 local display_math_dollars = between(Cs(dollar_math_content),
14130 display_math_opening_dollars,
14131 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

14132 local backslash_math_content
14133 = (parsers.newline * (parsers.check_optional_indent / "")
14134 + parsers.linechar)
14135 - parsers.blankline~2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

14136 local inline_math_opening_double = parsers.backslash
14137 * parsers.backslash
14138 * parsers.lparent
14139
14140 local inline_math_closing_double = allowed_before_closing

```

```

14141 * parsers.spacechar~0
14142 * parsers.backslash
14143 * parsers.backslash
14144 * parsers.rparent
14145
14146 local inline_math_double = between(Cs(backslash_math_content),
14147 inline_math_opening_double,
14148 inline_math_closing_double)
14149 / strip_preceding_whitespaces
14150
14151 local display_math_opening_double = parsers.backslash
14152 * parsers.backslash
14153 * parsers.lbracket
14154
14155 local display_math_closing_double = allowed_before_closing
14156 * parsers.spacechar~0
14157 * parsers.backslash
14158 * parsers.backslash
14159 * parsers.rbracket
14160
14161 local display_math_double = between(Cs(backslash_math_content),
14162 display_math_opening_double,
14163 display_math_closing_double)
14164 / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

14165 local inline_math_opening_single = parsers.backslash
14166 * parsers.lparent
14167
14168 local inline_math_closing_single = allowed_before_closing
14169 * parsers.spacechar~0
14170 * parsers.backslash
14171 * parsers.rparent
14172
14173 local inline_math_single = between(Cs(backslash_math_content),
14174 inline_math_opening_single,
14175 inline_math_closing_single)
14176 / strip_preceding_whitespaces
14177
14178 local display_math_opening_single = parsers.backslash
14179 * parsers.lbracket
14180
14181 local display_math_closing_single = allowed_before_closing
14182 * parsers.spacechar~0
14183 * parsers.backslash
14184 * parsers.rbracket
14185
14186 local display_math_single = between(Cs(backslash_math_content),

```

```

14187 display_math_opening_single,
14188 display_math_closing_single)
14189 / strip_preceding_whitespace
14190
14191 local display_math = parsers.fail
14192
14193 local inline_math = parsers.fail
14194
14195 if tex_math_dollars then
14196 display_math = display_math + display_math_dollars
14197 inline_math = inline_math + inline_math_dollars
14198 end
14199
14200 if tex_math_double_backslash then
14201 display_math = display_math + display_math_double
14202 inline_math = inline_math + inline_math_double
14203 end
14204
14205 if tex_math_single_backslash then
14206 display_math = display_math + display_math_single
14207 inline_math = inline_math + inline_math_single
14208 end
14209
14210 local TexMath = display_math
14211 / function(s)
14212 return writer.display_math(
14213 s, self.parser_functions.parse_inlines_math(s)
14214)
14215 end
14216 + inline_math
14217 / function(s)
14218 return writer.inline_math(
14219 s, self.parser_functions.parse_inlines_math(s)
14220)
14221 end
14222
14223 self.insert_pattern("Inline after LinkAndEmph",
14224 TexMath, "TexMath")
14225
14226 if tex_math_dollars then
14227 self.add_special_character("$")
14228 end
14229
14230 if tex_math_single_backslash or tex_math_double_backslash then
14231 self.add_special_character("\\")
14232 self.add_special_character("[")
14233 self.add_special_character("]")

```

```

14234 self.add_special_character("")
14235 self.add_special_character("")
14236 end
14237 end
14238 }
14239 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.

```

14240 M.extensions.jekyll_data = function(expect_jekyll_data,
14241 ensure_jekyll_data)
14242 return {
14243 name = "built-in jekyll_data syntax extension",
14244 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.

```

14245 function self.jekyllData(d, t, p)
14246 if not self.is_writing then return "" end
14247
14248 local buf = {}
14249
14250 local keys = {}
14251 for k, _ in pairs(d) do
14252 table.insert(keys, k)
14253 end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

14254 table.sort(keys, function(first, second)
14255 if type(first) ~= type(second) then
14256 return type(first) < type(second)
14257 else
14258 return first < second
14259 end
14260 end)
14261

```



```

14262 if not p then
14263 table.insert(buf, "\\markdownRendererJekyllDataBegin")
14264 end
14265
14266 local is_sequence = false
14267 if #d > 0 and #d == #keys then
14268 for i=1, #d do
14269 if d[i] == nil then
14270 goto not_a_sequence
14271 end
14272 end
14273 is_sequence = true
14274 end
14275 ::not_a_sequence::
14276
14277 if is_sequence then
14278 table.insert(buf,
14279 "\\markdownRendererJekyllDataSequenceBegin{")
14280 table.insert(buf, self.identifier(p or "null"))
14281 table.insert(buf, "}{")
14282 table.insert(buf, #keys)
14283 table.insert(buf, "}")
14284 else
14285 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
14286 table.insert(buf, self.identifier(p or "null"))
14287 table.insert(buf, "}{")
14288 table.insert(buf, #keys)
14289 table.insert(buf, "}")
14290 end
14291
14292 for _, k in ipairs(keys) do
14293 local v = d[k]
14294 local typ = type(v)
14295 k = tostring(k or "null")
14296 if typ == "table" and next(v) ~= nil then
14297 table.insert(
14298 buf,
14299 self.jekyllData(v, t, k)
14300)
14301 else
14302 k = self.identifier(k)
14303 v = tostring(v)
14304 if typ == "boolean" then
14305 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
14306 table.insert(buf, k)
14307 table.insert(buf, "}{")
14308 table.insert(buf, v)

```

```

14309 table.insert(buf, "}")
14310 elseif typ == "number" then
14311 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
14312 table.insert(buf, k)
14313 table.insert(buf, "{")
14314 table.insert(buf, v)
14315 table.insert(buf, "}")
14316 elseif typ == "string" then
14317 table.insert(buf,
14318 "\\markdownRendererJekyllDataProgrammaticString{")
14319 table.insert(buf, k)
14320 table.insert(buf, "{")
14321 table.insert(buf, self.identifier(v))
14322 table.insert(buf, "}")
14323 table.insert(buf,
14324 "\\markdownRendererJekyllDataTypographicString{")
14325 table.insert(buf, k)
14326 table.insert(buf, "{")
14327 table.insert(buf, t(v))
14328 table.insert(buf, "}")
14329 elseif typ == "table" then
14330 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
14331 table.insert(buf, k)
14332 table.insert(buf, "}")
14333 else
14334 local error = self.error(format(
14335 "Unexpected type %s for value of "
14336 .. "YAML key %s.", typ, k))
14337 table.insert(buf, error)
14338 end
14339 end
14340 end
14341
14342 if is_sequence then
14343 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
14344 else
14345 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
14346 end
14347
14348 if not p then
14349 table.insert(buf, "\\markdownRendererJekyllDataEnd")
14350 end
14351
14352 return buf
14353 end
14354 end, extend_reader = function(self)
14355 local parsers = self.parsers

```

```

14356 local writer = self.writer
14357
14358 local JekyllData
14359 = Cmt(C((parsers.line - P("----") - P("..."))^0)
14360 , function(s, i, text) -- luacheck: ignore s i
14361 local data
14362 local ran_ok, _ = pcall(function()
14363 local tinyyaml = require("tinyyaml")
14364 data = tinyyaml.parse(text, {timestamps=false})
14365 end)
14366 if ran_ok and data ~= nil then
14367 return true, writer.jekyllData(data, function(s)
14368 return self.parser_functions.parse_blocks_nested(s)
14369 end, nil)
14370 else
14371 return false
14372 end
14373 end
14374)
14375
14376 local UnexpectedJekyllData
14377 = P("----")
14378 * parsers.blankline / 0
14379 -- if followed by blank, it's thematic break
14380 * #(-parsers.blankline)
14381 * JekyllData
14382 * (P("----") + P("..."))
14383
14384 local ExpectedJekyllData
14385 = (P("----")
14386 * parsers.blankline / 0
14387 -- if followed by blank, it's thematic break
14388 * #(-parsers.blankline)
14389)^-1
14390 * JekyllData
14391 * (P("----") + P("..."))^-1
14392
14393 if ensure_jekyll_data then
14394 ExpectedJekyllData = ExpectedJekyllData
14395 * parsers.eof
14396 else
14397 ExpectedJekyllData = (ExpectedJekyllData
14398 * (V("Blank")^0 / writer.interblocksep)
14399)^-1
14400 end
14401
14402 self.insert_pattern("Block before Blockquote",

```

```

14403 UnexpectedJekyllData, "UnexpectedJekyllData")
14404 if expect_jekyll_data then
14405 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
14406 end
14407 end
14408 }
14409 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`.

```

14410 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```

14411 options = options or {}
14412 setmetatable(options, { __index = function (_, key)
14413 return defaultOptions[key] end })
14414 if options.experimental then
14415 setmetatable(options, { __index = function (_, key)
14416 return experimentalOptions[key] end })
14417 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.

```

14418 local parser_convert = nil
14419 return function(input, include_flat_output)
14420 local function convert(input)
14421 if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

14422 local parser = require("markdown-parser")
14423 if metadata.version ~= parser.metadata.version then
14424 warn("markdown.lua " .. metadata.version .. " used with " ..
14425 "markdown-parser.lua " .. parser.metadata.version .. ".")
14426 end
14427 parser_convert = parser.new(options)
14428 end
14429 return parser_convert(input)
14430 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain T<sub>E</sub>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).

```

14431 local raw_output, flat_output
14432 if options.eagerCache or options.finalizeCache then
14433 local salt = util.salt(options)
14434 local name, result = util.cache(options.cacheDir, input, salt,
14435 convert, ".md.tex")
14436 raw_output = [[\input{}} .. name .. [{}\relax]]
14437 flat_output = function()
14438 if result == nil then
14439 local input_file = assert(io.open(name, "r"),
14440 [[Could not open file "]] .. name .. [[" for reading]])
14441 result = assert(input_file:read("*a"))
14442 assert(input_file:close())
14443 end
14444 return result
14445 end

```

Otherwise, return the result of the conversion directly.

```

14446 else
14447 raw_output = convert(input)
14448 flat_output = function()
14449 return raw_output
14450 end
14451 end

```

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

```

14452 if options.finalizeCache then
14453 local file, mode
14454 if options.frozenCacheCounter > 0 then
14455 mode = "a"
14456 else
14457 mode = "w"
14458 end
14459 file = assert(io.open(options.frozenCacheFileName, mode),
14460 [[Could not open file "]] .. options.frozenCacheFileName
14461 .. [[" for writing]])
14462 assert(file:write(
14463 [[\expandafter\global\expandafter\def\csname]]
14464 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
14465 .. [[\endcsname{}} .. raw_output .. [{}]] .. "\n"))
14466 assert(file:close())
14467 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.

```

14468 if include_flat_output then
14469 return raw_output, flat_output
14470 else
14471 return raw_output
14472 end
14473 end
14474 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output. See Section 2.1.1.

```

14475 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` and `experimentalOptions` tables.

```

14476 options = options or {}
14477 setmetatable(options, { __index = function (_, key)
14478 return defaultOptions[key] end })
14479 if options.experimental then
14480 setmetatable(options, { __index = function (_, key)
14481 return experimentalOptions[key] end })
14482 end

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

14483 if options.singletonCache and singletonCache.convert then
14484 for k, v in pairs(defaultOptions) do
14485 if type(v) == "table" then
14486 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
14487 if singletonCache.options[k][i] ~= options[k][i] then
14488 goto miss
14489 end
14490 end

```

The `cacheDir` option is disregarded.

```

14491 elseif k ~= "cacheDir"
14492 and singletonCache.options[k] ~= options[k] then
14493 goto miss
14494 end
14495 end
14496 return singletonCache.convert
14497 end
14498 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

14499 local extensions = {}
14500

```

```

14501 if #options.acronyms > 0 then
14502 local acronyms_extension = M.extensions.acronyms(options.acronyms)
14503 table.insert(extensions, acronyms_extension)
14504 end
14505
14506 if options.bracketedSpans then
14507 local bracketed_spans_extension = M.extensions.bracketed_spans()
14508 table.insert(extensions, bracketed_spans_extension)
14509 end
14510
14511 if options.contentBlocks then
14512 local content_blocks_extension = M.extensions.content_blocks(
14513 options.contentBlocksLanguageMap)
14514 table.insert(extensions, content_blocks_extension)
14515 end
14516
14517 if options.definitionLists then
14518 local definition_lists_extension = M.extensions.definition_lists(
14519 options.tightLists)
14520 table.insert(extensions, definition_lists_extension)
14521 end
14522
14523 if options.fencedCode then
14524 local fenced_code_extension = M.extensions.fenced_code(
14525 options.blankBeforeCodeFence,
14526 options.fencedCodeAttributes,
14527 options.rawAttribute)
14528 table.insert(extensions, fenced_code_extension)
14529 end
14530
14531 if options.fencedDivs then
14532 local fenced_div_extension = M.extensions.fenced_divs(
14533 options.blankBeforeDivFence)
14534 table.insert(extensions, fenced_div_extension)
14535 end
14536
14537 if options.headerAttributes then
14538 local header_attributes_extension = M.extensions.header_attributes()
14539 table.insert(extensions, header_attributes_extension)
14540 end
14541
14542 if options.inlineCodeAttributes then
14543 local inline_code_attributes_extension =
14544 M.extensions.inline_code_attributes()
14545 table.insert(extensions, inline_code_attributes_extension)
14546 end
14547

```

```

14548 if options.jekyllData then
14549 local jekyll_data_extension = M.extensions.jekyll_data(
14550 options.expectJekyllData, options.ensureJekyllData)
14551 table.insert(extensions, jekyll_data_extension)
14552 end
14553
14554 if options.linkAttributes then
14555 local link_attributes_extension =
14556 M.extensions.link_attributes()
14557 table.insert(extensions, link_attributes_extension)
14558 end
14559
14560 if options.lineBlocks then
14561 local line_block_extension = M.extensions.line_blocks()
14562 table.insert(extensions, line_block_extension)
14563 end
14564
14565 if options.mark then
14566 local mark_extension = M.extensions.mark()
14567 table.insert(extensions, mark_extension)
14568 end
14569
14570 if options.pipeTables then
14571 local pipe_tables_extension = M.extensions.pipe_tables(
14572 options.tableCaptions, options.tableAttributes)
14573 table.insert(extensions, pipe_tables_extension)
14574 end
14575
14576 if options.rawAttribute then
14577 local raw_inline_extension = M.extensions.raw_inline()
14578 table.insert(extensions, raw_inline_extension)
14579 end
14580
14581 if options.strikeThrough then
14582 local strike_through_extension = M.extensions.strike_through()
14583 table.insert(extensions, strike_through_extension)
14584 end
14585
14586 if options.subscripts then
14587 local subscript_extension = M.extensions.subscripts()
14588 table.insert(extensions, subscript_extension)
14589 end
14590
14591 if options.superscripts then
14592 local superscript_extension = M.extensions.superscripts()
14593 table.insert(extensions, superscript_extension)
14594 end

```



```

14595
14596 if options.texMathDollars or
14597 options.texMathSingleBackslash or
14598 options.texMathDoubleBackslash then
14599 local tex_math_extension = M.extensions.tex_math(
14600 options.texMathDollars,
14601 options.texMathSingleBackslash,
14602 options.texMathDoubleBackslash)
14603 table.insert(extensions, tex_math_extension)
14604 end
14605
14606 if options.notes or options.inlineNotes then
14607 local notes_extension = M.extensions.notes(
14608 options.notes, options.inlineNotes)
14609 table.insert(extensions, notes_extension)
14610 end
14611
14612 if options.citations then
14613 local citations_extension
14614 = M.extensions.citations(options.citationNbsps)
14615 table.insert(extensions, citations_extension)
14616 end
14617
14618 if options.fancyLists then
14619 local fancy_lists_extension = M.extensions.fancy_lists()
14620 table.insert(extensions, fancy_lists_extension)
14621 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

14622 for _, user_extension_filename in ipairs(options.extensions) do
14623 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

14624 local pathname = assert(util.find_file(filename),
14625 [[Could not locate user-defined syntax extension "]]
14626 .. filename)
14627 local input_file = assert(io.open(pathname, "r"),
14628 [[Could not open user-defined syntax extension "]]
14629 .. pathname .. [[" for reading]])
14630 local input = assert(input_file:read("*a"))
14631 assert(input_file:close())
14632 local user_extension, err = load([[
14633 local sandbox = {}
14634 setmetatable(sandbox, {__index = _G})
14635 _ENV = sandbox
14636]] .. input)()
14637 assert(user_extension,
14638 [[Failed to compile user-defined syntax extension "]]

```

```
14639 .. pathname .. [": "] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
14640 assert(user_extension.api_version ~= nil,
14641 [[User-defined syntax extension "] .. pathname
14642 .. [" does not specify mandatory field "api_version"]])
14643 assert(type(user_extension.api_version) == "number",
14644 [[User-defined syntax extension "] .. pathname
14645 .. [" specifies field "api_version" of type "]
14646 .. type(user_extension.api_version)
14647 .. [" but "number" was expected]])
14648 assert(user_extension.api_version > 0
14649 and user_extension.api_version
14650 <= metadata.user_extension_api_version,
14651 [[User-defined syntax extension "] .. pathname
14652 .. [" uses syntax extension API version "]
14653 .. user_extension.api_version .. [" but markdown.lua "]
14654 .. metadata.version .. [" uses API version "]
14655 .. metadata.user_extension_api_version
14656 .. [", which is incompatible"]])
14657
14658 assert(user_extension.grammar_version ~= nil,
14659 [[User-defined syntax extension "] .. pathname
14660 .. [" does not specify mandatory field "grammar_version"]])
14661 assert(type(user_extension.grammar_version) == "number",
14662 [[User-defined syntax extension "] .. pathname
14663 .. [" specifies field "grammar_version" of type "]
14664 .. type(user_extension.grammar_version)
14665 .. [" but "number" was expected]])
14666 assert(user_extension.grammar_version == metadata.grammar_version,
14667 [[User-defined syntax extension "] .. pathname
14668 .. [" uses grammar version "]
14669 .. user_extension.grammar_version
14670 .. [" but markdown.lua "] .. metadata.version
14671 .. [" uses grammar version "] .. metadata.grammar_version
14672 .. [", which is incompatible"]])
14673
14674 assert(user_extension.finalize_grammar ~= nil,
14675 [[User-defined syntax extension "] .. pathname
14676 .. [" does not specify mandatory "finalize_grammar" field]])
14677 assert(type(user_extension.finalize_grammar) == "function",
14678 [[User-defined syntax extension "] .. pathname
14679 .. [" specifies field "finalize_grammar" of type "]
14680 .. type(user_extension.finalize_grammar)
14681 .. [" 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.)

```

14682 local extension = {
14683 name = [[user-defined "]] .. pathname .. [[" syntax extension]],
14684 extend_reader = user_extension.finalize_grammar,
14685 extend_writer = function() end,
14686 }
14687 return extension
14688 end)(user_extension_filename)
14689 table.insert(extensions, user_extension)
14690 end

```

Produce a conversion function from markdown to plain T<sub>E</sub>X.

```

14691 local writer = M.writer.new(options)
14692 local reader = M.reader.new(writer, options)
14693 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`.

```

14694 collectgarbage("collect")

```

Update the singleton cache.

```

14695 if options.singletonCache then
14696 local singletonCacheOptions = {}
14697 for k, v in pairs(options) do
14698 singletonCacheOptions[k] = v
14699 end
14700 setmetatable(singletonCacheOptions,
14701 { __index = function (_, key)
14702 return defaultOptions[key] end })
14703 singletonCache.options = singletonCacheOptions
14704 singletonCache.convert = convert
14705 end

```

Return the conversion function from markdown to plain T<sub>E</sub>X.

```

14706 return convert
14707 end
14708 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.

```

14709
14710 local input
14711 if input_filename then
14712 local input_file = assert(io.open(input_filename, "r"),
14713 [[Could not open file "]] .. input_filename .. [[" for reading]])
14714 input = assert(input_file:read("*a"))
14715 assert(input_file:close())

```

```

14716 else
14717 input = assert(io.read("*a"))
14718 end
14719

```

First, ensure that the `options.cacheDir` directory exists.

```

14720 local lfs = require("lfs")
14721 if options.cacheDir and not lfs.isdir(options.cacheDir) then
14722 assert(lfs.mkdir(options["cacheDir"]))
14723 end

```

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

```

14724 local kpse
14725 (function()
14726 local should_initialize = package.loaded.kpse == nil
14727 or tex.initialize ~= nil
14728 kpse = require("kpse")
14729 if should_initialize then
14730 kpse.set_program_name("luatex")
14731 end
14732 end)()
14733 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.

```

14734 if metadata.version ~= md.metadata.version then
14735 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
14736 "markdown.lua " .. md.metadata.version .. ".")
14737 end
14738
14739 local convert = md.new(options)
14740 local raw_output, flat_output = convert(input, true)
14741 local output
14742 if flat_output == nil then
14743 if options.eagerCache then
14744 warn("markdown.lua has not produced flat output, so I am using " ..
14745 "backwards-compatible raw output instead. This may cause " ..
14746 "the conversion result to be hidden behind "\\input".')
14747 end
14748 output = raw_output
14749 else
14750 output = flat_output()
14751 end
14752
14753 if output_filename then
14754 local output_file = assert(io.open(output_filename, "w"),
14755 [[Could not open file]] .. output_filename .. [[for writing]])

```

```

14756 assert(output_file:write(output))
14757 assert(output_file:close())
14758 else
14759 assert(io.write(output))
14760 end

```

Remove the `options.cacheDir` directory if it is empty.

```

14761 if options.cacheDir then
14762 lfs.rmdir(options.cacheDir)
14763 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

```

14764 \ExplSyntaxOn
14765 \cs_if_free:NT
14766 \markdownInfo
14767 {
14768 \cs_new_protected:Npn
14769 \markdownInfo #1
14770 {
14771 \msg_info:nne
14772 { markdown }
14773 { generic-message }
14774 { #1 }
14775 }
14776 }
14777 \cs_if_free:NT
14778 \markdownWarning
14779 {
14780 \cs_new_protected:Npn
14781 \markdownWarning #1
14782 {
14783 \msg_warning:nne
14784 { markdown }
14785 { generic-message }
14786 { #1 }
14787 }
14788 }
14789 \cs_if_free:NT
14790 \markdownError

```

```

14791 {
14792 \cs_new_protected:Npn
14793 \markdownError #1 #2
14794 {
14795 \msg_error:nnee
14796 { markdown }
14797 { generic-message-with-help-text }
14798 { #1 }
14799 { #2 }
14800 }
14801 }
14802 \msg_new:nnn
14803 { markdown }
14804 { generic-message }
14805 { #1 }
14806 \msg_new:nnnn
14807 { markdown }
14808 { generic-message-with-help-text }
14809 { #1 }
14810 { #2 }
14811 \cs_generate_variant:Nn
14812 \msg_info:nnn
14813 { nne }
14814 \cs_generate_variant:Nn
14815 \msg_warning:nnn
14816 { nne }
14817 \cs_generate_variant:Nn
14818 \msg_error:nnnn
14819 { nnee }
14820 \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.

```

14821 \ExplSyntaxOn
14822 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
14823 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
14824 \cs_new_protected:Nn
14825 \@@_plain_tex_load_theme:nnn
14826 {
14827 \prop_get:NnNTF
14828 \g_@@_plain_tex_loaded_themes_linenos_prop
14829 { #1 }
14830 \l_tmpa_tl
14831 {

```

```

14832 \prop_get:NnN
14833 \g_@@_plain_tex_loaded_themes_versions_prop
14834 { #1 }
14835 \l_tmpb_tl
14836 \str_if_eq:nVTF
14837 { #2 }
14838 \l_tmpb_tl
14839 {
14840 \msg_warning:nnnVn
14841 { markdown }
14842 { repeatedly-loaded-plain-tex-theme }
14843 { #1 }
14844 \l_tmpa_tl
14845 { #2 }
14846 }
14847 {
14848 \msg_error:nnnnVV
14849 { markdown }
14850 { different-versions-of-plain-tex-theme }
14851 { #1 }
14852 { #2 }
14853 \l_tmpb_tl
14854 \l_tmpa_tl
14855 }
14856 }
14857 {
14858 \prop_gput:Nnx
14859 \g_@@_plain_tex_loaded_themes_linenos_prop
14860 { #1 }
14861 { \tex_the:D \tex_inputlineno:D } % noqa: W200
14862 \prop_gput:Nnn
14863 \g_@@_plain_tex_loaded_themes_versions_prop
14864 { #1 }
14865 { #2 }

```

Load built-in plain  $\text{\TeX}$  themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

14866 \prop_if_in:NnTF
14867 \g_@@_plain_tex_built_in_themes_prop
14868 { #1 }
14869 {
14870 \msg_info:nnnn
14871 { markdown }
14872 { loading-built-in-plain-tex-theme }
14873 { #1 }
14874 { #2 }
14875 \prop_item:Nn

```

```

14876 \g_@@_plain_tex_built_in_themes_prop
14877 { #1 }
14878 }
14879 {
14880 \msg_info:nnnn
14881 { markdown }
14882 { loading-plain-tex-theme }
14883 { #1 }
14884 { #2 }
14885 \file_input:n
14886 { markdown theme #3 }
14887 }
14888 }
14889 }
14890 \msg_new:nnn
14891 { markdown }
14892 { loading-plain-tex-theme }
14893 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
14894 \msg_new:nnn
14895 { markdown }
14896 { loading-built-in-plain-tex-theme }
14897 { Loading~version~#2~of~built-in~plain~TeX~Markdown~theme~#1 }
14898 \msg_new:nnn
14899 { markdown }
14900 { repeatedly-loaded-plain-tex-theme }
14901 {
14902 Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
14903 loaded~on~line~#2,~not~loading~it~again
14904 }
14905 \msg_new:nnn
14906 { markdown }
14907 { different-versions-of-plain-tex-theme }
14908 {
14909 Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
14910 but~version~#3~has~already~been~loaded~on~line~#4
14911 }
14912 \cs_generate_variant:Nn
14913 \prop_gput:Nnn
14914 { Nnx }
14915 \cs_gset_eq:NN
14916 \@@_load_theme:nnn
14917 \@@_plain_tex_load_theme:nnn
14918 \cs_generate_variant:Nn
14919 \@@_load_theme:nnn
14920 { VeV }
14921 \cs_generate_variant:Nn
14922 \msg_error:nnnnnn

```



```

14923 { nnnnVV }
14924 \cs_generate_variant:Nn
14925 \msg_warning:nnnnn
14926 { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

14927 \cs_new_protected:Npn
14928 \markdownLoadPlainTeXTheme
14929 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

14930 \tl_set:NV
14931 \l_tmpa_tl
14932 \g_@@_current_theme_tl
14933 \tl_reverse:N
14934 \l_tmpa_tl
14935 \tl_set:Ne
14936 \l_tmpb_tl
14937 {
14938 \tl_tail:V
14939 \l_tmpa_tl
14940 }
14941 \tl_reverse:N
14942 \l_tmpb_tl

```

Next, we munge the theme name.

```

14943 \str_set:NV
14944 \l_tmpa_str
14945 \l_tmpb_tl
14946 \str_replace_all:Nnn
14947 \l_tmpa_str
14948 { / }
14949 { _ }

```

Finally, we load the plain TeX theme.

```

14950 \@@_plain_tex_load_theme:VeV
14951 \l_tmpb_tl
14952 { \markdownThemeVersion }
14953 \l_tmpa_str
14954 }
14955 \cs_generate_variant:Nn
14956 \tl_set:Nn
14957 { Ne }
14958 \cs_generate_variant:Nn
14959 \@@_plain_tex_load_theme:nnn
14960 { VeV }

```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```

14961 \prop_gput:Nnn
14962 \g_@@_plain_tex_built_in_themes_prop
14963 { witiko / dot }
14964 {
14965 \str_if_eq:enF
14966 { \markdownThemeVersion }
14967 { silent }
14968 {
14969 \markdownWarning
14970 {
14971 The~theme~name~"witiko/dot"~has~been~soft-deprecated.
14972 \iow_newline:
14973 Consider~changing~the~name~to~"witiko/diagrams@v1".
14974 }
14975 }

```

We enable the `fencedCode` Lua option.

```

14976 \markdownSetup { fencedCode }

```

We store the previous definition of the fenced code token renderer prototype:

```

14977 \cs_set_eq:NN
14978 \c_@@_dot_previous_definition:nnn
14979 \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:

```

14980 \regex_const:Nn
14981 \c_@@_dot_infostring_regex
14982 { ^dot(\s+(.+))? }
14983 \seq_new:N
14984 \l_@@_dot_matches_seq
14985 \markdownSetup {
14986 rendererPrototypes = {
14987 inputFencedCode = {
14988 \regex_extract_once:NnNTF
14989 \c_@@_dot_infostring_regex
14990 { #2 }
14991 \l_@@_dot_matches_seq
14992 {
14993 \c_@@_if_option:nF
14994 { frozenCache }
14995 {
14996 \sys_shell_now:n
14997 {

```

```

14998 if~!~test~-e~#1.pdf.source~
14999 ||~!~diff~#1~#1.pdf.source;
15000 then~
15001 dot~-Tpdf~-o~#1.pdf~#1;
15002 cp~#1~#1.pdf.source;
15003 fi
15004 }
15005 }

```

We include the typeset image using the image token renderer:

```

15006 \exp_args:NNne
15007 \exp_last_unbraced:No
15008 \markdownRendererImage
15009 {
15010 { Graphviz~image }
15011 { #1.pdf }
15012 { #1.pdf }
15013 }
15014 {
15015 \seq_item:Nn
15016 \l_@@_dot_matches_seq
15017 { 3 }
15018 }
15019 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

15020 {
15021 \@@_dot_previous_definition:nnn
15022 { #1 }
15023 { #2 }
15024 { #3 }
15025 }
15026 },
15027 },
15028 }
15029 }

```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

15030 \prop_gput:Nnn
15031 \g_@@_plain_tex_built_in_themes_prop
15032 { witiko / diagrams }
15033 {
15034 \str_case:enF
15035 { \markdownThemeVersion }
15036 {
15037 { latest }

```

```

15038 {
15039 \msg_warning:nnnn
15040 { markdown }
15041 { pin-theme-version }
15042 { witiko/diagrams }
15043 { v2 }
15044 \markdownSetup
15045 {
15046 import = witiko/diagrams/v2,
15047 }
15048 }
15049 { v2 }
15050 {
15051 \markdownSetup
15052 {
15053 import = witiko/diagrams/v2,
15054 }
15055 }
15056 { v1 }
15057 {
15058 \markdownSetup
15059 {
15060 import = witiko/dot@silent,
15061 }
15062 }
15063 }
15064 {
15065 \msg_error:nnnen
15066 { markdown }
15067 { unknown-theme-version }
15068 { witiko/diagrams }
15069 { \markdownThemeVersion }
15070 { v1~and~v2 }
15071 }
15072 }
15073 \cs_generate_variant:Nn
15074 \msg_error:nnnnn
15075 { nnnen }
15076 \msg_new:nnn
15077 { markdown }
15078 { pin-theme-version }
15079 {
15080 Write~"#1@#2"~to~pin-version~"#2"~of~the~theme~"#1".
15081 This~will~keep~your~documents~from~suddenly~breaking
15082 when~we~have~released~future~versions~of~the~theme
15083 with~backwards-incompatible~syntax~and~behavior.
15084 }

```

```

15085 \msg_new:nnnn
15086 { markdown }
15087 { unknown-theme-version }
15088 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
15089 { Known~versions~are:~#3 }

```

Next, we implement the theme [witiko/diagrams/v2](#).

```

15090 \prop_gput:Nnn
15091 \g_@@_plain_tex_built_in_themes_prop
15092 { witiko / diagrams / v2 }
15093 {

```

We enable the [fencedCode](#) and [fencedCodeAttributes](#) Lua option.

```

15094 \@@_setup:n
15095 {
15096 fencedCode = true,
15097 fencedCodeAttributes = true,
15098 }

```

Store the previous fenced code token renderer prototype.

```

15099 \cs_set_eq:NN
15100 \@@_diagrams_previous_fenced_code:nnn
15101 \markdownRendererInputFencedCodePrototype

```

Store the caption, the desired format, and the command to produce the diagram.

```

15102 \tl_new:N
15103 \l_@@_diagrams_caption_tl
15104 \tl_new:N
15105 \l_@@_diagrams_format_tl
15106 \tl_set:Nn
15107 \l_@@_diagrams_format_tl
15108 { pdf }
15109 \tl_new:N
15110 \l_@@_diagrams_command_tl
15111 \@@_setup:n
15112 {
15113 rendererPrototypes = {

```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```

15114 fencedCodeAttributeContextBegin = {
15115 \group_begin:
15116 \markdownRendererImageAttributeContextBegin
15117 \cs_set_eq:NN
15118 \@@_diagrams_previous_key_value:nn
15119 \markdownRendererAttributeKeyValuePrototype
15120 \@@_setup:n
15121 {
15122 rendererPrototypes = {
15123 attributeKeyValue = {

```

```

15124 \str_case:nnF
15125 { ##1 }
15126 {
15127 { caption }
15128 {
15129 \tl_set:Nn
15130 \l_@@_diagrams_caption_tl
15131 { ##2 }
15132 }
15133 { format }
15134 {
15135 \tl_set:Nn
15136 \l_@@_diagrams_format_tl
15137 { ##2 }
15138 }
15139 { command }
15140 {
15141 \tl_set:Nn
15142 \l_@@_diagrams_command_tl
15143 { ##2 }
15144 }
15145 }
15146 {
15147 \@@_diagrams_previous_key_value:nn
15148 { ##1 }
15149 { ##2 }
15150 }
15151 },
15152 },
15153 }
15154 },
15155 fencedCodeAttributeContextEnd = {
15156 \markdownRendererImageAttributeContextEnd
15157 \group_end:
15158 },
15159 },
15160 }
15161 \cs_new:Nn
15162 \@@_diagrams_render_diagram:nnnn
15163 {
15164 \@@_if_option:nF
15165 { frozenCache }
15166 {
15167 \sys_shell_now:n
15168 {
15169 if~!~test~-e~#2.source~
15170 ||~!~diff~#1~#2.source;

```

```

15171 then~
15172 (#3);
15173 cp~#1~#2.source;
15174 fi
15175 }
15176 \exp_args:NNnV
15177 \exp_last_unbraced:No
15178 \markdownRendererImage
15179 {
15180 { #4 }
15181 { #2 }
15182 { #2 }
15183 }
15184 \l_@@_diagrams_caption_tl
15185 }
15186 }

```

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.

```

15187 \prop_new:N
15188 \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

15189 \@@_setup:n
15190 {
15191 rendererPrototypes = {
15192 inputFencedCode = {
15193 \prop_get:NnNTF
15194 \g_markdown_diagrams_infostrings_prop
15195 { #2 }
15196 \l_tmpa_tl
15197 {
15198 \cs_set:NV
15199 \@@_diagrams_infostrings_current:n
15200 \l_tmpa_tl
15201 \@@_diagrams_infostrings_current:n
15202 { #1 }
15203 }

```

Otherwise, use the previous fenced code token renderer prototype.

```

15204 {
15205 \@@_diagrams_previous_fenced_code:nnn
15206 { #1 }
15207 { #2 }
15208 { #3 }
15209 }
15210 },

```

```

15211 },
15212 }
15213 \cs_generate_variant:Nn
15214 \cs_set:Nn
15215 { NV }

```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```

15216 \cs_set:Nn
15217 \@@_diagrams_infostrings_current:n
15218 {
15219 \tl_set:Nn
15220 \l_tmpb_tl
15221 { dot~T }
15222 \tl_put_right:NV
15223 \l_tmpb_tl
15224 \l_@@_diagrams_format_tl
15225 \tl_put_right:Nn
15226 \l_tmpb_tl
15227 { ~-o~#1. }
15228 \tl_put_right:NV
15229 \l_tmpb_tl
15230 \l_@@_diagrams_format_tl
15231 \tl_put_right:Nn
15232 \l_tmpb_tl
15233 { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

15234 \str_if_eq:VnT
15235 \l_@@_diagrams_format_tl
15236 { svg }
15237 {
15238 \tl_put_right:Nn
15239 \l_tmpb_tl
15240 { ;~inkscape~#1. }
15241 \tl_put_right:NV
15242 \l_tmpb_tl
15243 \l_@@_diagrams_format_tl
15244 \tl_put_right:Nn
15245 \l_tmpb_tl
15246 { ~---export-area-drawing---export-dpi=300~-o~#1.pdf }
15247 }
15248 \@@_diagrams_render_diagram:nnVn
15249 { #1 }
15250 { #1.pdf }
15251 \l_tmpb_tl
15252 { Graphviz~image }
15253 }

```



```

15254 \cs_generate_variant:Nn
15255 \@@_diagrams_render_diagram:nnnn
15256 { nnVn }
15257 \@@_tl_set_from_cs:NNn
15258 \l_tmpa_tl
15259 \@@_diagrams_infostrings_current:n
15260 { 1 }
15261 \prop_gput:NnV
15262 \g_markdown_diagrams_infostrings_prop
15263 { dot }
15264 \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, by redefining the `\mmdcCommand` macro, or using the HTML attribute `command` on the fenced code.

Unlike the token list, the macro can be redefined even before loading the Markdown package. Unlike the token list and the macro, the HTML attribute is local to a single fenced code.

```

15265 \tl_new:N
15266 \g_@@_diagrams_mmdc_command_tl
15267 \tl_gset:Nn
15268 \g_@@_diagrams_mmdc_command_tl
15269 {
15270 \tl_if_empty:NTF
15271 \l_@@_diagrams_command_tl
15272 { mmdc }
15273 {
15274 \tl_use:N
15275 \l_@@_diagrams_command_tl
15276 }
15277 }
15278 \cs_if_free:NT
15279 \mmdcCommand
15280 {
15281 \cs_new:Npn
15282 \mmdcCommand
15283 {
15284 \tl_use:N
15285 \g_@@_diagrams_mmdc_command_tl
15286 }
15287 }
15288 \cs_set:Nn
15289 \@@_diagrams_infostrings_current:n
15290 {
15291 \tl_set:Nx

```

```

15292 \l_tmpb_tl
15293 { \mmdcCommand }
15294 \tl_put_right:Nn
15295 \l_tmpb_tl
15296 { ---pdfFit~i~#1~o~#1.pdf }
15297 \@@_diagrams_render_diagram:nnVn
15298 { #1 }
15299 { #1.pdf }
15300 \l_tmpb_tl
15301 { Mermaid~image }
15302 }
15303 \@@_tl_set_from_cs:NNn
15304 \l_tmpa_tl
15305 \@@_diagrams_infostrings_current:n
15306 { 1 }
15307 \prop_gput:NnV
15308 \g_markdown_diagrams_infostrings_prop
15309 { mermaid }
15310 \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

15311 \regex_const:Nn
15312 \c_@@_diagrams_filename_suffix_regex
15313 { \.[^\.]*$ }
15314 \cs_set:Nn
15315 \@@_diagrams_infostrings_current:n
15316 {

```

Use the output format provided by the user.

```

15317 \tl_set:Nn
15318 \l_tmpa_tl
15319 { #1 }
15320 \regex_replace_once:NxN
15321 \c_@@_diagrams_filename_suffix_regex
15322 {
15323 .
15324 \tl_use:N
15325 \l_@@_diagrams_format_tl
15326 }
15327 \l_tmpa_tl
15328 \tl_set:Nn
15329 \l_tmpb_tl
15330 { plantuml~-t }
15331 \tl_put_right:NV
15332 \l_tmpb_tl
15333 \l__markdown_diagrams_format_tl
15334 \tl_put_right:Nn

```

```

15335 \l_tmpb_tl
15336 { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

15337 \str_if_eq:VnT
15338 \l_@@_diagrams_format_tl
15339 { svg }
15340 {
15341 \tl_put_right:Nn
15342 \l_tmpb_tl
15343 { ;~inkscape~ }
15344 \tl_put_right:NV
15345 \l_tmpb_tl
15346 \l_tmpa_tl
15347 \tl_put_right:Nn
15348 \l_tmpb_tl
15349 { ~---export-area-drawing---export-dpi=300~-o~ }
15350 \tl_set:Nn
15351 \l_tmpa_tl
15352 { #1 }
15353 \regex_replace_once:NnN
15354 \c_@@_diagrams_filename_suffix_regex
15355 { .pdf }
15356 \l_tmpa_tl
15357 \tl_put_right:NV
15358 \l_tmpb_tl
15359 \l_tmpa_tl
15360 }
15361 \@@_diagrams_render_diagram:nVVn
15362 { #1 }
15363 \l_tmpa_tl
15364 \l_tmpb_tl
15365 { PlantUML~image }
15366 }
15367 \cs_generate_variant:Nn
15368 \@@_diagrams_render_diagram:nnnn
15369 { nVVn }
15370 \cs_generate_variant:Nn
15371 \regex_replace_once:NnN
15372 { NxN }
15373 \@@_tl_set_from_cs:NNn
15374 \l_tmpa_tl
15375 \@@_diagrams_infostrings_current:n
15376 { 1 }
15377 \prop_gput:NnV
15378 \g_markdown_diagrams_infostrings_prop
15379 { plantuml }
15380 \l_tmpa_tl

```

```
15381 }
```

We locally change the category code of percent signs, so that we can use them in the shell code:

```
15382 \group_begin:
15383 \char_set_catcode_other:N \%
```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```
15384 \prop_gput:Nnn
15385 \g_@@_plain_tex_built_in_themes_prop
15386 { witiko / graphicx / http }
15387 {
15388 \cs_set_eq:NN
15389 \@@_graphicx_http_previous_definition:nnnn
15390 \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.

```
15391 \int_new:N
15392 \g_@@_graphicx_http_image_number_int
15393 \int_gset:Nn
15394 \g_@@_graphicx_http_image_number_int
15395 { 0 }
15396 \cs_new:Nn
15397 \@@_graphicx_http_filename:
15398 {
15399 \markdownOptionCacheDir
15400 / witiko_graphicx_http .
15401 \int_use:N
15402 \g_@@_graphicx_http_image_number_int
15403 }
```

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.

```
15404 \cs_new:Nn
15405 \@@_graphicx_http_download:nn
15406 {
15407 wget~-0~#2~#1~
15408 ||~curl~--location~-o~#2~#1~
15409 ||~rm~-f~#2
15410 }
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
15411 \str_new:N
```

```

15412 \l_@@_graphicx_http_filename_str
15413 \ior_new:N
15414 \g_@@_graphicx_http_filename_ior
15415 \markdownSetup {
15416 rendererPrototypes = {
15417 image = {
15418 \@@_if_option:nF
15419 { frozenCache }
15420 }

```

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:

```

15421 \sys_shell_now:e
15422 {
15423 mkdir~~p~" \markdownOptionCacheDir ";
15424 if~printf~'%s'~"#3"~|~grep~-q~-E~'^https?:';
15425 then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

15426 OUTPUT_PREFIX=" \markdownOptionCacheDir ";
15427 OUTPUT_BODY="$(printf~'%s'~'#3'
15428 |~md5sum~|~cut~-d~'~'~-f1)";
15429 OUTPUT_SUFFIX="$(printf~'%s'~'#3'
15430 |~sed~'s/.*[.]//')";
15431 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

15432 if~!~[~-e~"$OUTPUT"~];
15433 then~
15434 \@@_graphicx_http_download:nn
15435 { '#3' }
15436 { "$OUTPUT" } ;
15437 printf~'%s'~"$OUTPUT"~
15438 >~" \@@_graphicx_http_filename: ";
15439 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:

```

15440 else~
15441 printf~'%s'~'#3'~
15442 >~" \@@_graphicx_http_filename: ";
15443 fi
15444 }
15445 }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

15446 \ior_open:Ne

```

```

15447 \g_@@_graphicx_http_filename_ior
15448 { \@@_graphicx_http_filename: }
15449 \ior_str_get:NN
15450 \g_@@_graphicx_http_filename_ior
15451 \l_@@_graphicx_http_filename_str
15452 \ior_close:N
15453 \g_@@_graphicx_http_filename_ior
15454 \@@_graphicx_http_previous_definition:nnVn
15455 { #1 }
15456 { #2 }
15457 \l_@@_graphicx_http_filename_str
15458 { #4 }
15459 \int_gincr:N
15460 \g_@@_graphicx_http_image_number_int
15461 }
15462 }
15463 }
15464 \cs_generate_variant:Nn
15465 \ior_open:Nn
15466 { Ne }
15467 \cs_generate_variant:Nn
15468 \@@_graphicx_http_previous_definition:nnnn
15469 { nnVn }
15470 }
15471 \group_end:

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

15472 \prop_gput:Nnn
15473 \g_@@_plain_tex_built_in_themes_prop
15474 { witiko / tilde }
15475 {
15476 \markdownSetup {
15477 rendererPrototypes = {
15478 tilde = {~},
15479 },
15480 }
15481 }

```

The themes [witiko/example/foo](#) and [witiko/example/bar](#) are supposed to be used in code examples. They don't do anything.

```

15482 \clist_map_inline:nn
15483 { foo, bar }
15484 {
15485 \prop_gput:Nnn
15486 \g_@@_plain_tex_built_in_themes_prop
15487 { witiko / example / #1 }
15488 {

```

```

15489 \markdownWarning
15490 {
15491 The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
15492 examples.~Using~it~in~actual~code~has~no~effect,~except~
15493 this~warning~message,~and~is~usually~a~mistake.
15494 }
15495 }
15496 }
15497 \ExplSyntaxOff

```

The `witiko/markdown/defaults` plain TeX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

15498 \def\markdownRendererInterblockSeparatorPrototype{\par}%
15499 \def\markdownRendererParagraphSeparatorPrototype{%
15500 \markdownRendererInterblockSeparator}%
15501 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
15502 \def\markdownRendererSoftLineBreakPrototype{ }%
15503 \let\markdownRendererEllipsisPrototype\dots
15504 \def\markdownRendererNbspPrototype{~}%
15505 \def\markdownRendererLeftBracePrototype{\char`\{}%
15506 \def\markdownRendererRightBracePrototype{\char`\}%
15507 \def\markdownRendererDollarSignPrototype{\char`$}%
15508 \def\markdownRendererPercentSignPrototype{\char`\}%
15509 \def\markdownRendererAmpersandPrototype{\&}%
15510 \def\markdownRendererUnderscorePrototype{\char`_%
15511 \def\markdownRendererHashPrototype{\char`\#}%
15512 \def\markdownRendererCircumflexPrototype{\char`^}%
15513 \def\markdownRendererBackslashPrototype{\char`\}%
15514 \def\markdownRendererTildePrototype{\char`~}%
15515 \def\markdownRendererPipePrototype{|}%
15516 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
15517 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
15518 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
15519 \markdownInput{#3}}%
15520 \def\markdownRendererContentBlockOnlineImagePrototype{%
15521 \markdownRendererImage}%
15522 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
15523 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
15524 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
15525 \def\markdownRendererUlBeginPrototype{}%
15526 \def\markdownRendererUlBeginTightPrototype{}%
15527 \def\markdownRendererUlItemPrototype{}%
15528 \def\markdownRendererUlItemEndPrototype{}%

```

```

15529 \def\markdownRendererU1EndPrototype{}%
15530 \def\markdownRendererU1EndTightPrototype{}%
15531 \def\markdownRendererO1BeginPrototype{}%
15532 \def\markdownRendererO1BeginTightPrototype{}%
15533 \def\markdownRendererFancyO1BeginPrototype#1#2{%
15534 \markdownRendererO1Begin}%
15535 \def\markdownRendererFancyO1BeginTightPrototype#1#2{%
15536 \markdownRendererO1BeginTight}%
15537 \def\markdownRendererO1ItemPrototype{}%
15538 \def\markdownRendererO1ItemWithNumberPrototype#1{}%
15539 \def\markdownRendererO1ItemEndPrototype{}%
15540 \def\markdownRendererFancyO1ItemPrototype{\markdownRendererO1Item}%
15541 \def\markdownRendererFancyO1ItemWithNumberPrototype{%
15542 \markdownRendererO1ItemWithNumber}%
15543 \def\markdownRendererFancyO1ItemEndPrototype{}%
15544 \def\markdownRendererO1EndPrototype{}%
15545 \def\markdownRendererO1EndTightPrototype{}%
15546 \def\markdownRendererFancyO1EndPrototype{\markdownRendererO1End}%
15547 \def\markdownRendererFancyO1EndTightPrototype{%
15548 \markdownRendererO1EndTight}%
15549 \def\markdownRendererD1BeginPrototype{}%
15550 \def\markdownRendererD1BeginTightPrototype{}%
15551 \def\markdownRendererD1ItemPrototype#1{#1}%
15552 \def\markdownRendererD1ItemEndPrototype{}%
15553 \def\markdownRendererD1DefinitionBeginPrototype{}%
15554 \def\markdownRendererD1DefinitionEndPrototype{\par}%
15555 \def\markdownRendererD1EndPrototype{}%
15556 \def\markdownRendererD1EndTightPrototype{}%
15557 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
15558 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
15559 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
15560 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
15561 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
15562 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
15563 \def\markdownRendererInputVerbatimPrototype#1{%
15564 \par{\tt\input#1\relax{}}\par}%
15565 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
15566 \markdownRendererInputVerbatim{#1}}%
15567 \def\markdownRendererHeadingOnePrototype#1{#1}%
15568 \def\markdownRendererHeadingTwoPrototype#1{#1}%
15569 \def\markdownRendererHeadingThreePrototype#1{#1}%
15570 \def\markdownRendererHeadingFourPrototype#1{#1}%
15571 \def\markdownRendererHeadingFivePrototype#1{#1}%
15572 \def\markdownRendererHeadingSixPrototype#1{#1}%
15573 \def\markdownRendererThematicBreakPrototype{}%
15574 \def\markdownRendererNotePrototype#1{#1}%
15575 \def\markdownRendererCitePrototype#1{}%

```



```

15576 \def\markdownRendererTextCitePrototype#1{%
15577 \def\markdownRendererTickedBoxPrototype{[X]}%
15578 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
15579 \def\markdownRendererUntickedBoxPrototype{[]}%
15580 \def\markdownRendererStrikeThroughPrototype#1{#1}%
15581 \def\markdownRendererSuperscriptPrototype#1{#1}%
15582 \def\markdownRendererSubscriptPrototype#1{#1}%
15583 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
15584 \def\markdownRendererInlineMathPrototype#1{$#1$}%
15585 \ExplSyntaxOn
15586 \cs_gset_protected:Npn
15587 \markdownRendererHeaderAttributeContextBeginPrototype
15588 {
15589 \group_begin:
15590 \color_group_begin:
15591 }
15592 \cs_gset_protected:Npn
15593 \markdownRendererHeaderAttributeContextEndPrototype
15594 {
15595 \color_group_end:
15596 \group_end:
15597 }
15598 \cs_gset_eq:NN
15599 \markdownRendererBracketedSpanAttributeContextBeginPrototype
15600 \markdownRendererHeaderAttributeContextBeginPrototype
15601 \cs_gset_eq:NN
15602 \markdownRendererBracketedSpanAttributeContextEndPrototype
15603 \markdownRendererHeaderAttributeContextEndPrototype
15604 \cs_gset_eq:NN
15605 \markdownRendererFencedDivAttributeContextBeginPrototype
15606 \markdownRendererHeaderAttributeContextBeginPrototype
15607 \cs_gset_eq:NN
15608 \markdownRendererFencedDivAttributeContextEndPrototype
15609 \markdownRendererHeaderAttributeContextEndPrototype
15610 \cs_gset_eq:NN
15611 \markdownRendererFencedCodeAttributeContextBeginPrototype
15612 \markdownRendererHeaderAttributeContextBeginPrototype
15613 \cs_gset_eq:NN
15614 \markdownRendererFencedCodeAttributeContextEndPrototype
15615 \markdownRendererHeaderAttributeContextEndPrototype
15616 \cs_gset:Npn
15617 \markdownRendererReplacementCharacterPrototype
15618 { \codepoint_str_generate:n { fffd } }
15619 \ExplSyntaxOff
15620 \def\markdownRendererSectionBeginPrototype{%
15621 \def\markdownRendererSectionEndPrototype{%
15622 \ExplSyntaxOn

```

```

15623 \cs_gset_protected:Npn
15624 \markdownRendererWarningPrototype
15625 #1#2#3#4
15626 {
15627 \tl_set:Nn
15628 \l_tmpa_tl
15629 { #2 }
15630 \tl_if_empty:nF
15631 { #4 }
15632 {
15633 \tl_put_right:Nn
15634 \l_tmpa_tl
15635 { \iow_newline: #4 }
15636 }
15637 \exp_args:NV
15638 \markdownWarning
15639 \l_tmpa_tl
15640 }
15641 \ExplSyntaxOff
15642 \def\markdownRendererErrorPrototype#1#2#3#4{%
15643 \markdownError{#2}{#4}}%
15644 \def\markdownRendererBracketedSpanPrototype#1{#1}%

```

### 3.2.3.1 CommonMark Raw html Renderers

The CommonMark raw HTML renderer prototypes (see Section 2.2.5.13) fall back onto the renderer for generic inline HTML tags.

```

15645 \def\markdownRendererInlineHtmlProcessingInstructionPrototype{%
15646 \markdownRendererInlineHtmlTagPrototype}%
15647 \def\markdownRendererInlineHtmlDeclarationPrototype{%
15648 \markdownRendererInlineHtmlTagPrototype}%
15649 \def\markdownRendererInlineHtmlCdataSectionPrototype{%
15650 \markdownRendererInlineHtmlTagPrototype}%
15651 \def\markdownRendererInlineHtmlOpenTagPrototype{%
15652 \markdownRendererInlineHtmlTagPrototype}%
15653 \def\markdownRendererInlineHtmlCloseTagPrototype{%
15654 \markdownRendererInlineHtmlTagPrototype}%
15655 \def\markdownRendererInlineHtmlEmptyTagPrototype{%
15656 \markdownRendererInlineHtmlTagPrototype}%

```

### 3.2.3.2 CommonMark Block html Renderers

Most CommonMark block HTML renderer prototypes (see Section 2.2.5.12) fall back onto the renderer prototype for generic block HTML elements.

```

15657 \def\markdownRendererInputBlockHtmlCdataElementPrototype{%
15658 \markdownRendererInputBlockHtmlElementPrototype}%
15659 \def\markdownRendererInputBlockHtmlProcessingInstructionPrototype{%

```

```

15660 \markdownRendererInputBlockHtmlElementPrototype}%
15661 \def\markdownRendererInputBlockHtmlDeclarationPrototype{%
15662 \markdownRendererInputBlockHtmlElementPrototype}%
15663 \def\markdownRendererInputBlockHtmlCdataSectionPrototype{%
15664 \markdownRendererInputBlockHtmlElementPrototype}%
15665 \def\markdownRendererInputBlockHtmlPcdataElementPrototype{%
15666 \markdownRendererInputBlockHtmlElementPrototype}%

```

The block comment HTML renderer (type 2) falls back on the renderer prototype for inline HTML elements.

```

15667 \def\markdownRendererBlockHtmlCommentPrototype{%
15668 \markdownRendererInlineHtmlCommentPrototype}%

```

The standalone HTML tag renderer (type 7) falls back on the renderer prototype for generic inline HTML tags.

```

15669 \def\markdownRendererBlockHtmlStandaloneTagPrototype{%
15670 \markdownRendererInlineHtmlTagPrototype}%

```

### 3.2.3.3 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.

```

15671 \ExplSyntaxOn
15672 \cs_new:Nn
15673 \@@_plain_tex_default_input_raw_inline:nn
15674 {
15675 \str_case:nn
15676 { #2 }
15677 {
15678 { md } { \markdownInput{#1} }
15679 { tex } { \markdownEscape{#1} \unskip }
15680 }
15681 }
15682 \cs_new:Nn
15683 \@@_plain_tex_default_input_raw_block:nn
15684 {
15685 \str_case:nn
15686 { #2 }
15687 {
15688 { md } { \markdownInput{#1} }
15689 { tex } { \markdownEscape{#1} }
15690 }
15691 }
15692 \cs_gset:Npn
15693 \markdownRendererInputRawInlinePrototype#1#2
15694 {
15695 \@@_plain_tex_default_input_raw_inline:nn

```

```

15696 { #1 }
15697 { #2 }
15698 }
15699 \cs_gset:Npn
15700 \markdownRendererInputRawBlockPrototype#1#2
15701 {
15702 \@@_plain_tex_default_input_raw_block:nn
15703 { #1 }
15704 { #2 }
15705 }
15706 \ExplSyntaxOff

```

### 3.2.3.4 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  $p$ .

```

15707 \ExplSyntaxOn
15708 \seq_new:N \g_@@_jekyll_data_datatypes_seq
15709 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
15710 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
15711 \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.

```

15712 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
15713 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
15714 {
15715 \seq_if_empty:NF
15716 \g_@@_jekyll_data_datatypes_seq
15717 {
15718 \seq_get_right:NN
15719 \g_@@_jekyll_data_datatypes_seq
15720 \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.

```

15721 \str_if_eq:NNTF
15722 \l_tmpa_tl
15723 \c_@@_jekyll_data_sequence_tl
15724 {
15725 \seq_put_right:Nn
15726 \g_@@_jekyll_data_wildcard_absolute_address_seq
15727 { * }
15728 }
15729 {
15730 \seq_put_right:Nn
15731 \g_@@_jekyll_data_wildcard_absolute_address_seq
15732 { #1 }
15733 }
15734 }
15735 }

```

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.

```

15736 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
15737 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
15738 \cs_new_protected:Nn \@@_jekyll_data_concatenate_address:NN
15739 {
15740 \seq_pop_left:NN #1 \l_tmpa_tl
15741 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
15742 \seq_put_left:NV #1 \l_tmpa_tl
15743 }
15744 \cs_new_protected:Nn \@@_jekyll_data_update_address_tls:
15745 {
15746 \@@_jekyll_data_concatenate_address:NN
15747 \g_@@_jekyll_data_wildcard_absolute_address_seq
15748 \g_@@_jekyll_data_wildcard_absolute_address_tl
15749 \seq_get_right:NN
15750 \g_@@_jekyll_data_wildcard_absolute_address_seq
15751 \g_@@_jekyll_data_wildcard_relative_address_tl
15752 }

```

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.

```

15753 \cs_new_protected:Nn \@@_jekyll_data_push:nN
15754 {
15755 \@@_jekyll_data_push_address_segment:n
15756 { #1 }
15757 \seq_put_right:NV
15758 \g_@@_jekyll_data_datatypes_seq
15759 #2
15760 \@@_jekyll_data_update_address_tls:
15761 }
15762 \cs_new_protected:Nn \@@_jekyll_data_pop:
15763 {
15764 \seq_pop_right:NN
15765 \g_@@_jekyll_data_wildcard_absolute_address_seq
15766 \l_tmpa_tl
15767 \seq_pop_right:NN
15768 \g_@@_jekyll_data_datatypes_seq
15769 \l_tmpa_tl
15770 \@@_jekyll_data_update_address_tls:
15771 }

```

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.

```

15772 \cs_new_protected:Nn \@@_jekyll_data_set_keyval_known:nn
15773 {
15774 \keys_set_known:nn
15775 { markdown/jekyllData }
15776 { { #1 } = { #2 } }

```

```

15777 }
15778 \cs_generate_variant:Nn
15779 \@@_jekyll_data_set_keyval_known:nn
15780 { Vn }
15781 \cs_new_protected:Nn \@@_jekyll_data_set_keyvals_known:nn
15782 {
15783 \@@_jekyll_data_push:nN
15784 { #1 }
15785 \c_@@_jekyll_data_scalar_tl
15786 \@@_jekyll_data_set_keyval_known:Vn
15787 \g_@@_jekyll_data_wildcard_absolute_address_tl
15788 { #2 }
15789 \@@_jekyll_data_set_keyval_known:Vn
15790 \g_@@_jekyll_data_wildcard_relative_address_tl
15791 { #2 }
15792 \@@_jekyll_data_pop:
15793 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

15794 \cs_gset_protected:Npn
15795 \markdownRendererJekyllDataSequenceBeginPrototype #1#2
15796 {
15797 \@@_jekyll_data_push:nN
15798 { #1 }
15799 \c_@@_jekyll_data_sequence_tl
15800 }
15801 \cs_gset_protected:Npn
15802 \markdownRendererJekyllDataMappingBeginPrototype #1#2
15803 {
15804 \@@_jekyll_data_push:nN
15805 { #1 }
15806 \c_@@_jekyll_data_mapping_tl
15807 }
15808 \cs_gset_protected:Npn
15809 \markdownRendererJekyllDataSequenceEndPrototype
15810 {
15811 \@@_jekyll_data_pop:
15812 }
15813 \cs_gset_protected:Npn
15814 \markdownRendererJekyllDataMappingEndPrototype
15815 {
15816 \@@_jekyll_data_pop:
15817 }
15818 \cs_gset_protected:Npn
15819 \markdownRendererJekyllDataBooleanPrototype #1#2
15820 {

```

```

15821 \@@_jekyll_data_set_keyvals_known:nn
15822 { #1 }
15823 { #2 }
15824 }
15825 \def\markdownRendererJekyllDataEmptyPrototype#1{}
15826 \cs_gset_protected:Npn
15827 \markdownRendererJekyllDataNumberPrototype #1#2
15828 {
15829 \@@_jekyll_data_set_keyvals_known:nn
15830 { #1 }
15831 { #2 }
15832 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

15833 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
15834 \cs_gset_protected:Npn
15835 \markdownRendererJekyllDataTypographicStringPrototype #1#2
15836 {
15837 \@@_jekyll_data_set_keyvals_known:nn
15838 { #1 }
15839 { #2 }
15840 }
15841 \ExplSyntaxOff

```

### 3.2.3.5 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.

```

15842 \ExplSyntaxOn
15843 \@@_with_various_cases:nn
15844 { jekyllDataKeyValue }
15845 {
15846 \keys_define:nn
15847 { markdown/options }
15848 {
15849 #1 .code:n = {
15850 \@@_route_jekyll_data_to_key_values:n
15851 { ##1 }
15852 },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

15853 #1 .value_required:n = { false },
15854 #1 .default:n = { },
15855 }
15856 }

```



```

15857 \seq_new:N
15858 \l_@@_jekyll_data_current_position_seq
15859 \tl_new:N
15860 \l_@@_jekyll_data_current_position_tl
15861 \cs_new_protected:Nn
15862 \@@_route_jekyll_data_to_key_values:n
15863 {
15864 \markdownSetup
15865 {
15866 renderers = {
15867 jekyllData(Sequence|Mapping)Begin = {
15868 \bool_lazy_and:nnTF
15869 {
15870 \seq_if_empty_p:N
15871 \l_@@_jekyll_data_current_position_seq
15872 }
15873 {
15874 \str_if_eq_p:nn
15875 { ##1 }
15876 { null }
15877 }
15878 }
15879 \tl_if_empty:nF
15880 { #1 }
15881 {
15882 \seq_put_right:Nn
15883 \l_@@_jekyll_data_current_position_seq
15884 { #1 }
15885 }
15886 }
15887 }
15888 \seq_put_right:Nn
15889 \l_@@_jekyll_data_current_position_seq
15890 { ##1 }
15891 }
15892 },
15893 jekyllData(Sequence|Mapping)End = {
15894 \seq_pop_right:NN
15895 \l_@@_jekyll_data_current_position_seq
15896 \l_tmpa_tl
15897 },

```

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`.

```

15898 jekyllDataBoolean = {

```

```

15899 \tl_set:Nx
15900 \l_@@_jekyll_data_current_position_tl
15901 {
15902 \seq_use:Nn
15903 \l_@@_jekyll_data_current_position_seq
15904 { / }
15905 }
15906 \keys_if_exist:VnTF
15907 \l_@@_jekyll_data_current_position_tl
15908 { ##1 / boolean }
15909 {
15910 \@@_keys_set:xn
15911 {
15912 \tl_use:N
15913 \l_@@_jekyll_data_current_position_tl
15914 / ##1 / boolean
15915 }
15916 { ##2 }
15917 }
15918 {
15919 \@@_keys_set:xn
15920 {
15921 \tl_use:N
15922 \l_@@_jekyll_data_current_position_tl
15923 / ##1
15924 }
15925 { ##2 }
15926 }
15927 },
15928 jekyllDataNumber = {
15929 \tl_set:Nx
15930 \l_@@_jekyll_data_current_position_tl
15931 {
15932 \seq_use:Nn
15933 \l_@@_jekyll_data_current_position_seq
15934 { / }
15935 }
15936 \keys_if_exist:VnTF
15937 \l_@@_jekyll_data_current_position_tl
15938 { ##1 / number }
15939 {
15940 \@@_keys_set:xn
15941 {
15942 \tl_use:N
15943 \l_@@_jekyll_data_current_position_tl
15944 / ##1 / number
15945 }

```

```

15946 { ##2 }
15947 }
15948 {
15949 \@@_keys_set:xn
15950 {
15951 \tl_use:N
15952 \l_@@_jekyll_data_current_position_tl
15953 / ##1
15954 }
15955 { ##2 }
15956 }
15957 },

```

For the  $\langle non-string\ type \rangle$  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`.

```

15958 jekyllDataEmpty = {
15959 \tl_set:Nx
15960 \l_@@_jekyll_data_current_position_tl
15961 {
15962 \seq_use:Nn
15963 \l_@@_jekyll_data_current_position_seq
15964 { / }
15965 }
15966 \keys_if_exist:VnTF
15967 \l_@@_jekyll_data_current_position_tl
15968 { ##1 / empty }
15969 {
15970 \keys_set:xn
15971 {
15972 \tl_use:N
15973 \l_@@_jekyll_data_current_position_tl
15974 / ##1
15975 }
15976 { empty }
15977 }
15978 {
15979 \keys_set:Vn
15980 \l_@@_jekyll_data_current_position_tl
15981 { ##1 }
15982 }
15983 },

```

For every YAML key `path.to.<key>` with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key–value  $\langle module \rangle/path/to/\langle key \rangle$  if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key  $\langle key \rangle$  of the

key–value  $\langle module \rangle$ /[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  $\langle key \rangle$  if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set  $\langle key \rangle$  normally, i.e. regardless of whether it is known or unknown.

```

15984 jekyllDataTypographicString = {
15985 \tl_set:Nx
15986 \l_@@_jekyll_data_current_position_tl
15987 {
15988 \seq_use:Nn
15989 \l_@@_jekyll_data_current_position_seq
15990 { / }
15991 }
15992 \keys_if_exist:VnTF
15993 \l_@@_jekyll_data_current_position_tl
15994 { ##1 / typographicString }
15995 {
15996 \@@_keys_set:xn
15997 {
15998 \tl_use:N
15999 \l_@@_jekyll_data_current_position_tl
16000 / ##1 / typographicString
16001 }
16002 { ##2 }
16003 }
16004 {
16005 \keys_if_exist:VnTF
16006 \l_@@_jekyll_data_current_position_tl
16007 { ##1 / programmaticString }
16008 {
16009 \@@_keys_set_known:xn
16010 {
16011 \tl_use:N
16012 \l_@@_jekyll_data_current_position_tl
16013 / ##1
16014 }
16015 { ##2 }
16016 }
16017 {
16018 \@@_keys_set:xn
16019 {
16020 \tl_use:N
16021 \l_@@_jekyll_data_current_position_tl
16022 / ##1
16023 }
16024 { ##2 }

```

```

16025 }
16026 }
16027 },
16028 jekyllDataProgrammaticString = {
16029 \tl_set:Nx
16030 \l_@@_jekyll_data_current_position_tl
16031 {
16032 \seq_use:Nn
16033 \l_@@_jekyll_data_current_position_seq
16034 { / }
16035 }
16036 \keys_if_exist:VnT
16037 \l_@@_jekyll_data_current_position_tl
16038 { ##1 / programmaticString }
16039 {
16040 \@@_keys_set:xn
16041 {
16042 \tl_use:N
16043 \l_@@_jekyll_data_current_position_tl
16044 / ##1 / programmaticString
16045 }
16046 { ##2 }
16047 }
16048 },
16049 },
16050 }
16051 }
16052 \cs_new_protected:Nn
16053 \@@_keys_set:nn
16054 {
16055 \keys_set:nn
16056 { }
16057 { { #1 } = { #2 } }
16058 }
16059 \cs_new_protected:Nn
16060 \@@_keys_set_known:nn
16061 {
16062 \keys_set_known:nn
16063 { }
16064 { { #1 } = { #2 } }
16065 }
16066 \cs_generate_variant:Nn
16067 \@@_keys_set:nn
16068 { xn }
16069 \cs_generate_variant:Nn
16070 \@@_keys_set_known:nn
16071 { xn }

```

```

16072 \cs_generate_variant:Nn
16073 \keys_set:nn
16074 { xn, Vn }
16075 \prg_generate_conditional_variant:Nnn
16076 \keys_if_exist:nn
16077 { Vn }
16078 { T, TF }
16079 \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).

```

16080 \ExplSyntaxOn
16081 \str_if_eq:VVT
16082 \c_@@_top_layer_tl
16083 \c_@@_option_layer_plain_tex_tl
16084 {
16085 \use:c
16086 { ExplSyntaxOff }
16087 \@@_if_option:nF
16088 { noDefaults }
16089 {
16090 \@@_if_option:nTF
16091 { experimental }
16092 {
16093 \@@_setup:n
16094 { theme = witiko/markdown/defaults@experimental }
16095 }
16096 {
16097 \@@_setup:n
16098 { theme = witiko/markdown/defaults }
16099 }
16100 }
16101 \use:c
16102 { ExplSyntaxOn }
16103 }
16104 \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 T<sub>E</sub>X options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

16105 \ExplSyntaxOn
16106 \tl_new:N \g_@@_formatted_lua_options_tl
16107 \cs_new_protected:Nn \@@_format_lua_options:
16108 {

```

```

16109 \tl_gclear:N
16110 \g_@@_formatted_lua_options_tl
16111 \seq_map_function:NN
16112 \g_@@_lua_options_seq
16113 \@@_format_lua_option:n
16114 }
16115 \cs_new_protected:Nn \@@_format_lua_option:n
16116 {
16117 \@@_typecheck_option:n
16118 { #1 }
16119 \@@_get_option_type:nN
16120 { #1 }
16121 \l_tmpa_tl
16122 \bool_case_true:nF
16123 {
16124 {
16125 \str_if_eq_p:VV
16126 \l_tmpa_tl
16127 \c_@@_option_type_boolean_tl ||
16128 \str_if_eq_p:VV
16129 \l_tmpa_tl
16130 \c_@@_option_type_number_tl ||
16131 \str_if_eq_p:VV
16132 \l_tmpa_tl
16133 \c_@@_option_type_counter_tl
16134 }
16135 {
16136 \@@_get_option_value:nN
16137 { #1 }
16138 \l_tmpa_tl
16139 \tl_gput_right:Nx
16140 \g_@@_formatted_lua_options_tl
16141 { #1~~~ \l_tmpa_tl ,~ }
16142 }
16143 }
16144 \str_if_eq_p:VV
16145 \l_tmpa_tl
16146 \c_@@_option_type_clist_tl
16147 }
16148 {
16149 \@@_get_option_value:nN
16150 { #1 }
16151 \l_tmpa_tl
16152 \tl_gput_right:Nx
16153 \g_@@_formatted_lua_options_tl
16154 { #1~~~\c_left_brace_str }
16155 \clist_map_inline:Vn

```

```

16156 \l_tmpa_tl
16157 {
16158 \@@_lua_escape:xN
16159 { ##1 }
16160 \l_tmpb_tl
16161 \tl_gput_right:Nn
16162 \g_@@_formatted_lua_options_tl
16163 { " }
16164 \tl_gput_right:NV
16165 \g_@@_formatted_lua_options_tl
16166 \l_tmpb_tl
16167 \tl_gput_right:Nn
16168 \g_@@_formatted_lua_options_tl
16169 { " ,~ }
16170 }
16171 \tl_gput_right:Nx
16172 \g_@@_formatted_lua_options_tl
16173 { \c_right_brace_str ,~ }
16174 }
16175 }
16176 {
16177 \@@_get_option_value:nN
16178 { #1 }
16179 \l_tmpa_tl
16180 \@@_lua_escape:xN
16181 { \l_tmpa_tl }
16182 \l_tmpb_tl
16183 \tl_gput_right:Nn
16184 \g_@@_formatted_lua_options_tl
16185 { #1~~ " }
16186 \tl_gput_right:NV
16187 \g_@@_formatted_lua_options_tl
16188 \l_tmpb_tl
16189 \tl_gput_right:Nn
16190 \g_@@_formatted_lua_options_tl
16191 { " ,~ }
16192 }
16193 }
16194 \cs_generate_variant:Nn
16195 \clist_map_inline:nn
16196 { Vn }
16197 \let
16198 \markdownPrepareLuaOptions
16199 \@@_format_lua_options:
16200 \def
16201 \markdownLuaOptions
16202 {

```



```

16203 {
16204 \g_@@_formatted_lua_options_tl
16205 }
16206 }
16207 \sys_if_engine luatex:TF
16208 {
16209 \cs_new_protected:Nn
16210 \@@_lua_escape:nN
16211 {
16212 \tl_set:Nx
16213 #2
16214 {
16215 \lua_escape:n
16216 { #1 }
16217 }
16218 }
16219 }
16220 {
16221 \regex_const:Nn
16222 \c_@@_lua_escape_regex
16223 { [\\"'] }
16224 \cs_new_protected:Nn
16225 \@@_lua_escape:nN
16226 {
16227 \tl_set:Nn
16228 #2
16229 { #1 }
16230 \regex_replace_all:NnN
16231 \c_@@_lua_escape_regex
16232 { \u { c_backslash_str } \0 }
16233 #2
16234 }
16235 }
16236 \cs_generate_variant:Nn
16237 \@@_lua_escape:nN
16238 { 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.

```

16239 \tl_new:N
16240 \markdownInputFilename
16241 \cs_new_protected:Npn
16242 \markdownPrepareInputFilename
16243 #1
16244 {
16245 \@@_lua_escape:xN

```

```

16246 { #1 }
16247 \markdownInputFilename
16248 \tl_gset:Nx
16249 \markdownInputFilename
16250 { " \markdownInputFilename " }
16251 }

```

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.

```

16252 \cs_new:Npn
16253 \markdownPrepare
16254 {

```

First, ensure that the `cacheDir` directory exists.

```

16255 local~lfs = require("lfs")
16256 local~options = \markdownLuaOptions
16257 if~not~lfs.isdir(options.cacheDir) then~
16258 assert(lfs.mkdir(options.cacheDir))
16259 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.

```

16260 local~md = require("markdown")
16261 local~convert = md.new(options)
16262 }

```

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.

```

16263 \cs_new:Npn
16264 \markdownConvert
16265 {
16266 local~filename = \markdownInputFilename
16267 local~file = assert(io.open(filename, "r"),
16268 [[Could~not~open~file~]] .. filename .. [[~for~reading]])
16269 local~input = assert(file:read("*a"))
16270 assert(file:close())
16271 print(convert(input))
16272 }
16273 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```

16274 \def\markdownCleanup{%

```

Remove the `options.cacheDir` directory if it is empty.

```

16275 if options.cacheDir then
16276 lfs.rmdir(options.cacheDir)

```

```

16277 end
16278 }%

```

### 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.

```

16279 \csname newread\endcsname\markdownInputFileStream
16280 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

16281 \begingroup
16282 \catcode\^^I=12%
16283 \gdef\markdownReadAndConvertTab{^^I}%
16284 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

16285 \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.

```

16286 \catcode\^^M=13%
16287 \catcode\^^I=13%
16288 \catcode|=0%
16289 \catcode\=12%
16290 |catcode@=14%
16291 |catcode|=12@
16292 |gdef|markdownReadAndConvert#1#2{@
16293 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

16294 |markdownIfOption{frozenCache}{@}{@
16295 |immediate|openout|markdownOutputFileStream@
16296 |markdownOptionInputTempFileName|relax@
16297 |markdownInfo{@
16298 Buffering block-level markdown input into the temporary @
16299 input file "|markdownOptionInputTempFileName" and scanning @
16300 for the closing token sequence "#1"}@
16301 }@

```

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.

```
16302 |def|do##1{|catcode`##1=12}|dospecials@
16303 |catcode`| =12@
16304 |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.

```
16305 |def|markdownReadAndConvertStripPercentSign##1{@
16306 |markdownIfOption{stripPercentSigns}{@
16307 |if##1%@
16308 |expandafter|expandafter|expandafter@
16309 |markdownReadAndConvertProcessLine@
16310 |else@
16311 |expandafter|expandafter|expandafter@
16312 |markdownReadAndConvertProcessLine@
16313 |expandafter|expandafter|expandafter##1@
16314 |fi@
16315 }{@
16316 |expandafter@
16317 |markdownReadAndConvertProcessLine@
16318 |expandafter##1@
16319 }@
16320 }@
```

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.

```
16321 |def|markdownReadAndConvertProcessLine##1##2##3|relax{@
```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```
16322 |ifx|relax##3|relax@
16323 |markdownIfOption{frozenCache}{}{@
16324 |immediate|write|markdownOutputFileStream{##1}@
16325 }@
16326 |else@
```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

16327 |def^^M{@
16328 |markdownInfo{The ending token sequence was found}@
16329 |markdownIfOption{frozenCache}{-}{@
16330 |immediate|closeout|markdownOutputFileStream@
16331 }@
16332 |endgroup@
16333 |markdownInput{@
16334 |markdownOptionOutputDir@
16335 /|markdownOptionInputTempFileName@
16336 }@
16337 #2}@
16338 |fi@

```

Repeat with the next line.

```

16339 ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

16340 |catcode`\^^I=13@
16341 |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.

```

16342 |catcode`\^^M=13@
16343 |def^^M##1^^M{@
16344 |def^^M###1^^M{@
16345 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
16346 ^^M}@
16347 ^^M}@

```

Reset the character categories back to the former state.

```

16348 |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.

```

16349 \ExplSyntaxOn
16350 \cs_new_protected:Npn
16351 \markdownLuaExecute
16352 #1
16353 {
16354 \int_compare:nNtT
16355 { \g_luabridge_method_int }
16356 =
16357 { \c_luabridge_method_shell_int }
16358 {
16359 \sys_if_shell_unrestricted:F
16360 {
16361 \sys_if_shell:TF

```

```

16362 {
16363 \msg_error:nn
16364 { markdown }
16365 { restricted-shell-access }
16366 }
16367 {
16368 \msg_error:nn
16369 { markdown }
16370 { disabled-shell-access }
16371 }
16372 }
16373 }
16374 \str_gset:NV
16375 \g_luabridge_output_dirname_str
16376 \markdownOptionOutputDir
16377 \luabridge_now:e
16378 { #1 }
16379 }
16380 \cs_generate_variant:Nn
16381 \msg_new:nnnn
16382 { nnnV }
16383 \tl_set:Nn
16384 \l_tmpa_tl
16385 {
16386 You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
16387 --enable-write18~flag,~or~write~shell_escape=t~in~the~
16388 texmf.cnf~file.
16389 }
16390 \msg_new:nnnV
16391 { markdown }
16392 { restricted-shell-access }
16393 { Shell~escape~is~restricted }
16394 \l_tmpa_tl
16395 \msg_new:nnnV
16396 { markdown }
16397 { disabled-shell-access }
16398 { Shell~escape~is~disabled }
16399 \l_tmpa_tl
16400 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

16401 \ExplSyntaxOn
16402 \tl_new:N
16403 \g_@@_after_markinline_tl
16404 \tl_gset:Nn

```

```

16405 \g_@@_after_markinline_tl
16406 { \unskip }
16407 \cs_new_protected:Npn
16408 \markinline
16409 {

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input markdown text as T<sub>E</sub>X code.

```

16410 \group_begin:
16411 \cctab_select:N
16412 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

16413 \@@_if_option:nF
16414 { frozenCache }
16415 {
16416 \immediate
16417 \openout
16418 \markdownOutputFileStream
16419 \markdownOptionInputTempFileName
16420 \relax
16421 \msg_info:nne
16422 { markdown }
16423 { buffering-markinline }
16424 { \markdownOptionInputTempFileName }
16425 }

```

Peek ahead and extract the inline markdown text.

```

16426 \peek_after:Nw
16427 \@@_inline_read_markdown_text:
16428 }
16429 \cs_new_protected:Nn
16430 \@@_inline_read_markdown_text:
16431 {

```

Allow both `{<markdown text>}` but also any other delimiter such as `|<markdown text>|`.

```

16432 \bool_if:nTF
16433 {
16434 \token_if_group_begin_p:N
16435 \l_peek_token ||
16436 \token_if_eq_charcode_p:NN
16437 \l_peek_token
16438 \c_group_begin_token
16439 }
16440 {
16441 \regex_set:Nn

```

```

16442 \l_tmpa_regex
16443 { { (.*) } }
16444 }
16445 {
16446 \tl_set:Nx
16447 \l_tmpa_tl
16448 {
16449 \text_purify:n
16450 { \l_peek_token }
16451 }
16452 \regex_set:Nn
16453 \l_tmpa_regex
16454 {
16455 \u { \l_tmpa_tl }
16456 (.*)
16457 \u { \l_tmpa_tl }
16458 }
16459 }
16460 \peek_regex_replace_once:NnF
16461 \l_tmpa_regex
16462 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

16463 \c { @@_if_option:nF }
16464 \cB { frozenCache \cE }
16465 \cB {
16466 \c { immediate }
16467 \c { write }
16468 \c { markdownOutputFileStream }
16469 \cB { \1 \cE }
16470 \c { immediate }
16471 \c { closeout }
16472 \c { markdownOutputFileStream }
16473 \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

16474 \c { group_end: }
16475 \c { group_begin: }
16476 \c { @@_setup:n }
16477 \cB { contentLevel = inline \cE }
16478 \c { markdownInput }
16479 \cB {
16480 \c { markdownOptionOutputDir } /
16481 \c { markdownOptionInputTempFileName }
16482 \cE }
16483 \c { group_end: }
16484 \c { tl_use:N }

```



```

16485 \c { g_@@_after_markinline_tl }
16486 }
16487 {
16488 \msg_error:nn
16489 { markdown }
16490 { markinline-peek-failure }
16491 \group_end:
16492 \tl_use:N
16493 \g_@@_after_markinline_tl
16494 }
16495 }
16496 \msg_new:nnn
16497 { markdown }
16498 { buffering-markinline }
16499 { Buffering~inline~markdown~input~into~
16500 the~temporary~input~file~"#1". }
16501 \msg_new:nnnn
16502 { markdown }
16503 { markinline-peek-failure }
16504 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
16505 { The~macro~should~be~followed~by~inline~
16506 markdown~text~in~curly~braces }
16507 \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.

```

16508 \ExplSyntaxOn
16509 \cs_new_protected:Npn
16510 \markdownInput
16511 #1
16512 {
16513 \@@_if_option:nTF
16514 { frozenCache }
16515 {
16516 \markdownInputRaw
16517 { #1 }
16518 }
16519 {

```

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`.

```

16520 \tl_set:Nx
16521 \l_tmpa_tl

```

```

16522 { #1 }
16523 \file_get_full_name:VNTF
16524 \l_tmpa_tl
16525 \l_tmpb_tl
16526 {
16527 \exp_args:NV
16528 \markdownInputRaw
16529 \l_tmpb_tl
16530 }
16531 {
16532 \msg_error:nnV
16533 { markdown }
16534 { markdown-file-does-not-exist }
16535 \l_tmpa_tl
16536 }
16537 }
16538 }
16539 \msg_new:nnn
16540 { markdown }
16541 { markdown-file-does-not-exist }
16542 {
16543 Markdown~file~#1~does~not~exist
16544 }
16545 \ExplSyntaxOff
16546 \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.

```

16547 \catcode`\|=0%
16548 \catcode`\|=12%
16549 \catcode`\&=6%
16550 \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.

```

16551 |begingroup
16552 |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.

```

16553 |catcode`\#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

16554 |markdownIfOption{frozenCache}{%

```

```

16555 |ifnum|markdownOptionFrozenCacheCounter=0|relax
16556 |markdownInfo{Reading frozen cache from
16557 "|markdownOptionFrozenCacheFileName"}%
16558 |input|markdownOptionFrozenCacheFileName|relax
16559 |fi
16560 |markdownInfo{Including markdown document number
16561 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
16562 |csname markdownFrozenCache%
16563 |the|markdownOptionFrozenCacheCounter|endcsname
16564 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
16565 }{%
16566 |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

16567 |openin|markdownInputFileStream{&1}%
16568 |closein|markdownInputFileStream
16569 |markdownPrepareLuaOptions
16570 |markdownPrepareInputFilename{&1}%
16571 |markdownLuaExecute{%
16572 |markdownPrepare
16573 |markdownConvert
16574 |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

16575 |markdownIfOption{finalizeCache}{%
16576 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
16577 }%
16578 |endgroup
16579 }%
16580 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of T<sub>E</sub>X to execute a T<sub>E</sub>X document in the middle of a markdown document fragment.

```

16581 \gdef\markdownEscape#1{%
16582 \catcode`\%=14\relax
16583 \catcode`\#=6\relax
16584 \input #1\relax
16585 \catcode`\%=12\relax
16586 \catcode`\#=12\relax
16587 }%

```

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implementation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [19, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```
16588 \def\markdownVersionSpace{ }%
16589 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
16590 \markdownVersion\markdownVersionSpace markdown renderer]%
```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```
16591 \ExplSyntaxOn
16592 \cs_gset_eq:NN
16593 \markinlinePlainTeX
16594 \markinline
16595 \cs_gset_protected:Npn
16596 \markinline
16597 {
16598 \peek_regex_replace_once:nn
16599 { (\[(.*) \]) ? }
16600 {
```

Apply the options locally.

```
16601 \c { group_begin: }
16602 \c { @@_setup:n }
16603 \cB { \2 \cE }
16604 \c { tl_put_right:Nn }
16605 \c { g_@@_after_markinline_tl }
16606 \cB { \c { group_end: } \cE }
16607 \c { markinlinePlainTeX }
16608 }
16609 }
16610 \ExplSyntaxOff
```

The `\markdownInputPlainTeX` macro is used to store the original plain T<sub>E</sub>X implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```
16611 \let\markdownInputPlainTeX\markdownInput
16612 \renewcommand\markdownInput[2][{}]{%
16613 \begingroup
16614 \markdownSetup{#1}%
16615 \markdownInputPlainTeX{#2}%
```

```

16616 \endgroup}%
16617 \renewcommand\yamlInput[2][]{%
16618 \begingroup
16619 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
16620 \markdownInputPlainTeX{#2}%
16621 \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

16622 \ExplSyntaxOn
16623 \renewenvironment
16624 { markdown }
16625 {

```

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.

```

16626 \group_begin:
16627 \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).

```

16628 \char_set_catcode_letter:n { 35 }

```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```

16629 \peek_regex_replace_once:nnF

```

```

16630 { \ *\[r*([~]*)\[^\r]* }
16631 {

```

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.

```

16632 \c { group_end: }
16633 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }

```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```

16634 \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}

```

```

16635 \c { exp_args:NV }
16636 \c { markdownReadAndConvert@ }
16637 \c { @currenvir }
16638 }
16639 {
16640 \group_end:
16641 \exp_args:NV
16642 \markdownReadAndConvert@
16643 \@currenvir
16644 }
16645 }
16646 { \markdownEnd }
16647 \renewenvironment
16648 { markdown* }
16649 [1]
16650 {
16651 \@@_if_option:nTF
16652 { experimental }
16653 {
16654 \msg_error:nn
16655 { markdown }
16656 { latex-markdown-star-deprecated }
16657 }

```

```

16658 {
16659 \msg_warning:nn
16660 { markdown }
16661 { latex-markdown-star-deprecated }
16662 }
16663 \@@_setup:n
16664 { #1 }
16665 \markdownReadAndConvert@
16666 { markdown* }
16667 }
16668 { \markdownEnd }
16669 \renewenvironment
16670 { yaml }
16671 {
16672 \group_begin:
16673 \yamlSetup
16674 { jekyllData, expectJekyllData, ensureJekyllData }
16675 \markdown
16676 }
16677 { \yamlEnd }
16678 \msg_new:nnn
16679 { markdown }
16680 { latex-markdown-star-deprecated }
16681 {
16682 The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
16683 be~removed~in~the~next~major~version~of~the~Markdown~package.
16684 }
16685 \cs_generate_variant:Nn % noqa: w402
16686 \@@_setup:n
16687 { V }
16688 \ExplSyntaxOff
16689 \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*.

```

16690 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
16691 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
16692 \gdef|markdownReadAndConvert@#1<%
16693 |markdownReadAndConvert<\end{#1}>%
16694 <|end<#1>>>%
16695 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\mathrm{T}_{\mathrm{E}}\mathrm{X}$  implementation of the theme-loading mechanism

from Section 3.2.2. Furthermore, this section also implements the built-in L<sup>A</sup>T<sub>E</sub>X themes provided with the Markdown package.

```

16696 \ExplSyntaxOn
16697 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
16698 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
16699 \cs_gset_protected:Nn
16700 \@@_load_theme:nnn
16701 {

```

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.

```

16702 \ifmarkdownLaTeXLoaded
16703 \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.

```

16704 \bool_if:nTF
16705 {
16706 \bool_lazy_or_p:nn
16707 {
16708 \prop_if_in_p:Nn
16709 \g_@@_latex_built_in_themes_prop
16710 { #1 }
16711 }
16712 {
16713 \file_if_exist_p:n
16714 { markdown theme #3.sty }
16715 }
16716 }
16717 {
16718 \msg_error:nn
16719 { markdown }
16720 { latex-theme-after-preamble }
16721 }

```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

16722 {
16723 \@@_plain_tex_load_theme:nnn
16724 { #1 }
16725 { #2 }
16726 { #3 }
16727 }
16728 \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.



```

16729 \bool_if:nTF
16730 {
16731 \bool_lazy_or_p:nn
16732 {
16733 \prop_if_in_p:Nn
16734 \g_@@_latex_built_in_themes_prop
16735 { #1 }
16736 }
16737 {
16738 \file_if_exist_p:n
16739 { markdown theme #3.sty }
16740 }
16741 }
16742 {
16743 \prop_get:NnNTF
16744 \g_@@_latex_loaded_themes_linenos_prop
16745 { #1 }
16746 \l_tmpa_tl
16747 {
16748 \prop_get:NnN
16749 \g_@@_latex_loaded_themes_versions_prop
16750 { #1 }
16751 \l_tmpb_tl
16752 \str_if_eq:nVTF
16753 { #2 }
16754 \l_tmpb_tl
16755 {
16756 \msg_warning:nnnVn
16757 { markdown }
16758 { repeatedly-loaded-latex-theme }
16759 { #1 }
16760 \l_tmpa_tl
16761 { #2 }
16762 }
16763 {
16764 \msg_error:nnnnVV
16765 { markdown }
16766 { different-versions-of-latex-theme }
16767 { #1 }
16768 { #2 }
16769 \l_tmpb_tl
16770 \l_tmpa_tl
16771 }
16772 }
16773 }
16774 \prop_gput:Nnx
16775 \g_@@_latex_loaded_themes_linenos_prop

```

```

16776 { #1 }
16777 { \tex_the:D \tex_inputlineno:D } % noqa: W200
16778 \prop_gput:Nnn
16779 \g_@@_latex_loaded_themes_versions_prop
16780 { #1 }
16781 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

16782 \prop_if_in:NnTF
16783 \g_@@_latex_built_in_themes_prop
16784 { #1 }
16785 {
16786 \msg_info:nnnn
16787 { markdown }
16788 { loading-built-in-latex-theme }
16789 { #1 }
16790 { #2 }
16791 \prop_item:Nn
16792 \g_@@_latex_built_in_themes_prop
16793 { #1 }
16794 }
16795 {
16796 \msg_info:nnnn
16797 { markdown }
16798 { loading-latex-theme }
16799 { #1 }
16800 { #2 }
16801 \RequirePackage
16802 { markdown theme #3 }
16803 }
16804 }
16805 }
16806 {
16807 \@@_plain_tex_load_theme:nnn
16808 { #1 }
16809 { #2 }
16810 { #3 }
16811 }
16812 \fi
16813 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

16814 \msg_info:nnnn
16815 { markdown }
16816 { theme-loading-postponed }
16817 { #1 }

```

```

16818 { #2 }
16819 \AtEndOfPackage
16820 {
16821 \@@_set_theme:n
16822 { #1 @ #2 }
16823 }
16824 \fi
16825 }
16826 \msg_new:nnn
16827 { markdown }
16828 { theme-loading-postponed }
16829 {
16830 Postponing~loading~version~#2~of~Markdown~theme~#1~until~
16831 Markdown~package~has~finished~loading
16832 }
16833 \msg_new:nnn
16834 { markdown }
16835 { loading-built-in-latex-theme }
16836 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
16837 \msg_new:nnn
16838 { markdown }
16839 { loading-latex-theme }
16840 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
16841 \msg_new:nnn
16842 { markdown }
16843 { repeatedly-loaded-latex-theme }
16844 {
16845 Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
16846 loaded~on~line~#2,~not~loading~it~again
16847 }
16848 \msg_new:nnn
16849 { markdown }
16850 { different-versions-of-latex-theme }
16851 {
16852 Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
16853 but~version~#3~has~already~been~loaded~on~line~#4
16854 }
16855 \cs_generate_variant:Nn
16856 \msg_new:nnnn
16857 { nnVV }
16858 \tl_set:Nn
16859 \l_tmpa_tl
16860 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
16861 \tl_put_right:NV
16862 \l_tmpa_tl
16863 \c_backslash_str
16864 \tl_put_right:Nn

```

```

16865 \l_tmpa_tl
16866 { begin { document } }
16867 \tl_set:Nn
16868 \l_tmpb_tl
16869 { Load~Markdown~theme~#1~before~ }
16870 \tl_put_right:NV
16871 \l_tmpb_tl
16872 \c_backslash_str
16873 \tl_put_right:Nn
16874 \l_tmpb_tl
16875 { begin { document } }
16876 \msg_new:nnVV
16877 { markdown }
16878 { latex-theme-after-preamble }
16879 \l_tmpa_tl
16880 \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.

```

16881 \tl_set:Nn
16882 \l_tmpa_tl
16883 {
16884 \RequirePackage
16885 { graphicx }
16886 \markdownLoadPlainTeXTheme
16887 }
16888 \prop_gput:NnV
16889 \g_@@_latex_built_in_themes_prop
16890 { witiko / dot }
16891 \l_tmpa_tl
16892 \prop_gput:NnV
16893 \g_@@_latex_built_in_themes_prop
16894 { witiko / graphicx / http }
16895 \l_tmpa_tl

```

The [witiko/glossaries](#) theme first checks that the version is either `v1` or `latest`, issuing a warning for the latter.

```

16896 \prop_gput:Nnn
16897 \g_@@_latex_built_in_themes_prop
16898 { witiko / glossaries }
16899 {
16900 \str_case:enF
16901 { \markdownThemeVersion }
16902 {
16903 { latest }
16904 {
16905 \msg_warning:nnnn
16906 { markdown }

```

```

16907 { pin-theme-version }
16908 { witiko/glossaries }
16909 { v1 }
16910 }
16911 { v1 }
16912 { }
16913 }
16914 {
16915 \msg_error:nnen
16916 { markdown }
16917 { unknown-theme-version }
16918 { witiko/glossaries }
16919 { \markdownThemeVersion }
16920 { v1 }
16921 }
16922 }
16923 \ExplSyntaxOff

```

In Section 3.3.3, we'll continue the implementation of the theme by defining the snippet `witiko/glossaries/import-acronyms`. 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.

```

16924 \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 3.3.5 for the actual definitions.

### 3.3.3 Snippets

This section defines the built-in L<sup>A</sup>T<sub>E</sub>X snippet `witiko/glossaries/import-acronyms`.

```

16925 \ExplSyntaxOn
16926 \prop_get:NnN
16927 \g_@@_latex_built_in_themes_prop
16928 { witiko / glossaries }
16929 \l_tmpa_tl
16930 \tl_put_right:Nn
16931 \l_tmpa_tl
16932 {
16933 \cs_new:Nn
16934 \@@_register_acronym:n
16935 {
16936 \@@_setup:n
16937 {

```

When we register an acronym with the `acronyms` option, enclose it in an extra set of braces in case the acronym contained a comma.

```

16938 acronym += {{ #1 }},
16939 }
16940 }

```

```

16941 \cs_generate_variant:Nn
16942 \@@_register_acronym:n
16943 { V }
16944 \prop_new:N
16945 \g_@@_acronyms_to_labels_prop
16946 \tl_new:N
16947 \l_@@_acronym_short_tl

```

When we import an acronym from the glossaries package, we receive a  $\langle label \rangle$ , which doesn't correspond to an acronym. Therefore, we register the short form of the corresponding acronym instead and keep a mapping from these short forms back to the labels when we need to display an acronym in the markdown text.

```

16948 \cs_new:Nn
16949 \@@_register_acronym_label:n
16950 {
16951 \tl_set:Nx
16952 \l_@@_acronym_short_tl
16953 { \glsentryshort { #1 } }

```

Be idempotent and only import each acronym once.

```

16954 \prop_if_in:NVF
16955 \g_@@_acronyms_to_labels_prop
16956 \l_@@_acronym_short_tl
16957 {
16958 \@@_register_acronym:V
16959 \l_@@_acronym_short_tl
16960 \prop_gput:NVn
16961 \g_@@_acronyms_to_labels_prop
16962 \l_@@_acronym_short_tl
16963 { #1 }
16964 }
16965 }
16966 \cs_generate_variant:Nn
16967 \@@_register_acronym_label:n
16968 { V }

```

When we need to display an acronym in the markdown text, first consult our mapping to see if it has been imported from the glossaries package.

```

16969 \tl_new:N
16970 \l_@@_acronym_label_tl
16971 \cs_new:Nn
16972 \@@_print_acronym:n
16973 {
16974 \prop_get:NnNTF
16975 \g_@@_acronyms_to_labels_prop
16976 { #1 }
16977 \l_@@_acronym_label_tl
16978 {

```

If it has, retrieve the  $\langle label \rangle$  and display the acronym using the glossaries package, linking it to the corresponding glossary entry.

```

16979 \group_begin:
16980 \cs_set_eq:NN
16981 \acronymfont
16982 \markdownRendererAcronymPrototype
16983 \exp_args:NV
16984 \acrshort
16985 \l_@@_acronym_label_tl
16986 \group_end:
16987 }
16988 {

```

Otherwise, just format the acronym using the current renderer prototype.

```

16989 \markdownRendererAcronymPrototype
16990 { #1 }
16991 }
16992 }
16993 \cs_generate_variant:Nn
16994 \@@_print_acronym:n
16995 { x }

```

In the snippet [witiko/glossaries/import-acronyms](#), first import all acronyms from the glossaries package and register them with the [acronyms](#) option.

```

16996 \markdownSetupSnippet
16997 { import-acronyms }
16998 {
16999 code = {
17000 \forlseqentries
17001 [acronym]
17002 \l_@@_acronym_label_tl
17003 {
17004 \@@_register_acronym_label:V
17005 \l_@@_acronym_label_tl
17006 }

```

Then, redefine the corresponding token renderers to format acronyms that appear in the markdown text.

```

17007 \@@_setup:n
17008 {
17009 renderers = {
17010 acronym = {

```

Fold nested acronyms.

```

17011 \group_begin:
17012 \markdownSetup {
17013 unprotectedRenderers = {
17014 acronym = { ##1 },

```

```

17015 }
17016 }
17017 \@@_print_acronym:x
17018 { #1 }
17019 \group_end:
17020 }
17021 }
17022 }
17023 }
17024 }
17025 }
17026 \prop_gput:NnV
17027 \g_@@_latex_built_in_themes_prop
17028 { witiko / glossaries }
17029 \l_tmpa_tl
17030 \ExplSyntaxOff

```

### 3.3.4 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

17031 \DeclareOption*{%
17032 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
17033 \ProcessOptions\relax

```

### 3.3.5 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.

```

17034 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

#### 3.3.5.1 Acronyms

If the `acronyms` option is non-empty, render acronyms, initialisms, and other all-caps sequences using small caps.

```

17035 \ExplSyntaxOn
17036 \prop_new_linked:N
17037 \g_@@_recorded_acronyms_prop
17038 \bool_new:N
17039 \g_@@_record_acronyms_bool
17040 \bool_gset_false:N
17041 \g_@@_record_acronyms_bool
17042 \cs_new_protected:Nn
17043 \@@_record_acronym:n
17044 {

```

For explicitly listed acronyms using the HTML class `acronym` that only contain simple text and not control sequences after cleaning up any potential nested acronyms,



record them in the auxiliary file, so that they are applied throughout the document in the next compilation run.

```

17045 \bool_if:NT
17046 \g_@@_record_acronyms_bool
17047 {
17048 \tl_set:Nn
17049 \l_tmpa_tl
17050 { #1 }
17051 \regex_replace_all:nnN
17052 { \c { markdownRendererAcronym } | \cB\{ | \cE\} }
17053 { }
17054 \l_tmpa_tl
17055 \prop_if_in:NVF
17056 \g_@@_recorded_acronyms_prop
17057 \l_tmpa_tl
17058 {
17059 \regex_if_match:nVF
17060 { \cC. }
17061 \l_tmpa_tl
17062 {
17063 \prop_gput:Nvn
17064 \g_@@_recorded_acronyms_prop
17065 \l_tmpa_tl
17066 { true }
17067 \tl_gput_right:Nx
17068 \markdownOptionAcronyms
17069 { , { \tl_use:N \l_tmpa_tl } }
17070 \tl_set:Nn
17071 \l_tmpb_tl
17072 {
17073 \ExplSyntaxOn
17074 \tl_gput_right:Nn
17075 \markdownOptionAcronyms
17076 }
17077 \tl_gput_right:Nx
17078 \l_tmpb_tl
17079 { { , { \tl_use:N \l_tmpa_tl } } }
17080 \tl_gput_right:Nx
17081 \l_tmpb_tl
17082 { \ExplSyntaxOff }
17083 \iow_now:NV
17084 \@auxout
17085 \l_tmpb_tl
17086 }
17087 }
17088 }
17089 }

```

```

17090 \int_new:N
17091 \g_@@_heading_nesting_depth_int
17092 \markdownSetup {
17093 rendererPrototypes = {
17094 acronym = {

```

Prevent acronyms in headings.

```

17095 \int_if_zero:nTF
17096 { \g_@@_heading_nesting_depth_int }
17097 {

```

Prevent nested acronyms.

```

17098 \group_begin:
17099 \markdownSetup {
17100 unprotectedRenderers = {
17101 acronym = { ##1 },
17102 }
17103 }
17104 \@@_record_acronym:n
17105 { #1 }
17106 \tl_set:Nn
17107 \l_tmpa_tl
17108 { #1 }

```

Render all characters that are neither digits nor ASCII lower case characters in small caps. Ideally, we would also be able to exclude Unicode lower-case characters but that doesn't seem possible with expl3 at the moment [20, Chapter 34 (The l3unicode module)].

```

17109 \regex_replace_all:nnN
17110 { [^\d a-z]+ }
17111 {
17112 \c{ textsc } \cB\{
17113 \c{ MakeLowercase } \cB\{ \0 \cE\}
17114 \cE\}
17115 }
17116 \l_tmpa_tl

```

Furthermore, if the current typeface supports it, render all digits using old-style numerals.

```

17117 \cs_if_exist:NT
17118 \oldstylenums
17119 {
17120 \regex_replace_all:nnN
17121 { \d+ }
17122 { \c{ oldstylenums } \cB\{ \0 \cE\} }
17123 \l_tmpa_tl
17124 }

```

If the package microtype is loaded, letter-space the small caps.

```

17125 \@ifpackageloaded
17126 { microtype }
17127 {
17128 \textls
17129 {
17130 \tl_use:N
17131 \l_tmpa_tl
17132 }
17133 }
17134 {
17135 \tl_use:N
17136 \l_tmpa_tl
17137 }
17138 \group_end:
17139 }

```

Prevent acronyms in headings.

```

17140 { #1 }
17141 },
17142 },
17143 }

```

If the package glossaries is loaded with the option `acronym`, use the built-in snippet `LATEX witiko/glossaries/import-acronyms`.

```

17144 \@ifpackageloaded
17145 { glossaries }
17146 {
17147 \cs_if_exist:NT
17148 \@gls@do@acronymsdef
17149 {
17150 \markdownSetup {
17151 import = /witiko/glossaries@v1,
17152 snippet = /witiko/glossaries/import-acronyms,
17153 }
17154 }
17155 }
17156 { }

```

### 3.3.5.2 Lists

If either the `tightLists` or the `fancyLists` 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`.

```

17157 \bool_new:N

```

```

17158 \g_@@_tight_or_fancy_lists_bool
17159 \bool_gset_false:N
17160 \g_@@_tight_or_fancy_lists_bool
17161 \@@_if_option:nTF
17162 { tightLists }
17163 {
17164 \bool_gset_true:N
17165 \g_@@_tight_or_fancy_lists_bool
17166 }
17167 {
17168 \@@_if_option:nT
17169 { fancyLists }
17170 {
17171 \bool_gset_true:N
17172 \g_@@_tight_or_fancy_lists_bool
17173 }
17174 }
17175 \bool_new:N
17176 \g_@@_beamer_paralist_or_enumitem_bool
17177 \bool_gset_true:N
17178 \g_@@_beamer_paralist_or_enumitem_bool
17179 \@ifclassloaded
17180 { beamer }
17181 { }
17182 {
17183 \@ifpackageloaded
17184 { paralist }
17185 { }
17186 {
17187 \@ifpackageloaded
17188 { enumitem }
17189 { }
17190 {
17191 \bool_gset_false:N
17192 \g_@@_beamer_paralist_or_enumitem_bool
17193 }
17194 }
17195 }
17196 \bool_if:nT
17197 {
17198 \g_@@_tight_or_fancy_lists_bool &&
17199 ! \g_@@_beamer_paralist_or_enumitem_bool
17200 }
17201 {
17202 \str_if_eq:enTF
17203 { \markdownThemeVersion }
17204 { experimental }

```

```

17205 {
17206 \RequirePackage
17207 { enumitem }
17208 }
17209 {
17210 \IfDocumentMetadataTF
17211 {
17212 \RequirePackage
17213 { enumitem }
17214 }
17215 {
17216 \RequirePackage
17217 { paralist }
17218 }
17219 }
17220 }
17221 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

17222 \ExplSyntaxOn
17223 \cs_new:Nn
17224 \@@_latex_fancy_list_item_label_number:nn
17225 {
17226 \str_case:nn
17227 { #1 }
17228 {
17229 { Decimal } { #2 }
17230 { LowerRoman } { \int_to_roman:n { #2 } }
17231 { UpperRoman } { \int_to_Roman:n { #2 } }
17232 { LowerAlpha } { \int_to_alph:n { #2 } }
17233 { UpperAlpha } { \int_to_Alph:n { #2 } }
17234 }
17235 }
17236 \cs_new:Nn
17237 \@@_latex_fancy_list_item_label_delimiter:n
17238 {
17239 \str_case:nn
17240 { #1 }
17241 {
17242 { Default } { . }
17243 { OneParen } {) }
17244 { Period } { . }
17245 }
17246 }
17247 \cs_new:Nn
17248 \@@_latex_fancy_list_item_label:nnn

```

```

17249 {
17250 \@@_latex_fancy_list_item_label_number:nn
17251 { #1 }
17252 { #3 }
17253 \@@_latex_fancy_list_item_label_delimiter:n
17254 { #2 }
17255 }
17256 \cs_generate_variant:Nn
17257 \@@_latex_fancy_list_item_label:nnn
17258 { VVn }
17259 \tl_new:N
17260 \l_@@_latex_fancy_list_item_label_number_style_tl
17261 \tl_new:N
17262 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17263 \ifpackageloaded { enumitem } {
17264 \markdownSetup { rendererPrototypes = {

```

First, let's define the tight list item renderer prototypes.

```

17265 ulBeginTight = {
17266 \begin
17267 { itemize }
17268 [noitemsep]
17269 },
17270 ulEndTight = {
17271 \end
17272 { itemize }
17273 },
17274 olBeginTight = {
17275 \begin
17276 { enumerate }
17277 [noitemsep]
17278 },
17279 olEndTight = {
17280 \end
17281 { enumerate }
17282 },
17283 dlBeginTight = {
17284 \begin
17285 { description }
17286 [noitemsep]
17287 },
17288 dlEndTight = {
17289 \end
17290 { description }
17291 },

```

Second, let's define the fancy list item renderer prototypes.

```

17292 fancyOlBegin = {

```

```

17293 \group_begin:
17294 \tl_set:Nn
17295 \l_@@_latex_fancy_list_item_label_number_style_tl
17296 { #1 }
17297 \tl_set:Nn
17298 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17299 { #2 }
17300 \begin
17301 { enumerate }
17302 },
17303 fancyOlBeginTight = {
17304 \group_begin:
17305 \tl_set:Nn
17306 \l_@@_latex_fancy_list_item_label_number_style_tl
17307 { #1 }
17308 \tl_set:Nn
17309 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17310 { #2 }
17311 \begin
17312 { enumerate }
17313 [noitemsep]
17314 },
17315 fancyOlEnd(|Tight) = {
17316 \end { enumerate }
17317 \group_end:
17318 },
17319 fancyOlItemWithNumber = {
17320 \item
17321 [
17322 \@@_latex_fancy_list_item_label:VVn
17323 \l_@@_latex_fancy_list_item_label_number_style_tl
17324 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17325 { #1 }
17326]
17327 },
17328 } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

17329 }
17330 { \ifpackageloaded { paralist } {
17331 \markdownSetup { rendererPrototypes = {

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

17332 ulBeginTight = {
17333 \group_begin:
17334 \pltopsep=\topsep

```

```

17335 \plpartopsep=\partopsep
17336 \begin { compactitem }
17337 },
17338 ulEndTight = {
17339 \end { compactitem }
17340 \group_end:
17341 },
17342 fancyOlBegin = {
17343 \group_begin:
17344 \tl_set:Nn
17345 \l_@@_latex_fancy_list_item_label_number_style_tl
17346 { #1 }
17347 \tl_set:Nn
17348 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17349 { #2 }
17350 \begin { enumerate }
17351 },
17352 fancyOlEnd = {
17353 \end { enumerate }
17354 \group_end:
17355 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

17356 olBeginTight = {
17357 \group_begin:
17358 \plpartopsep=\partopsep
17359 \pltopsep=\topsep
17360 \begin { compactenum }
17361 },
17362 olEndTight = {
17363 \end { compactenum }
17364 \group_end:
17365 },
17366 fancyOlBeginTight = {
17367 \group_begin:
17368 \tl_set:Nn
17369 \l_@@_latex_fancy_list_item_label_number_style_tl
17370 { #1 }
17371 \tl_set:Nn
17372 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17373 { #2 }
17374 \plpartopsep=\partopsep
17375 \pltopsep=\topsep
17376 \begin { compactenum }
17377 },
17378 fancyOlEndTight = {

```



```

17379 \end { compactenum }
17380 \group_end:
17381 },
17382 fancyOllItemWithNumber = {
17383 \item
17384 [
17385 \@@_latex_fancy_list_item_label:VVn
17386 \l_@@_latex_fancy_list_item_label_number_style_tl
17387 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
17388 { #1 }
17389]
17390 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

17391 dlBeginTight = {
17392 \group_begin:
17393 \plpartopsep=\partopsep
17394 \pltopsep=\topsep
17395 \begin { compactdesc }
17396 },
17397 dlEndTight = {
17398 \end { compactdesc }
17399 \group_end:
17400 }
17401 } }
17402 }
17403 {

```

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.

```

17404 \markdownSetup
17405 {
17406 rendererPrototypes = {
17407 ulBeginTight = \markdownRendererUlBegin,
17408 ulEndTight = \markdownRendererUlEnd,
17409 fancyOlBegin = \markdownRendererOlBegin,
17410 fancyOlEnd = \markdownRendererOlEnd,
17411 olBeginTight = \markdownRendererOlBegin,
17412 olEndTight = \markdownRendererOlEnd,
17413 fancyOlBeginTight = \markdownRendererOlBegin,
17414 fancyOlEndTight = \markdownRendererOlEnd,
17415 dlBeginTight = \markdownRendererDlBegin,
17416 dlEndTight = \markdownRendererDlEnd,
17417 },
17418 }
17419 } }

```

```

17420 \ExplSyntaxOff
17421 \RequirePackage{amsmath}

Unless the unicode-math package has been loaded, load the amssymb package with
symbols to be used for tickboxes.

17422 \@ifpackageloaded{unicode-math}{
17423 \markdownSetup{rendererPrototypes={
17424 untickedBox = {\mdlgwhtsquare$},
17425 }}
17426 }{
17427 \RequirePackage{amssymb}
17428 \markdownSetup{rendererPrototypes={
17429 untickedBox = {\square$},
17430 }}
17431 }
17432 \RequirePackage{csvsimple}
17433 \RequirePackage{fancyvrb}
17434 \RequirePackage{graphicx}
17435 \markdownSetup{rendererPrototypes={
17436 hardLineBreak = {\},
17437 leftBrace = {\textbraceleft},
17438 rightBrace = {\textbraceright},
17439 dollarSign = {\textdollar},
17440 underscore = {\textunderscore},
17441 circumflex = {\textasciicircum},
17442 backslash = {\textbackslash},
17443 tilde = {\textasciitilde},
17444 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>41</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.

```

17445 codeSpan = {%
17446 \ifmmode
17447 \text{#1}%
17448 \else
17449 \texttt{#1}%
17450 \fi
17451 }}}

```

---

<sup>41</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.

### 3.3.5.3 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.

```
17452 \ExplSyntaxOn
17453 \markdownSetup{
17454 rendererPrototypes = {
17455 contentBlock = {
17456 \str_case:nnF
17457 { #1 }
17458 {
17459 { csv }
17460 {
17461 \begin { table }
17462 \begin { center }
17463 \csvautotabular { #3 }
17464 \end { center }
17465 \tl_if_empty:nF
17466 { #4 }
17467 { \caption { #4 } }
17468 \end { table }
17469 }
17470 { html }
17471 {
```

If we are using  $\text{\TeX}4\text{ht}$ <sup>42</sup>, we will pass HTML elements to the output HTML document unchanged.

```
17472 \cs_if_exist:NTF
17473 \HCode
17474 {
17475 \if_mode_vertical:
17476 \IgnorePar
17477 \fi:
17478 \EndP
17479 \special
17480 { t4ht* < #3 }
17481 \par
17482 \ShowPar
17483 }
17484 {
17485 \@@_luaxml_print_html:n
17486 { #3 }
17487 }
```

---

<sup>42</sup>See <https://tug.org/tex4ht/>.

```

17488 }
17489 { tex }
17490 {
17491 \markdownEscape
17492 { #3 }
17493 }
17494 }
17495 {
17496 \markdownInput
17497 { #3 }
17498 }
17499 },
17500 },
17501 }
17502 \ExplSyntaxOff
17503 \markdownSetup{rendererPrototypes={
17504 ulBegin = {\begin{itemize}},
17505 ulEnd = {\end{itemize}},
17506 olBegin = {\begin{enumerate}},
17507 olItem = {\item{}},
17508 olItemWithNumber = {\item[#1.]},
17509 olEnd = {\end{enumerate}},
17510 dlBegin = {\begin{description}},
17511 dlItem = {\item[#1]},
17512 dlEnd = {\end{description}},
17513 emphasis = {\emph{#1}},
17514 tickedBox = {\boxtimes},
17515 halfTickedBox = {\boxdot}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

17516 \ExplSyntaxOn
17517 \seq_new:N
17518 \g_@@_header_identifiers_seq
17519 \markdownSetup
17520 {
17521 rendererPrototypes = {
17522 headerAttributeContextBegin = {
17523 \markdownSetup
17524 {
17525 rendererPrototypes = {
17526 attributeIdentifier = {
17527 \seq_gput_right:Nn
17528 \g_@@_header_identifiers_seq
17529 { ##1 }
17530 },
17531 },
17532 }
17533 },

```

```

17534 headerAttributeContextEnd = {
17535 \seq_map_inline:Nn
17536 \g_@@_header_identifiers_seq
17537 { \label { ##1 } }
17538 \seq_gclear:N
17539 \g_@@_header_identifiers_seq
17540 },
17541 },
17542 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

17543 \bool_new:N
17544 \l_@@_header_unnumbered_bool
17545 \markdownSetup
17546 {
17547 rendererPrototypes = {
17548 headerAttributeContextBegin += {
17549 \markdownSetup
17550 {
17551 rendererPrototypes = {
17552 attributeClassName = {
17553 \bool_if:nT
17554 {
17555 \str_if_eq_p:nn
17556 { ##1 }
17557 { unnumbered } &&
17558 ! \l_@@_header_unnumbered_bool
17559 }
17560 {
17561 \group_begin:
17562 \bool_set_true:N
17563 \l_@@_header_unnumbered_bool
17564 \c@secnumdepth = -2
17565 \markdownSetup
17566 {
17567 rendererPrototypes = {
17568 sectionBegin = {
17569 \group_begin:
17570 },
17571 sectionEnd = {
17572 \group_end:
17573 },
17574 },
17575 }
17576 }
17577 },

```

```

17578 },
17579 }
17580 },
17581 },
17582 }
17583 \ExplSyntaxOff
17584 \markdownSetup{rendererPrototypes={
17585 superscript = {#1},
17586 subscript = {\textsubscript{#1}},
17587 blockQuoteBegin = {\begin{quotation}},
17588 blockQuoteEnd = {\end{quotation}},
17589 inputVerbatim = {\VerbatimInput{#1}},
17590 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
17591 note = {\footnote{#1}}}}

```

### 3.3.5.4 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

17592 \RequirePackage{ltxcmds}
17593 \ExplSyntaxOn
17594 \cs_gset_protected:Npn
17595 \markdownRendererInputFencedCodePrototype#1#2#3
17596 {
17597 \tl_if_empty:nTF
17598 { #2 }
17599 { \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.

```

17600 {
17601 \regex_extract_once:nnN
17602 { \w* }
17603 { #2 }
17604 \l_tmpa_seq
17605 \seq_pop_left:NN
17606 \l_tmpa_seq
17607 \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

17608 \ltx@ifpackageloaded
17609 { minted }
17610 {
17611 \catcode`\%=14\relax
17612 \catcode`\#=6\relax
17613 \exp_args:NV
17614 \inputminted
17615 \l_tmpa_tl
17616 { #1 }

```

```

17617 \catcode`\%=12\relax
17618 \catcode`\#=12\relax
17619 }
17620 {

```

When the listings package is loaded, use it for syntax highlighting.

```

17621 \ltx@ifpackageloaded
17622 { listings }
17623 { \lstinputlisting [language = \l_tmpa_tl] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

17624 { \markdownRendererInputFencedCode { #1 } { } { } }
17625 }
17626 }
17627 }
17628 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

17629 \ExplSyntaxOn
17630 \cs_new_protected:Npn
17631 \markdownLATEXStrongEmphasis #1
17632 {
17633 \str_if_in:NnTF
17634 \f@series
17635 { b }
17636 { \textnormal{#1} }
17637 { \textbf{#1} }
17638 }
17639 \ExplSyntaxOff
17640 \markdownSetup{rendererPrototypes={strongEmphasis={%
17641 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

17642 \@ifundefined{chapter}{%
17643 \markdownSetup{rendererPrototypes = {
17644 headingOne = {\section{#1}},
17645 headingTwo = {\subsection{#1}},
17646 headingThree = {\subsubsection{#1}},
17647 headingFour = {\paragraph{#1}},
17648 headingFive = {\subparagraph{#1}}}}
17649 }{%
17650 \markdownSetup{rendererPrototypes = {
17651 headingOne = {\chapter{#1}},
17652 headingTwo = {\section{#1}},
17653 headingThree = {\subsection{#1}},
17654 headingFour = {\subsubsection{#1}},
17655 headingFive = {\paragraph{#1}},
17656 headingSix = {\subparagraph{#1}}}}

```

```

17657 }%
Prevent acronyms in headings.
17658 \ExplSyntaxOn
17659 \markdownSetup {
17660 rendererPrototypes = {
17661 heading* ^= {
17662 \int_gincr:N
17663 \g_@@_heading_nesting_depth_int
17664 },
17665 heading* $= {
17666 \int_gdecr:N
17667 \g_@@_heading_nesting_depth_int
17668 }
17669 }
17670 }
17671 \ExplSyntaxOff

```

### 3.3.5.5 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

17672 \markdownSetup{
17673 rendererPrototypes = {
17674 ulItem = {%
17675 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
17676 },
17677 },
17678 }
17679 \def\markdownLaTeXUItem{%
17680 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
17681 \item[\markdownLaTeXCheckbox]%
17682 \expandafter\@gobble
17683 \else
17684 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
17685 \item[\markdownLaTeXCheckbox]%
17686 \expandafter\expandafter\expandafter\@gobble
17687 \else
17688 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
17689 \item[\markdownLaTeXCheckbox]%
17690 \expandafter\expandafter\expandafter\expandafter
17691 \expandafter\expandafter\expandafter\@gobble
17692 \else
17693 \item{}%
17694 \fi
17695 \fi
17696 \fi
17697 }

```



### 3.3.5.6 html elements

If the `html` option is enabled and we are using  $\text{\TeX 4ht}$ <sup>43</sup>, we will pass HTML elements to the output HTML document unchanged.

```
17698 \@ifundefined{HCode}{\}{
17699 \markdownSetup{
17700 rendererPrototypes = {
17701 inlineHtmlTag = {%
17702 \ifvmode
17703 \IgnorePar
17704 \EndP
17705 \fi
17706 \HCode{#1}%
17707 },
17708 inputBlockHtmlElement = {%
17709 \ifvmode
17710 \IgnorePar
17711 \fi
17712 \EndP
17713 \special{t4ht*<#1}%
17714 \par
17715 \ShowPar
17716 },
17717 },
17718 }
17719 }
```

### 3.3.5.7 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 `\citete` macros, and the `Bib $\text{\LaTeX}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
17720 \newcount\markdownLaTeXCitationsCounter
17721
17722 % Basic implementation
17723 \long\def@gobblethree#1#2#3{}%
17724 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
17725 \advance\markdownLaTeXCitationsCounter by 1\relax
17726 \ifx\relax#4\relax
17727 \ifx\relax#5\relax
17728 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17729 \relax
17730 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
17731 \expandafter\expandafter\expandafter
17732 \expandafter\expandafter\expandafter\expandafter
```

---

<sup>43</sup>See <https://tug.org/tex4ht/>.

```

17733 \@gobblethree
17734 \fi
17735 \else% Before a postnote (#5), dump the accumulator
17736 \ifx\relax#1\relax\else
17737 \cite{#1}%
17738 \fi
17739 \cite[#5]{#6}%
17740 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17741 \relax
17742 \else
17743 \expandafter\expandafter\expandafter
17744 \expandafter\expandafter\expandafter\expandafter
17745 \expandafter\expandafter\expandafter
17746 \expandafter\expandafter\expandafter\expandafter
17747 \markdownLaTeXBasicCitations
17748 \fi
17749 \expandafter\expandafter\expandafter
17750 \expandafter\expandafter\expandafter\expandafter{%
17751 \expandafter\expandafter\expandafter
17752 \expandafter\expandafter\expandafter\expandafter}%
17753 \expandafter\expandafter\expandafter
17754 \expandafter\expandafter\expandafter\expandafter{%
17755 \expandafter\expandafter\expandafter
17756 \expandafter\expandafter\expandafter\expandafter}%
17757 \expandafter\expandafter\expandafter
17758 \@gobblethree
17759 \fi
17760 \else% Before a prenote (#4), dump the accumulator
17761 \ifx\relax#1\relax\else
17762 \cite{#1}%
17763 \fi
17764 \ifnum\markdownLaTeXCitationsCounter>1\relax
17765 \space % Insert a space before the prenote in later citations
17766 \fi
17767 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
17768 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17769 \relax
17770 \else
17771 \expandafter\expandafter\expandafter
17772 \expandafter\expandafter\expandafter\expandafter
17773 \markdownLaTeXBasicCitations
17774 \fi
17775 \expandafter\expandafter\expandafter{%
17776 \expandafter\expandafter\expandafter}%
17777 \expandafter\expandafter\expandafter{%
17778 \expandafter\expandafter\expandafter}%
17779 \expandafter

```

```

17780 \gobblethree
17781 \fi\markdownLaTeXBasicCitations{#1#2#6},}
17782 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
17783
17784 % Natbib implementation
17785 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
17786 \advance\markdownLaTeXCitationsCounter by 1\relax
17787 \ifx\relax#3\relax
17788 \ifx\relax#4\relax
17789 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17790 \relax
17791 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
17792 \expandafter\expandafter\expandafter
17793 \expandafter\expandafter\expandafter\expandafter
17794 \@gobbletwo
17795 \fi
17796 \else% Before a postnote (#4), dump the accumulator
17797 \ifx\relax#1\relax\else
17798 \citep{#1}%
17799 \fi
17800 \citep[] [#4]{#5}%
17801 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17802 \relax
17803 \else
17804 \expandafter\expandafter\expandafter
17805 \expandafter\expandafter\expandafter\expandafter
17806 \expandafter\expandafter\expandafter
17807 \expandafter\expandafter\expandafter\expandafter
17808 \markdownLaTeXNatbibCitations
17809 \fi
17810 \expandafter\expandafter\expandafter
17811 \expandafter\expandafter\expandafter\expandafter{%
17812 \expandafter\expandafter\expandafter
17813 \expandafter\expandafter\expandafter\expandafter}%
17814 \expandafter\expandafter\expandafter
17815 \@gobbletwo
17816 \fi
17817 \else% Before a prenote (#3), dump the accumulator
17818 \ifx\relax#1\relax\relax\else
17819 \citep{#1}%
17820 \fi
17821 \citep[#3] [#4]{#5}%
17822 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17823 \relax
17824 \else
17825 \expandafter\expandafter\expandafter
17826 \expandafter\expandafter\expandafter\expandafter

```

```

17827 \markdownLaTeXNatbibCitations
17828 \fi
17829 \expandafter\expandafter\expandafter{%
17830 \expandafter\expandafter\expandafter}%
17831 \expandafter
17832 \@gobbletwo
17833 \fi\markdownLaTeXNatbibCitations{#1,#5}}
17834 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
17835 \advance\markdownLaTeXCitationsCounter by 1\relax
17836 \ifx\relax#3\relax
17837 \ifx\relax#4\relax
17838 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17839 \relax
17840 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
17841 \expandafter\expandafter\expandafter
17842 \expandafter\expandafter\expandafter\expandafter
17843 \@gobbletwo
17844 \fi
17845 \else% After a prenote or a postnote, dump the accumulator
17846 \ifx\relax#1\relax\else
17847 \citet{#1}%
17848 \fi
17849 , \citet[#3][#4]{#5}%
17850 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
17851 \relax
17852 ,
17853 \else
17854 \ifnum
17855 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
17856 \relax
17857 ,
17858 \fi
17859 \fi
17860 \expandafter\expandafter\expandafter
17861 \expandafter\expandafter\expandafter\expandafter
17862 \markdownLaTeXNatbibTextCitations
17863 \expandafter\expandafter\expandafter
17864 \expandafter\expandafter\expandafter\expandafter{%
17865 \expandafter\expandafter\expandafter
17866 \expandafter\expandafter\expandafter\expandafter}%
17867 \expandafter\expandafter\expandafter
17868 \@gobbletwo
17869 \fi
17870 \else% After a prenote or a postnote, dump the accumulator
17871 \ifx\relax#1\relax\relax\else
17872 \citet{#1}%
17873 \fi

```

```

17874 , \citet[#3][#4]{#5}%
17875 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
17876 \relax
17877 ,
17878 \else
17879 \ifnum
17880 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
17881 \relax
17882 ,
17883 \fi
17884 \fi
17885 \expandafter\expandafter\expandafter
17886 \markdownLaTeXNatbibTextCitations
17887 \expandafter\expandafter\expandafter{%
17888 \expandafter\expandafter\expandafter}%
17889 \expandafter
17890 \@gobbletwo
17891 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
17892
17893 % BibLaTeX implementation
17894 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
17895 \advance\markdownLaTeXCitationsCounter by 1\relax
17896 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17897 \relax
17898 \autocites#1[#3][#4]{#5}%
17899 \expandafter\@gobbletwo
17900 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
17901 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
17902 \advance\markdownLaTeXCitationsCounter by 1\relax
17903 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
17904 \relax
17905 \textcites#1[#3][#4]{#5}%
17906 \expandafter\@gobbletwo
17907 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
17908
17909 \markdownSetup{rendererPrototypes = {
17910 cite = {%
17911 \markdownLaTeXCitationsCounter=1%
17912 \def\markdownLaTeXCitationsTotal{#1}%
17913 \@ifundefined{autocites}{%
17914 \@ifundefined{citep}{%
17915 \expandafter\expandafter\expandafter
17916 \markdownLaTeXBasicCitations
17917 \expandafter\expandafter\expandafter{%
17918 \expandafter\expandafter\expandafter}%
17919 \expandafter\expandafter\expandafter{%
17920 \expandafter\expandafter\expandafter}%

```

```

17921 }{%
17922 \expandafter\expandafter\expandafter
17923 \markdownLaTeXNatbibCitations
17924 \expandafter\expandafter\expandafter{%
17925 \expandafter\expandafter\expandafter}%
17926 }%
17927 }{%
17928 \expandafter\expandafter\expandafter
17929 \markdownLaTeXBibLaTeXCitations
17930 \expandafter{\expandafter}%
17931 },
17932 textCite = {%
17933 \markdownLaTeXCitationsCounter=1%
17934 \def\markdownLaTeXCitationsTotal{#1}%
17935 \@ifundefined{autocites}{%
17936 \@ifundefined{citep}{%
17937 \expandafter\expandafter\expandafter
17938 \markdownLaTeXBasicTextCitations
17939 \expandafter\expandafter\expandafter{%
17940 \expandafter\expandafter\expandafter}%
17941 \expandafter\expandafter\expandafter{%
17942 \expandafter\expandafter\expandafter}%
17943 }{%
17944 \expandafter\expandafter\expandafter
17945 \markdownLaTeXNatbibTextCitations
17946 \expandafter\expandafter\expandafter{%
17947 \expandafter\expandafter\expandafter}%
17948 }%
17949 }{%
17950 \expandafter\expandafter\expandafter
17951 \markdownLaTeXBibLaTeXTextCitations
17952 \expandafter{\expandafter}%
17953 }}}

```

### 3.3.5.8 Links

Here is an implementation for hypertext links and relative references.

```

17954 \RequirePackage{url}
17955 \RequirePackage{expl3}
17956 \ExplSyntaxOn
17957 \cs_gset_protected:Npn
17958 \markdownRendererLinkPrototype
17959 #1#2#3#4
17960 {
17961 \tl_set:Nn \l_tmpa_tl { #1 }
17962 \tl_set:Nn \l_tmpb_tl { #2 }
17963 \bool_set:Nn

```

```

17964 \l_tmpa_bool
17965 {
17966 \tl_if_eq_p:NN
17967 \l_tmpa_tl
17968 \l_tmpb_tl
17969 }
17970 \tl_set:Nn \l_tmpa_tl { #4 }
17971 \bool_set:Nn
17972 \l_tmpb_bool
17973 {
17974 \tl_if_empty_p:N
17975 \l_tmpa_tl
17976 }

```

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.

```

17977 \bool_if:nTF
17978 {
17979 \l_tmpa_bool && \l_tmpb_bool
17980 }
17981 {
17982 \markdownLaTeXRendererAutolink { #2 } { #3 }
17983 }
17984 {
17985 \markdownLaTeXRendererDirectOrIndirectLink
17986 { #1 } { #2 } { #3 } { #4 }
17987 }
17988 }
17989 \cs_new_protected:Npn
17990 \markdownLaTeXRendererAutolink #1#2
17991 {

```

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.

```

17992 \tl_set:Nn
17993 \l_tmpa_tl
17994 { #2 }
17995 \tl_trim_spaces:N
17996 \l_tmpa_tl
17997 \tl_set:Nx
17998 \l_tmpb_tl
17999 {
18000 \tl_range:Nnn
18001 \l_tmpa_tl
18002 { 1 }
18003 { 1 }
18004 }

```

```

18005 \str_if_eq:NNTF
18006 \l_tmpb_tl
18007 \c_hash_str
18008 {
18009 \tl_set:Nx
18010 \l_tmpb_tl
18011 {
18012 \tl_range:Nnn
18013 \l_tmpa_tl
18014 { 2 }
18015 { -1 }
18016 }
18017 \exp_args:NV
18018 \ref
18019 \l_tmpb_tl
18020 }
18021 {
18022 \url { #2 }
18023 }
18024 }
18025 \ExplSyntaxOff
18026 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
18027 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}%

```

### 3.3.5.9 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

18028 \newcount\markdownLaTeXRowCount
18029 \newcount\markdownLaTeXRowTotal
18030 \newcount\markdownLaTeXColumnCounter
18031 \newcount\markdownLaTeXColumnTotal
18032 \newtoks\markdownLaTeXTable
18033 \newtoks\markdownLaTeXTableAlignment
18034 \newtoks\markdownLaTeXTableEnd
18035 \AtBeginDocument{%
18036 \ifpackageloaded{booktabs}{%
18037 \def\markdownLaTeXTopRule{\toprule}%
18038 \def\markdownLaTeXMidRule{\midrule}%
18039 \def\markdownLaTeXBottomRule{\bottomrule}%
18040 }{%
18041 \def\markdownLaTeXTopRule{\hline}%
18042 \def\markdownLaTeXMidRule{\hline}%
18043 \def\markdownLaTeXBottomRule{\hline}%
18044 }%
18045 }
18046 \markdownSetup{rendererPrototypes={

```



```

18047 table = {%
18048 \markdownLaTeXTable={}%
18049 \markdownLaTeXTableAlignment={}%
18050 \markdownLaTeXTableEnd={%
18051 \markdownLaTeXBottomRule
18052 \end{tabular}}}%
18053 \ifx\empty#1\empty\else
18054 \addto@hook\markdownLaTeXTable{%
18055 \begin{table}
18056 \centering}%
18057 \addto@hook\markdownLaTeXTableEnd{%
18058 \caption{#1}}}%
18059 \fi
18060 }
18061 }}

```

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.

```

18062 \ExplSyntaxOn
18063 \seq_new:N
18064 \l_@@_table_identifiers_seq
18065 \markdownSetup {
18066 rendererPrototypes = {
18067 table += {
18068 \seq_map_inline:Nn
18069 \l_@@_table_identifiers_seq
18070 {
18071 \addto@hook
18072 \markdownLaTeXTableEnd
18073 { \label { ##1 } }
18074 }
18075 },
18076 }
18077 }
18078 \markdownSetup {
18079 rendererPrototypes = {
18080 tableAttributeContextBegin = {
18081 \group_begin:
18082 \markdownSetup {
18083 rendererPrototypes = {
18084 attributeIdentifier = {
18085 \seq_put_right:Nn
18086 \l_@@_table_identifiers_seq
18087 { ##1 }
18088 },
18089 },
18090 }
18091 }
18092 }
18093 }

```

```

18091 },
18092 tableAttributeContextEnd = {
18093 \group_end:
18094 },
18095 },
18096 }
18097 \ExplSyntaxOff
18098 \markdownSetup{rendererPrototypes={
18099 table += {%
18100 \ifx\empty#1\empty\else
18101 \addto@hook\markdownLaTeXTableEnd{%
18102 \end{table}}}%
18103 \fi
18104 \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
18105 \markdownLaTeXRowCounter=0%
18106 \markdownLaTeXRowTotal=#2%
18107 \markdownLaTeXColumnTotal=#3%
18108 \markdownLaTeXRenderTableRow
18109 }
18110 }}
18111 \def\markdownLaTeXRenderTableRow#1{%
18112 \markdownLaTeXColumnCounter=0%
18113 \ifnum\markdownLaTeXRowCounter=0\relax
18114 \markdownLaTeXReadAlignments#1%
18115 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
18116 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
18117 \the\markdownLaTeXTableAlignment}}}%
18118 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
18119 \else
18120 \markdownLaTeXRenderTableCell#1%
18121 \fi
18122 \ifnum\markdownLaTeXRowCounter=1\relax
18123 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
18124 \fi
18125 \advance\markdownLaTeXRowCounter by 1\relax
18126 \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
18127 \the\markdownLaTeXTable
18128 \the\markdownLaTeXTableEnd
18129 \expandafter\@gobble
18130 \fi\markdownLaTeXRenderTableRow}
18131 \def\markdownLaTeXReadAlignments#1{%
18132 \advance\markdownLaTeXColumnCounter by 1\relax
18133 \if#1d%
18134 \addto@hook\markdownLaTeXTableAlignment{1}%
18135 \else
18136 \addto@hook\markdownLaTeXTableAlignment{#1}%
18137 \fi

```

```

18138 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
18139 \expandafter\@gobble
18140 \fi\markdownLaTeXReadAlignments}
18141 \def\markdownLaTeXRenderTableCell#1{%
18142 \advance\markdownLaTeXColumnCounter by 1\relax
18143 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
18144 \addto@hook\markdownLaTeXTable{#1&}%
18145 \else
18146 \addto@hook\markdownLaTeXTable{#1\\}%
18147 \expandafter\@gobble
18148 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.5.10 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

18149
18150 \markdownIfOption{lineBlocks}{%
18151 \RequirePackage{verse}
18152 \markdownSetup{rendererPrototypes={
18153 lineBlockBegin = {%
18154 \begingroup
18155 \def\markdownRendererHardLineBreak{\\}%
18156 \begin{verse}%
18157 },
18158 lineBlockEnd = {%
18159 \end{verse}%
18160 \endgroup
18161 },
18162 }}
18163 }{}
18164

```

### 3.3.5.11 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.

```

18165 \ExplSyntaxOn
18166 \keys_define:nn
18167 { markdown / jekyllData }
18168 {
18169 author .value_required:n = { true },
18170 author .code:n = {
18171 \author
18172 { #1 }
18173 },

```

```

18174 date .value_required:n = { true },
18175 date .code:n = {
18176 \date
18177 { #1 }
18178 },
18179 title .value_required:n = { true },
18180 title .code:n = {
18181 \title
18182 { #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.

```

18183 \char_set_catcode_comment:N \%
18184 \char_set_catcode_parameter:N \#
18185 \AddToHook
18186 { begindocument / end }
18187 { \maketitle }
18188 \char_set_catcode_other:N \%
18189 \char_set_catcode_other:N \#
18190 },
18191 }

```

### 3.3.5.12 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.

```

18192 \@@_if_option:nT
18193 { mark }
18194 {
18195 \sys_if_engine luatex:TF
18196 {
18197 \RequirePackage
18198 { luacolor }
18199 \RequirePackage
18200 { lua-ul }
18201 \markdownSetup
18202 {
18203 rendererPrototypes = {
18204 mark = {
18205 \highLight
18206 { #1 }
18207 },
18208 }

```

```

18209 }
18210 }
18211 {
18212 \RequirePackage
18213 { xcolor }
18214 \RequirePackage
18215 { soul }
18216 \markdownSetup
18217 {
18218 rendererPrototypes = {
18219 mark = {
18220 \hl
18221 { #1 }
18222 },
18223 }
18224 }
18225 }
18226 }

```

### 3.3.5.13 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.

```

18227 \@@_if_option:nT
18228 { strikeThrough }
18229 {
18230 \sys_if_engine luatex:TF
18231 {
18232 \RequirePackage
18233 { lua-ul }
18234 \markdownSetup
18235 {
18236 rendererPrototypes = {
18237 strikeThrough = {
18238 \strikeThrough
18239 { #1 }
18240 },
18241 }
18242 }
18243 }
18244 {
18245 \RequirePackage
18246 { soul }
18247 \markdownSetup
18248 {
18249 rendererPrototypes = {
18250 strikeThrough = {

```

```

18251 \st
18252 { #1 }
18253 },
18254 }
18255 }
18256 }
18257 }

```

### 3.3.5.14 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.

```

18258 \seq_new:N
18259 \l_@@_image_identifiers_seq
18260 \markdownSetup {
18261 rendererPrototypes = {
18262 image = {
18263 \tl_if_empty:nTF
18264 { #4 }
18265 {
18266 \begin { center }
18267 \includegraphics
18268 [alt = { #1 }]
18269 { #3 }
18270 \end { center }
18271 }
18272 {
18273 \begin { figure }
18274 \begin { center }
18275 \includegraphics
18276 [alt = { #1 }]
18277 { #3 }
18278 \caption { #4 }
18279 \seq_map_inline:Nn
18280 \l_@@_image_identifiers_seq
18281 { \label { ##1 } }
18282 \end { center }
18283 \end { figure }
18284 }
18285 },
18286 }
18287 }

```

```

18288 \@@_if_option:nT
18289 { linkAttributes }
18290 {
18291 \RequirePackage { graphicx }
18292 }
18293 \markdownSetup {
18294 rendererPrototypes = {
18295 imageAttributeContextBegin = {
18296 \group_begin:
18297 \markdownSetup {
18298 rendererPrototypes = {
18299 attributeIdentifier = {
18300 \seq_put_right:Nn
18301 \l_@@_image_identifiers_seq
18302 { ##1 }
18303 },
18304 attributeKeyValue = {
18305 \setkeys
18306 { Gin }
18307 { { ##1 } = { ##2 } }
18308 },
18309 },
18310 }
18311 },
18312 imageAttributeContextEnd = {
18313 \group_end:
18314 },
18315 },
18316 }
18317 \ExplSyntaxOff

```

### 3.3.5.15 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`.

```

18318 \ExplSyntaxOn
18319 \cs_new:Nn
18320 \@@_luaxml_print_html:n
18321 {
18322 \luabridge_now:n
18323 {
18324 local~input_file = assert(io.open(" #1 ", "r"))
18325 local~input = assert(input_file:read("*a"))
18326 assert(input_file:close())
18327 input = "<body>" .. input .. "</body>"

```

```

18328 local~dom = require("luaxml-domobject").html_parse(input)
18329 local~output = require("luaxml-htmltemplates"):process_dom(dom)
18330 print(output)
18331 }
18332 }
18333 \cs_gset_protected:Npn
18334 \markdownRendererInputRawInlinePrototype#1#2
18335 {
18336 \str_case:nnF
18337 { #2 }
18338 {
18339 { latex }
18340 {
18341 \@@_plain_tex_default_input_raw_inline:nn
18342 { #1 }
18343 { tex }
18344 }
18345 { html }
18346 {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>44</sup>, we will pass HTML elements to the output HTML document unchanged.

```

18347 \cs_if_exist:NTF
18348 \HCode
18349 {
18350 \if_mode_vertical:
18351 \IgnorePar
18352 \EndP
18353 \fi:
18354 \special
18355 { t4ht* < #1 }
18356 }
18357 {
18358 \@@_luaxml_print_html:n
18359 { #1 }
18360 }
18361 }
18362 }
18363 {
18364 \@@_plain_tex_default_input_raw_inline:nn
18365 { #1 }
18366 { #2 }
18367 }
18368 }
18369 \cs_gset_protected:Npn
18370 \markdownRendererInputRawBlockPrototype#1#2

```

---

<sup>44</sup>See <https://tug.org/tex4ht/>.



```

18371 {
18372 \str_case:nnF
18373 { #2 }
18374 {
18375 { latex }
18376 {
18377 \@@_plain_tex_default_input_raw_block:nn
18378 { #1 }
18379 { tex }
18380 }
18381 { html }
18382 {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>45</sup>, we will pass HTML elements to the output HTML document unchanged.

```

18383 \cs_if_exist:NTF
18384 \HCode
18385 {
18386 \if_mode_vertical:
18387 \IgnorePar
18388 \fi:
18389 \EndP
18390 \special
18391 { t4ht* < #1 }
18392 \par
18393 \ShowPar
18394 }
18395 {
18396 \@@_luaxml_print_html:n
18397 { #1 }
18398 }
18399 }
18400 }
18401 {
18402 \@@_plain_tex_default_input_raw_block:nn
18403 { #1 }
18404 { #2 }
18405 }
18406 }

```

### 3.3.5.16 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing the last  $\text{\LaTeX}$  counter that has been incremented in e.g. ordered lists.

```

18407 \seq_new:N

```

---

<sup>45</sup>See <https://tug.org/tex4ht/>.

```

18408 \l_@@_bracketed_span_identifiers_seq
18409 \markdownSetup {
18410 rendererPrototypes = {
18411 bracketedSpanAttributeContextBegin $= {
18412 \markdownSetup {
18413 rendererPrototypes = {
18414 attributeIdentifier = {
18415 \seq_put_right:Nn
18416 \l_@@_bracketed_span_identifiers_seq
18417 { ##1 }
18418 },
18419 },
18420 }
18421 },
18422 bracketedSpanAttributeContextEnd ^= {
18423 \seq_map_inline:Nn
18424 \l_@@_bracketed_span_identifiers_seq
18425 { \label { ##1 } }
18426 },
18427 },
18428 }

```

We will also allow acronyms to be manually marked up using the HTML class [acronym](#).

```

18429 \markdownSetup {
18430 rendererPrototypes = {
18431 bracketedSpanAttributeContextBegin += {
18432 \markdownSetup {
18433 rendererPrototypes = {
18434 attributeClassName = {
18435 \str_if_eq:nnT
18436 { ##1 }
18437 { acronym }
18438 {
18439 \markdownSetup {
18440 rendererPrototypes = {
18441 bracketedSpan = {
18442 \bool_set_true:N
18443 \g__markdown_record_acronyms_bool
18444 \markdownRendererAcronym
18445 { #####1 }
18446 }
18447 }
18448 }
18449 },
18450 },
18451 },
18452 }
18453 },

```

```

18454 },
18455 }
18456 \ExplSyntaxOff
18457 \fi % Closes ~\markdownIfOption{plain}{\iffalse}{\iftrue}~

```

### 3.3.6 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.

```

18458 \newcommand\markdownMakeOther{%
18459 \count0=128\relax
18460 \loop
18461 \catcode\count0=11\relax
18462 \advance\count0 by 1\relax
18463 \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.

```

18464 \def\markdownMakeOther{%
18465 \count0=128\relax
18466 \loop
18467 \catcode\count0=11\relax
18468 \advance\count0 by 1\relax
18469 \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.

```

18470 \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).

```

18471 \long\def\inputmarkdown{%

```

```

18472 \dosingleempty
18473 \doinputmarkdown}%
18474 \long\def\doinputmarkdown[#1]#2{%
18475 \begingroup
18476 \iffirstargument
18477 \setupmarkdown[#1]%
18478 \fi
18479 \markdownInput{#2}%
18480 \endgroup}%
18481 \long\def\inputyaml{%
18482 \dosingleempty
18483 \doinputyaml}%
18484 \long\def\doinputyaml[#1]#2{%
18485 \doinputmarkdown
18486 [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s  $\text{\TeX}$ , trailing spaces are removed very early on when a line is being put to the input buffer. [21, sec. 31]. According to Eijkhout [22, 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) $\text{\TeX}$ , but Con $\text{\TeX}$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\text{\TeX}$ t MkIV and therefore to insert hard line breaks into markdown text.

```

18487 \startluacode
18488 document.markdown_buffering = false
18489 local function preserve_trailing_spaces(line)
18490 if document.markdown_buffering then
18491 line = line:gsub("[\t][\t]$", "\t\t")
18492 end
18493 return line
18494 end
18495 resolvers.installinputlinehandler(preserve_trailing_spaces)
18496 \stopluacode
18497 \begingroup
18498 \catcode`\|=0%
18499 \catcode`\|=12%
18500 |gdef|startmarkdown{%
18501 |ctxlua{document.markdown_buffering = true}%
18502 |markdownReadAndConvert{\stopmarkdown}%
18503 {|stopmarkdown}}%
18504 |gdef|stopmarkdown{%
18505 |ctxlua{document.markdown_buffering = false}%
18506 |markdownEnd}%
18507 |gdef|startyaml{%
18508 |begingroup
18509 |ctxlua{document.markdown_buffering = true}%

```

```

18510 |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
18511 |markdownReadAndConvert{\stopyaml}%
18512 {\stopyaml}}%
18513 |gdef|stopyaml{%
18514 |ctxlua{document.markdown_buffering = false}%
18515 |yamlEnd}%
18516 |endgroup

```

### 3.4.2 Themes

This section overrides the plain T<sub>E</sub>X implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConT<sub>E</sub>Xt themes provided with the Markdown package.

```

18517 \ExplSyntaxOn
18518 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
18519 \prop_new:N \g_@@_context_loaded_themes_versions_prop
18520 \cs_gset_protected:Nn
18521 \@@_load_theme:nnn
18522 {

```

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 T<sub>E</sub>X theme instead.

```

18523 \bool_if:nTF
18524 {
18525 \bool_lazy_or_p:nn
18526 {
18527 \prop_if_in_p:Nn
18528 \g_@@_context_built_in_themes_prop
18529 { #1 }
18530 }
18531 {
18532 \file_if_exist_p:n
18533 { t - markdown theme #3.tex }
18534 }
18535 }
18536 {
18537 \prop_get:NnNTF
18538 \g_@@_context_loaded_themes_linenos_prop
18539 { #1 }
18540 \l_tmpa_tl
18541 {
18542 \prop_get:NnN
18543 \g_@@_context_loaded_themes_versions_prop
18544 { #1 }

```

```

18545 \l_tmpb_tl
18546 \str_if_eq:nVTF
18547 { #2 }
18548 \l_tmpb_tl
18549 {
18550 \msg_warning:nnnVn
18551 { markdown }
18552 { repeatedly-loaded-context-theme }
18553 { #1 }
18554 \l_tmpa_tl
18555 { #2 }
18556 }
18557 {
18558 \msg_error:nnnnVV
18559 { markdown }
18560 { different-versions-of-context-theme }
18561 { #1 }
18562 { #2 }
18563 \l_tmpb_tl
18564 \l_tmpa_tl
18565 }
18566 }
18567 {
18568 \prop_gput:Nnx
18569 \g_@@_context_loaded_themes_linenos_prop
18570 { #1 }
18571 { \tex_the:D \tex_inputlineno:D } % noqa: W200
18572 \prop_gput:Nnn
18573 \g_@@_context_loaded_themes_versions_prop
18574 { #1 }
18575 { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

18576 \prop_if_in:NnTF
18577 \g_@@_context_built_in_themes_prop
18578 { #1 }
18579 {
18580 \msg_info:nnnn
18581 { markdown }
18582 { loading-built-in-context-theme }
18583 { #1 }
18584 { #2 }
18585 \prop_item:Nn
18586 \g_@@_context_built_in_themes_prop
18587 { #1 }
18588 }

```

```

18589 {
18590 \msg_info:nnnn
18591 { markdown }
18592 { loading-context-theme }
18593 { #1 }
18594 { #2 }
18595 \usemodule
18596 [t]
18597 [markdown theme #3]
18598 }
18599 }
18600 }
18601 {
18602 \@@_plain_tex_load_theme:nnn
18603 { #1 }
18604 { #2 }
18605 { #3 }
18606 }
18607 }
18608 \msg_new:nnn
18609 { markdown }
18610 { loading-built-in-context-theme }
18611 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
18612 \msg_new:nnn
18613 { markdown }
18614 { loading-context-theme }
18615 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
18616 \msg_new:nnn
18617 { markdown }
18618 { repeatedly-loaded-context-theme }
18619 {
18620 Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
18621 loaded~on~line~#2,~not~loading~it~again
18622 }
18623 \msg_new:nnn
18624 { markdown }
18625 { different-versions-of-context-theme }
18626 {
18627 Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
18628 but~version~#3~has~already~been~loaded~on~line~#4
18629 }
18630 \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:

```

18631 \markdownLoadPlainTeXTheme

```

Next, the ConT<sub>E</sub>Xt 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.

```

18632 \markdownIfOption{plain}{\iffalse}{\iftrue}
18633 \def\markdownRendererHardLineBreakPrototype{\blank}%
18634 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
18635 \def\markdownRendererRightBracePrototype{\textbraceright}%
18636 \def\markdownRendererDollarSignPrototype{\textdollar}%
18637 \def\markdownRendererPercentSignPrototype{\percent}%
18638 \def\markdownRendererUnderscorePrototype{\textunderscore}%
18639 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
18640 \def\markdownRendererBackslashPrototype{\textbackslash}%
18641 \def\markdownRendererTildePrototype{\textasciitilde}%
18642 \def\markdownRendererPipePrototype{\char`|}%
18643 \def\markdownRendererLinkPrototype#1#2#3#4{%
18644 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
18645 \fi\texttt<\hyphenatedurl{#3}>}}%
18646 \usemodule[database]
18647 \defineseparatedlist
18648 [MarkdownConTeXtCSV]
18649 [separator={,},
18650 before=\bTABLE,after=\eTABLE,
18651 first=\bTR,last=\eTR,
18652 left=\bTD,right=\eTD]
18653 \def\markdownConTeXtCSV{csv}
18654 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
18655 \def\markdownConTeXtCSV@arg{#1}%
18656 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
18657 \placetable[] [tab:#1]{#4}{%
18658 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
18659 \else
18660 \markdownInput{#3}%
18661 \fi}%
18662 \def\markdownRendererImagePrototype#1#2#3#4{%
18663 \placefigure[] []{#4}{\externalfigure[#3]}}%
18664 \def\markdownRendererUlBeginPrototype{\startitemize}%
18665 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
18666 \def\markdownRendererUlItemPrototype{\item}%
18667 \def\markdownRendererUlEndPrototype{\stopitemize}%
18668 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
18669 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
18670 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%

```



```

18671 \def\markdownRendererOlItemPrototype{\item}%
18672 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
18673 \def\markdownRendererOlEndPrototype{\stopitemize}%
18674 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
18675 \definedescription
18676 [MarkdownConTeXtDlItemPrototype]
18677 [location=hanging,
18678 margin=standard,
18679 headstyle=bold]%
18680 \definestartstop
18681 [MarkdownConTeXtDlPrototype]
18682 [before=\blank,
18683 after=\blank]%
18684 \definestartstop
18685 [MarkdownConTeXtDlTightPrototype]
18686 [before=\blank\startpacked,
18687 after=\stoppacked\blank]%
18688 \def\markdownRendererDlBeginPrototype{%
18689 \startMarkdownConTeXtDlPrototype}%
18690 \def\markdownRendererDlBeginTightPrototype{%
18691 \startMarkdownConTeXtDlTightPrototype}%
18692 \def\markdownRendererDlItemPrototype#1{%
18693 \startMarkdownConTeXtDlItemPrototype{#1}}%
18694 \def\markdownRendererDlItemEndPrototype{%
18695 \stopMarkdownConTeXtDlItemPrototype}%
18696 \def\markdownRendererDlEndPrototype{%
18697 \stopMarkdownConTeXtDlPrototype}%
18698 \def\markdownRendererDlEndTightPrototype{%
18699 \stopMarkdownConTeXtDlTightPrototype}%
18700 \def\markdownRendererEmphasisPrototype#1{\em#1}%
18701 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
18702 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
18703 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
18704 \def\markdownRendererLineBlockBeginPrototype{%
18705 \begingroup
18706 \def\markdownRendererHardLineBreak{
18707 }%
18708 \startlines
18709 }%
18710 \def\markdownRendererLineBlockEndPrototype{%
18711 \stoplines
18712 \endgroup
18713 }%
18714 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

18715 \ExplSyntaxOn
18716 \cs_gset:Npn
18717 \markdownRendererInputFencedCodePrototype#1#2#3
18718 {
18719 \tl_if_empty:nTF
18720 { #2 }
18721 { \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

```

```

18722 {
18723 \regex_extract_once:nnN
18724 { \w* }
18725 { #2 }
18726 \l_tmpa_seq
18727 \seq_pop_left:NN
18728 \l_tmpa_seq
18729 \l_tmpa_tl
18730 \typefile[\l_tmpa_tl][] {#1}
18731 }
18732 }
18733 \ExplSyntaxOff
18734 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
18735 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
18736 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
18737 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
18738 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%

```

```

18739 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
18740 \def\markdownRendererThematicBreakPrototype{%
18741 \blackrule[height=1pt, width=\hsize]}%
18742 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
18743 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}
18744 \def\markdownRendererHalfTickedBoxPrototype{${\boxdot}$}
18745 \def\markdownRendererUntickedBoxPrototype{${\square}$}
18746 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
18747 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
18748 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
18749 \def\markdownRendererDisplayMathPrototype#1{%
18750 \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

18751 \newcount\markdownConTeXtRowCounter
18752 \newcount\markdownConTeXtRowTotal
18753 \newcount\markdownConTeXtColumnCounter
18754 \newcount\markdownConTeXtColumnTotal
18755 \newtoks\markdownConTeXtTable
18756 \newtoks\markdownConTeXtTableFloat
18757 \def\markdownRendererTablePrototype#1#2#3{%
18758 \markdownConTeXtTable={}%
18759 \ifx\empty#1\empty
18760 \markdownConTeXtTableFloat={%
18761 \the\markdownConTeXtTable}%
18762 \else
18763 \markdownConTeXtTableFloat={%
18764 \placetable{#1}{\the\markdownConTeXtTable}}%
18765 \fi
18766 \begingroup
18767 \setupTABLE[r][each][topframe=off, bottomframe=off,
18768 leftframe=off, rightframe=off]
18769 \setupTABLE[c][each][topframe=off, bottomframe=off,
18770 leftframe=off, rightframe=off]
18771 \setupTABLE[r][1][topframe=on, bottomframe=on]
18772 \setupTABLE[r][#1][bottomframe=on]
18773 \markdownConTeXtRowCounter=0%
18774 \markdownConTeXtRowTotal=#2%
18775 \markdownConTeXtColumnTotal=#3%
18776 \markdownConTeXtRenderTableRow}
18777 \def\markdownConTeXtRenderTableRow#1{%
18778 \markdownConTeXtColumnCounter=0%
18779 \ifnum\markdownConTeXtRowCounter=0\relax
18780 \markdownConTeXtReadAlignments#1%
18781 \markdownConTeXtTable={\bTABLE}%

```

```

18782 \else
18783 \markdownConTeXtTable=\expandafter{%
18784 \the\markdownConTeXtTable\bTR}%
18785 \markdownConTeXtRenderTableCell#1%
18786 \markdownConTeXtTable=\expandafter{%
18787 \the\markdownConTeXtTable\eTR}%
18788 \fi
18789 \advance\markdownConTeXtRowCounter by 1\relax
18790 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
18791 \markdownConTeXtTable=\expandafter{%
18792 \the\markdownConTeXtTable\eTABLE}%
18793 \the\markdownConTeXtTableFloat
18794 \endgroup
18795 \expandafter\gobbleoneargument
18796 \fi\markdownConTeXtRenderTableRow}
18797 \def\markdownConTeXtReadAlignments#1{%
18798 \advance\markdownConTeXtColumnCounter by 1\relax
18799 \if#1d%
18800 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
18801 \fi\if#1l%
18802 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
18803 \fi\if#1c%
18804 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
18805 \fi\if#1r%
18806 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
18807 \fi
18808 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
18809 \else
18810 \expandafter\gobbleoneargument
18811 \fi\markdownConTeXtReadAlignments}
18812 \def\markdownConTeXtRenderTableCell#1{%
18813 \advance\markdownConTeXtColumnCounter by 1\relax
18814 \markdownConTeXtTable=\expandafter{%
18815 \the\markdownConTeXtTable\bTD#1\eTD}%
18816 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
18817 \else
18818 \expandafter\gobbleoneargument
18819 \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`.

```

18820 \ExplSyntaxOn
18821 \cs_gset:Npn
18822 \markdownRendererInputRawInlinePrototype#1#2
18823 {

```

```

18824 \str_case:nnF
18825 { #2 }
18826 {
18827 { latex }
18828 {
18829 \@@_plain_tex_default_input_raw_inline:nn
18830 { #1 }
18831 { context }
18832 }
18833 }
18834 {
18835 \@@_plain_tex_default_input_raw_inline:nn
18836 { #1 }
18837 { #2 }
18838 }
18839 }
18840 \cs_gset:Npn
18841 \markdownRendererInputRawBlockPrototype#1#2
18842 {
18843 \str_case:nnF
18844 { #2 }
18845 {
18846 { context }
18847 {
18848 \@@_plain_tex_default_input_raw_block:nn
18849 { #1 }
18850 { tex }
18851 }
18852 }
18853 {
18854 \@@_plain_tex_default_input_raw_block:nn
18855 { #1 }
18856 { #2 }
18857 }
18858 }
18859 \cs_gset_eq:NN
18860 \markdownRendererInputRawBlockPrototype
18861 \markdownRendererInputRawInlinePrototype
18862 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}
18863 \ExplSyntaxOff
18864 \stopmodule
18865 \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).

```

18866 \ExplSyntaxOn

```

```

18867 \str_if_eq:VVT
18868 \c_@@_top_layer_tl
18869 \c_@@_option_layer_context_tl
18870 {
18871 \use:c
18872 { ExplSyntaxOff }
18873 \@@_if_option:nF
18874 { noDefaults }
18875 {
18876 \@@_if_option:nTF
18877 { experimental }
18878 {
18879 \@@_setup:n
18880 { theme = witiko/markdown/defaults@experimental }
18881 }
18882 {
18883 \@@_setup:n
18884 { theme = witiko/markdown/defaults }
18885 }
18886 }
18887 \use:c
18888 { ExplSyntaxOn }
18889 }
18890 \ExplSyntaxOff
18891 \stopmodule
18892 \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 shortvrb 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<sub>2<sub>ε</sub></sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).

- [20] L<sup>A</sup>T<sub>E</sub>X Project. *The L<sup>A</sup>T<sub>E</sub>X3 Interfaces*. Jan. 19, 2026. URL: <https://mirrors.ctan.org/macros/latex/required/l3kernel/interface3.pdf> (visited on 02/18/2026).
- [21] 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.
- [22] 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

acronyms	23, 94, 177, 485, 487, 488
autoIdentifiers	23, 36, 94, 116
blankBeforeBlockquote	24
blankBeforeCodeFence	24
blankBeforeDivFence	24
blankBeforeHeading	25
blankBeforeHtmlBlock	25
blankBeforeList	25
bracketedSpans	26, 98, 521
breakableBlockquotes	26
cacheDir	4, 18, 21, 63, 64, 175, 206, 422, 445, 466
citationNbsps	26
citations	27, 101, 102
codeSpans	27
contentBlocks	22, 28, 38
contentBlocksLanguageMap	22
contentLevel	28
debugExtensions	9, 22, 29, 363
debugExtensionsFileName	22, 29
defaultOptions	10, 55, 420, 422
definitionLists	29, 110
depth_first_search	183
eagerCache	18, 420
ensureJekyllData	30, 30, 31, 41
entities.char_entity	261
entities.dec_entity	260
entities.hex_entity	260
entities.hex_entity_with_x_char	260



escape_minimal	265
escape_programmatic_text	265
escape_typographic_text	265
expandtabs	323
expectJekyllData	30, 30, 31, 41
experimental	5, 19, 38, 491
experimentalOptions	10, 420, 422
extensions	32, 182, 369
extensions.acronyms	184, 369
extensions.citations	372
extensions.content_blocks	376
extensions.definition_lists	379
extensions.fancy_lists	381
extensions.fenced_code	387
extensions.fenced_divs	392
extensions.header_attributes	397
extensions.inline_code_attributes	398
extensions.jekyll_data	416
extensions.line_blocks	399
extensions.link_attributes	400
extensions.mark	400
extensions.notes	402
extensions.pipe_table	404
extensions.raw_inline	409
extensions.strike_through	410
extensions.subscripts	410
extensions.superscripts	411
extensions.tex_math	412
fancyLists	33, 126–131, 491
fencedCode	33, 44, 102, 115, 132, 434, 437
fencedCodeAttributes	34, 94, 115, 437
fencedDivs	34, 45, 115
finalizeCache	18, 19, 23, 35, 35, 63, 64, 174, 420, 421
frozenCache	23, 35, 64, 83, 86, 174, 434, 445
frozenCacheCounter	35, 421, 474, 475
frozenCacheFileName	23, 35, 63, 421
\g_markdown_diagrams_infostrings_prop	439
gfmAutoIdentifiers	24, 36, 94, 116
hashEnumerators	36
headerAttributes	36, 45, 94, 116

html	37, 37, 38, 96, 104, 106, 505
htmlOutput	37, 96, 104, 106
htmlOverLinks	19, 38
hybrid	38, 38, 44, 50, 52, 67, 86, 133, 175, 265, 324, 474
inlineCodeAttributes	40, 94, 103
inlineNotes	40
\inputmarkdown	178, 179, 180, 523
inputTempFileName	65, 67, 467, 468, 471, 472
\inputyaml	30, 31, 41, 178, 180, 523
iterlines	323
jekyllData	3, 30, 31, 40, 41, 142–145, 148
languages_json	376, 376
lineBlocks	42, 121
linkAttributes	41, 94, 119, 123, 343, 518
mark	42, 124, 516
\markdown	170, 170, 171, 478
markdown	169, 169, 170, 477
markdown*	169, 169, 174, 477
\markdownBegin	57, 58–60, 167, 169, 170, 178
\markdownCleanup	466
\markdownConvert	466
\markdownEnd	57, 58–60, 167, 169–171, 178
\markdownError	167, 167
\markdownEscape	57, 61, 475
\markdownIfOption	62
\markdownIfSnippetExists	88
\markdownInfo	167, 167
\markdownInput	57, 60, 169, 171, 174, 179, 473, 476
\markdownInputFilename	465
\markdownInputFileStream	467
\markdownInputPlainTeX	476
\markdownLoadPlainTeXTheme	175, 182, 433
\markdownLuaExecute	469, 473
\markdownLuaOptions	462, 466
\markdownMakeOther	167, 523
\markdownOptionExperimental	67
\markdownOptionFinalizeCache	63
\markdownOptionFrozenCache	63
\markdownOptionInputTempFileName	64

\markdownOptionNoDefaults	66
\markdownOptionOutputDir	64, 65, 68, 69
\markdownOptionPlain	65
\markdownOptionStripPercentSigns	66
\markdownOutputFileStream	467
\markdownPrepare	466
\markdownPrepareInputFilename	465
\markdownPrepareLuaOptions	462
\markdownReadAndConvert	167, 467, 477, 478, 524
\markdownReadAndConvertProcessLine	468, 469
\markdownReadAndConvertStripPercentSigns	468
\markdownReadAndConvertTab	467
\markdownRendererAcronym	94
\markdownRendererAttributeClassName	95
\markdownRendererAttributeIdentifier	95
\markdownRendererAttributeKeyValue	95
\markdownRendererBlockHtmlComment	104
\markdownRendererBlockHtmlStandaloneTag	104
\markdownRendererBlockQuoteBegin	97
\markdownRendererBlockQuoteEnd	97
\markdownRendererBracketedSpan	98
\markdownRendererBracketedSpanAttributeContextBegin	98
\markdownRendererBracketedSpanAttributeContextEnd	98
\markdownRendererCite	101, 102
\markdownRendererCodeSpan	103
\markdownRendererCodeSpanAttributeContextBegin	103
\markdownRendererCodeSpanAttributeContextEnd	103
\markdownRendererContentBlock	108, 109
\markdownRendererContentBlockCode	109
\markdownRendererContentBlockOnlineImage	109
\markdownRendererDisplayMath	139
\markdownRendererDlBegin	110
\markdownRendererDlBeginTight	111
\markdownRendererDlDefinitionBegin	112
\markdownRendererDlDefinitionEnd	112
\markdownRendererDlEnd	112
\markdownRendererDlEndTight	113
\markdownRendererDlItem	111
\markdownRendererDlItemEnd	111
\markdownRendererDocumentBegin	124
\markdownRendererDocumentEnd	124
\markdownRendererEllipsis	46, 113

<code>\markdownRendererEmphasis</code>	114, 152
<code>\markdownRendererError</code>	141
<code>\markdownRendererFancyOlBegin</code>	126, 127
<code>\markdownRendererFancyOlBeginTight</code>	127
<code>\markdownRendererFancyOlEnd</code>	131
<code>\markdownRendererFancyOlEndTight</code>	131
<code>\markdownRendererFancyOlItem</code>	129
<code>\markdownRendererFancyOlItemEnd</code>	129
<code>\markdownRendererFancyOlItemWithNumber</code>	129
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	115
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	115
<code>\markdownRendererFencedDivAttributeContextBegin</code>	115
<code>\markdownRendererFencedDivAttributeContextEnd</code>	115
<code>\markdownRendererHalfTickedBox</code>	140
<code>\markdownRendererHardLineBreak</code>	122
<code>\markdownRendererHeaderAttributeContextBegin</code>	116
<code>\markdownRendererHeaderAttributeContextEnd</code>	116
<code>\markdownRendererHeadingFive</code>	118
<code>\markdownRendererHeadingFour</code>	118
<code>\markdownRendererHeadingOne</code>	117
<code>\markdownRendererHeadingSix</code>	119
<code>\markdownRendererHeadingThree</code>	117
<code>\markdownRendererHeadingTwo</code>	117
<code>\markdownRendererImage</code>	119
<code>\markdownRendererImageAttributeContextBegin</code>	119
<code>\markdownRendererImageAttributeContextEnd</code>	119
<code>\markdownRendererInlineHtmlCdataSection</code>	107
<code>\markdownRendererInlineHtmlCloseTag</code>	107
<code>\markdownRendererInlineHtmlComment</code>	96
<code>\markdownRendererInlineHtmlDeclaration</code>	107
<code>\markdownRendererInlineHtmlEmptyTag</code>	107
<code>\markdownRendererInlineHtmlOpenTag</code>	107
<code>\markdownRendererInlineHtmlProcessingInstruction</code>	107
<code>\markdownRendererInlineHtmlTag</code>	96, 107
<code>\markdownRendererInlineMath</code>	139
<code>\markdownRendererInputBlockHtmlCdataElement</code>	104
<code>\markdownRendererInputBlockHtmlCdataSection</code>	104
<code>\markdownRendererInputBlockHtmlDeclaration</code>	104
<code>\markdownRendererInputBlockHtmlElement</code>	96, 104
<code>\markdownRendererInputBlockHtmlPcdataElement</code>	104
<code>\markdownRendererInputBlockHtmlProcessingInstruction</code>	104
<code>\markdownRendererInputFencedCode</code>	102

<code>\markdownRendererInputRawBlock</code>	132
<code>\markdownRendererInputRawInline</code>	131
<code>\markdownRendererInputVerbatim</code>	102
<code>\markdownRendererInterblockSeparator</code>	120
<code>\markdownRendererJekyllDataBegin</code>	142
<code>\markdownRendererJekyllDataBoolean</code>	144
<code>\markdownRendererJekyllDataEmpty</code>	148
<code>\markdownRendererJekyllDataEnd</code>	142
<code>\markdownRendererJekyllDataMappingBegin</code>	143
<code>\markdownRendererJekyllDataMappingEnd</code>	143
<code>\markdownRendererJekyllDataNumber</code>	145
<code>\markdownRendererJekyllDataProgrammaticString</code>	145, 145, 146
<code>\markdownRendererJekyllDataSequenceBegin</code>	144
<code>\markdownRendererJekyllDataSequenceEnd</code>	144
<code>\markdownRendererJekyllDataString</code>	146, 146, 150
<code>\markdownRendererJekyllDataStringPrototype</code>	162
<code>\markdownRendererJekyllDataTypographicString</code>	145, 145, 146, 416
<code>\markdownRendererLineBlockBegin</code>	121
<code>\markdownRendererLineBlockEnd</code>	121
<code>\markdownRendererLink</code>	122, 152
<code>\markdownRendererLinkAttributeContextBegin</code>	123
<code>\markdownRendererLinkAttributeContextEnd</code>	123
<code>\markdownRendererMark</code>	124
<code>\markdownRendererNbsp</code>	125
<code>\markdownRendererNote</code>	125
<code>\markdownRendererOlBegin</code>	126
<code>\markdownRendererOlBeginTight</code>	126
<code>\markdownRendererOlEnd</code>	130
<code>\markdownRendererOlEndTight</code>	130
<code>\markdownRendererOlItem</code>	46, 127
<code>\markdownRendererOlItemEnd</code>	128
<code>\markdownRendererOlItemWithNumber</code>	46, 128
<code>\markdownRendererParagraphSeparator</code>	121
<code>\markdownRendererReplacementCharacter</code>	133
<code>\markdownRendererSectionBegin</code>	132
<code>\markdownRendererSectionEnd</code>	132
<code>\markdownRendererSoftLineBreak</code>	122
<code>\markdownRendererStrikeThrough</code>	136
<code>\markdownRendererStrongEmphasis</code>	114
<code>\markdownRendererSubscript</code>	137
<code>\markdownRendererSuperscript</code>	137
<code>\markdownRendererTable</code>	138

<code>\markdownRendererTableAttributeContextBegin</code>	138
<code>\markdownRendererTableAttributeContextEnd</code>	138
<code>\markdownRendererTextCite</code>	102
<code>\markdownRendererThematicBreak</code>	140
<code>\markdownRendererTickedBox</code>	140
<code>\markdownRendererUlBegin</code>	99
<code>\markdownRendererUlBeginTight</code>	99
<code>\markdownRendererUlEnd</code>	100
<code>\markdownRendererUlEndTight</code>	101
<code>\markdownRendererUlItem</code>	100
<code>\markdownRendererUlItemEnd</code>	100
<code>\markdownRendererUntickedBox</code>	140
<code>\markdownRendererWarning</code>	141
<code>\markdownSetup</code>	62, 62, 66, 173, 174, 180, 478, 488
<code>\markdownSetupSnippet</code>	87, 87
<code>\markdownThemeVersion</code>	76, 76, 77
<code>\markdownWarning</code>	167, 167
<code>\markinline</code>	57, 59, 60, 169, 171, 470, 476
<code>\markinlinePlainTeX</code>	476
<code>\mmdcCommand</code>	441
<code>new</code>	7, 19, 20, 420, 422
<code>notes</code>	42, 125
<code>parsers</code>	284, 322
<code>parsers.commented_line</code>	302
<code>parsers.unicode</code>	286
<code>pipeTables</code>	7, 43, 49, 138
<code>preserveTabs</code>	44, 47, 323
<code>rawAttribute</code>	39, 44, 44, 131, 132
<code>read_decompositions</code>	185
<code>reader</code>	8, 33, 182, 284, 322, 369
<code>reader-&gt;add_special_character</code>	8, 9, 33, 363
<code>reader-&gt;auto_link_email</code>	351
<code>reader-&gt;auto_link_url</code>	351
<code>reader-&gt;create_parser</code>	323
<code>reader-&gt;finalize_grammar</code>	358, 427
<code>reader-&gt;initialize_named_group</code>	363
<code>reader-&gt;insert_pattern</code>	8, 9, 33, 359, 365
<code>reader-&gt;lookup_note_reference</code>	336
<code>reader-&gt;lookup_reference</code>	336
<code>reader-&gt;normalize_tag</code>	322

reader->options	322
reader->parser_functions	323
reader->parser_functions.name	323
reader->parsers	322, 322
reader->register_link	335
reader->update_rule	359, 362, 365
reader->writer	322
reader.new	322, 322, 427
relativeReferences	38, 45
serialize_byte_parser	184
serialize_byte_range_parser	184
serialize_replacement	184
\setupmarkdown	180, 180
\setupyaml	180
shiftHeadings	7, 45
singletonCache	19
slice	7, 45, 262, 277, 278
smartEllipses	46, 113, 175
\startmarkdown	178, 178, 524
startNumber	46, 127–129
\startyaml	30, 31, 41, 178, 178, 524
\stopmarkdown	178, 178, 524
\stopyaml	30, 31, 41, 178, 178, 524
strikeThrough	47, 136, 517
stripIndent	47, 323
stripPercentSigns	467, 468
subscripts	48, 137
superscripts	48, 137
syntax	360, 365
tableAttributes	48, 138, 513
tableCaptions	7, 48, 49, 138
taskLists	49, 140, 504
texComments	50, 324
texMathDollars	39, 50, 139
texMathDoubleBackslash	39, 51, 139
texMathSingleBackslash	39, 51, 139
tightLists	51, 99, 101, 111, 113, 126, 127, 130, 131, 491
underscores	52
unicode_data.casefold_mapping	196, 207
unicode_data.categories	198, 286

unicode_data.ccc	199, 207
unicode_data.composition_mapping	192, 211
unicode_data.decomposition_mapping	186, 188, 210
unicodeNormalization	20, 20, 21
unicodeNormalizationForm	20, 20
util.cache	202, 202
util.cache_verbatim	202
util.canonically_order	207
util.casefold	207
util.compose	211
util.decompose	210
util.encode_json_string	202
util.err	201
util.escaper	205
util.expand_tabs_in_line	202
util.find_file	214
util.find_files	214
util.flatten	203
util.intersperse	204
util.map	205
util.normalize	214
util.pathname	206
util.rope_last	204
util.rope_to_string	204
util.salt	206
util.table_copy	202
util.walk	203, 204
util.warning	206
walkable_syntax	8, 22, 29, 358, 359, 362, 363, 365
writer	182, 182, 261, 369
writer->acronym	369
writer->active_attributes	276, 276, 277
writer->attribute_type_levels	276
writer->attributes	274
writer->block_html_cdata_element	269
writer->block_html_cdata_section	270
writer->block_html_comment	270
writer->block_html_declaration	270
writer->block_html_element	269
writer->block_html_pdata_element	270
writer->block_html_processing_instruction	270



writer->block_html_standalone_tag	271
writer->blockquote	272
writer->bulletitem	268
writer->bulletlist	267
writer->citations	372
writer->code	266
writer->contentblock	377
writer->defer_call	284, 284
writer->definitionlist	379
writer->display_math	412
writer->div_begin	393
writer->div_end	393
writer->document	273
writer->ellipsis	264
writer->emphasis	272
writer->error	266
writer->escape	265
writer->escaped_chars	264, 265
writer->escaped_minimal_strings	264, 265
writer->escaped_strings	264
writer->escaped_uri_chars	264, 265
writer->fancyitem	382
writer->fancylist	382
writer->fencedCode	388
writer->flatten_inlines	262, 262
writer->get_state	283
writer->hard_line_break	263
writer->heading	282
writer->identifier	265
writer->image	267
writer->infostring	265
writer->inline_html_cdata_section	271
writer->inline_html_close_tag	271
writer->inline_html_comment	269
writer->inline_html_declaration	271
writer->inline_html_empty_tag	272
writer->inline_html_open_tag	271
writer->inline_html_processing_instruction	271
writer->inline_html_tag	269
writer->inline_math	412
writer->interblocksep	263
writer->is_writing	262, 262

writer->jekyllData	416
writer->lineblock	399
writer->link	266
writer->mark	400
writer->math	265
writer->nbsp	262
writer->note	402
writer->options	261
writer->ordereditem	269
writer->orderedlist	268
writer->paragraph	263
writer->paragraphsep	263
writer->plain	263
writer->pop_attributes	276, 277
writer->push_attributes	276, 277
writer->rawBlock	388
writer->rawInline	409
writer->set_state	284
writer->slice_begin	262
writer->slice_end	262
writer->soft_line_break	263
writer->space	262
writer->span	371
writer->strike_through	410
writer->string	265
writer->strong	272
writer->subscript	410
writer->superscript	411
writer->table	406
writer->thematic_break	264
writer->checkbox	272
writer->undosep	263, 367
writer->uri	265
writer->verbatim	273
writer->warning	206, 266
writer.new	261, 261, 427
\yaml	171
yaml	30, 31, 41, 169, 170, 171, 477
\yamlBegin	30, 31, 41, 57, 58, 59, 167, 170, 178
\yamlEnd	30, 31, 41, 57, 58, 59, 167, 170, 171, 178
\yamlInput	30, 31, 41, 57, 60, 169, 172, 180, 476

