

A Markdown Interpreter for T_EX

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

Version 3.11.5-0-g2f7bf8ca
2025-08-19

Contents

1	Introduction	1	3	Implementation	167
1.1	Requirements	2	3.1	Lua Implementation	168
1.2	Feedback	6	3.2	Plain T _E X Implementation	406
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation	449
2	Interfaces	7	3.4	ConT _E Xt Implementation	489
2.1	Lua Interface	7		References	500
2.2	Plain T _E X Interface	53		Index	502
2.3	L ^A T _E X Interface	153			
2.4	ConT _E Xt Interface	162			

List of Figures

1	A block diagram of the Markdown package	8
2	A sequence diagram of typesetting a document using the T _E X interface	49
3	A sequence diagram of typesetting a document using the Lua CLI	50
4	An example directed graph	76
5	An example mindmap	77
6	An example UML sequence diagram	78
7	The banner of the Markdown package	79
8	A pushdown automaton that recognizes T _E X comments	286

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             "2016-2024 Vít Starý Novotný, Andrej Genčur"},
9   license   = "LPPL 1.3c"
10 }
11
12 if not modules then modules = { } end
13 modules['markdown'] = metadata
```

1.1 Requirements

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

1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the Lua_{TEX} engine (though not necessarily in the LuaMeta_{TEX} engine).

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

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

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

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

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

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

```
16 ;(function() -- luacheck: ignore
```

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

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

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

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.

```
24 hard lua-tinyyaml
```

1.1.2 Plain TeX Requirements

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

expl3 A package that enables the expl3 language [2] from the L^AT_EX3 kernel in TeX Live ≤ 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
25 hard l3kernel
26 \unprotect
27 \expandafter\ifx\csname ExplSyntaxOn\endcsname\relax
28   \input expl3-generic
29 \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.

```
30 hard lt3luabridge
```

The plain T_EX 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 T_EX 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 LuaT_EX (T_EXLive \geq 2008).

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

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

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

1.1.3 L^AT_EX Requirements

The L^AT_EX part of the package requires that the L^AT_EX 2 _{ϵ} format is loaded, a T_EX engine that extends ϵ -T_EX, and all the plain T_EX prerequisites (see Section 1.1.2).

```
31 \NeedsTeXFormat{LaTeX2e}
32 \RequirePackage{expl3}
```

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or L^AT_EX 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.

```
33 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.

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

34 `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 [3] will also cause `enumitem` to be used. In a future major version, `enumitem` will replace `paralist` altogether.

35 `soft enumitem`

36 `soft paralist`

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

37 `soft fancyvrb`

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

38 `soft csvsimple`

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

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

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

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

42 `soft amsmath`

43 `soft amsfonts`

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.

44 `soft graphics`

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

46 `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 strikeouts and marked text in pdfT_EX.

47 `soft soul`

48 `soft xcolor`

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

```
49 soft lua-ul
50 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.

```
51 soft ltxcmds
```

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

```
52 soft luaxml
```

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

```
53 soft verse
```

1.1.4 Con \TeX t Prerequisites

The Con \TeX t part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain \TeX prerequisites (see Section 1.1.2), and the following Con \TeX t 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 \TeX - \LaTeX 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 [4] is gratefully acknowledged.

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

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

2 Interfaces

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

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

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

2.1 Lua Interface

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

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

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

2.1.1 Conversion from Markdown to Plain $\text{T}_{\text{E}}\text{X}$

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain $\text{T}_{\text{E}}\text{X}$ according to the table `options`

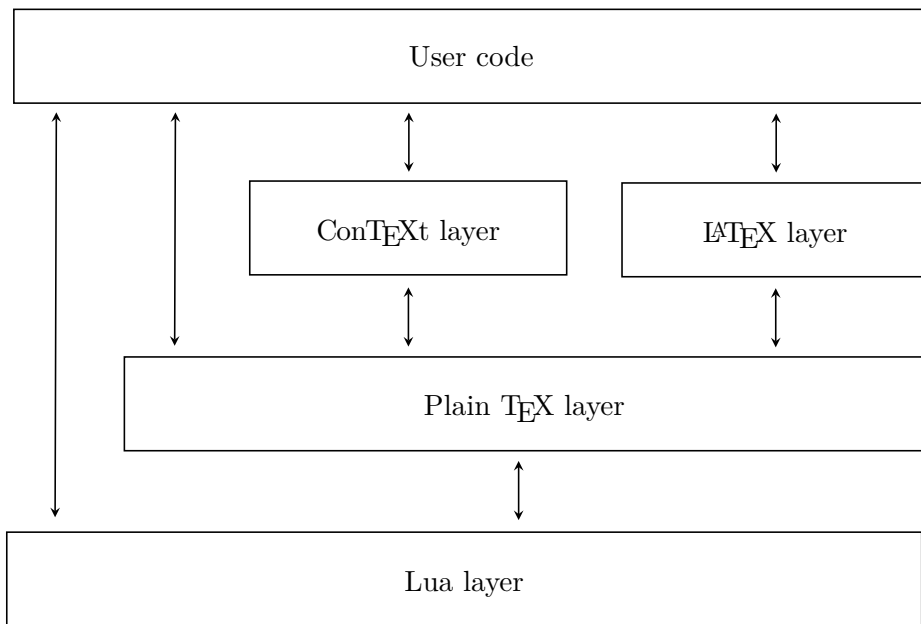


Figure 1: A block diagram of the Markdown package

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.

```

55 local walkable_syntax = {

```



```

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

```

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

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

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

2.1.3 Options

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

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

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

```
88 \ExplSyntaxOn
89 \seq_new:N \g_@@_lua_options_seq
```

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

```
90 \prop_new:N \g_@@_lua_option_types_prop
91 \prop_new:N \g_@@_default_lua_options_prop
92 \seq_new:N \g_@@_option_layers_seq
93 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
94 \seq_gput_right:NV
95   \g_@@_option_layers_seq
96   \c_@@_option_layer_lua_tl
97 \cs_new:Nn
98   \@@_add_lua_option:nnn
99   {
100     \@@_add_option:Vnnn
101       \c_@@_option_layer_lua_tl
102       { #1 }
103       { #2 }
104       { #3 }
105   }
106 \cs_new:Nn
107   \@@_add_option:nnnn
108   {
109     \seq_gput_right:cn
110       { g_@@_ #1 _options_seq }
111       { #2 }
112     \prop_gput:cnn
113       { g_@@_ #1 _option_types_prop }
114       { #2 }
115       { #3 }
116     \prop_gput:cnn
117       { g_@@_default_ #1 _options_prop }
118       { #2 }
119       { #4 }
120     \@@_typecheck_option:n
121       { #2 }
122   }
```

```

123 \cs_generate_variant:Nn
124   \@@_add_option:nnnn
125   { Vnnn }
126 \tl_const:Nn \c_@@_option_value_true_tl { true }
127 \tl_const:Nn \c_@@_option_value_false_tl { false }
128 \cs_new:Nn \@@_typecheck_option:n
129   {
130     \@@_get_option_type:nN
131     { #1 }
132     \l_tmpa_tl
133     \str_case_e:Vn
134     \l_tmpa_tl
135     {
136       { \c_@@_option_type_boolean_tl }
137       {
138         \@@_get_option_value:nN
139         { #1 }
140         \l_tmpa_tl
141         \bool_if:nF
142         {
143           \str_if_eq_p:VV
144           \l_tmpa_tl
145           \c_@@_option_value_true_tl ||
146           \str_if_eq_p:VV
147           \l_tmpa_tl
148           \c_@@_option_value_false_tl
149         }
150         {
151           \msg_error:nnnV
152           { markdown }
153           { failed-typecheck-for-boolean-option }
154           { #1 }
155           \l_tmpa_tl
156         }
157       }
158     }
159   }
160 \msg_new:nnn
161   { markdown }
162   { failed-typecheck-for-boolean-option }
163   {
164     Option~#1~has~value~#2,~
165     but~a~boolean~(true~or~false)~was~expected.
166   }
167 \cs_generate_variant:Nn
168   \str_case_e:nn
169   { Vn }

```

```

170 \cs_generate_variant:Nn
171   \msg_error:nnnn
172   { nnnV }
173 \seq_new:N
174   \g_@@_option_types_seq
175 \tl_const:Nn
176   \c_@@_option_type_clist_tl
177   { clist }
178 \seq_gput_right:NV
179   \g_@@_option_types_seq
180   \c_@@_option_type_clist_tl
181 \tl_const:Nn
182   \c_@@_option_type_counter_tl
183   { counter }
184 \seq_gput_right:NV
185   \g_@@_option_types_seq
186   \c_@@_option_type_counter_tl
187 \tl_const:Nn
188   \c_@@_option_type_boolean_tl
189   { boolean }
190 \seq_gput_right:NV
191   \g_@@_option_types_seq
192   \c_@@_option_type_boolean_tl
193 \tl_const:Nn
194   \c_@@_option_type_number_tl
195   { number }
196 \seq_gput_right:NV
197   \g_@@_option_types_seq
198   \c_@@_option_type_number_tl
199 \tl_const:Nn
200   \c_@@_option_type_path_tl
201   { path }
202 \seq_gput_right:NV
203   \g_@@_option_types_seq
204   \c_@@_option_type_path_tl
205 \tl_const:Nn
206   \c_@@_option_type_slice_tl
207   { slice }
208 \seq_gput_right:NV
209   \g_@@_option_types_seq
210   \c_@@_option_type_slice_tl
211 \tl_const:Nn
212   \c_@@_option_type_string_tl
213   { string }
214 \seq_gput_right:NV
215   \g_@@_option_types_seq
216   \c_@@_option_type_string_tl

```

```

217 \cs_new:Nn
218   \@@_get_option_type:nN
219   {
220     \bool_set_false:N
221       \l_tmpa_bool
222     \seq_map_inline:Nn
223       \g_@@_option_layers_seq
224       {
225         \prop_get:cnNT
226           { g_@@_ ##1 _option_types_prop }
227           { #1 }
228         \l_tmpa_tl
229         {
230           \bool_set_true:N
231             \l_tmpa_bool
232           \seq_map_break:
233         }
234       }
235     \bool_if:NF
236       \l_tmpa_bool
237       {
238         \msg_error:nnn
239           { markdown }
240           { undefined-option }
241           { #1 }
242       }
243     \seq_if_in:NVF
244       \g_@@_option_types_seq
245       \l_tmpa_tl
246       {
247         \msg_error:nnnV
248           { markdown }
249           { unknown-option-type }
250           { #1 }
251         \l_tmpa_tl
252       }
253     \tl_set_eq:NN
254       #2
255       \l_tmpa_tl
256   }
257 \msg_new:nnn
258   { markdown }
259   { unknown-option-type }
260   {
261     Option~#1~has~unknown~type~#2.
262   }
263 \msg_new:nnn

```

```

264 { markdown }
265 { undefined-option }
266 {
267   Option~#1~is~undefined.
268 }
269 \cs_new:Nn
270 \@@_get_default_option_value:nN
271 {
272   \bool_set_false:N
273     \l_tmpa_bool
274   \seq_map_inline:Nn
275     \g_@@_option_layers_seq
276     {
277       \prop_get:cnNT
278         { g_@@_default_ ##1 _options_prop }
279         { #1 }
280         #2
281         {
282           \bool_set_true:N
283             \l_tmpa_bool
284           \seq_map_break:
285         }
286       }
287   \bool_if:NF
288     \l_tmpa_bool
289     {
290       \msg_error:nnn
291         { markdown }
292         { undefined-option }
293         { #1 }
294     }
295 }
296 \cs_new:Nn
297 \@@_get_option_value:nN
298 {
299   \@@_option_tl_to_csname:nN
300     { #1 }
301   \l_tmpa_tl
302   \cs_if_free:cTF
303     { \l_tmpa_tl }
304     {
305       \@@_get_default_option_value:nN
306         { #1 }
307       #2
308     }
309   {
310     \@@_get_option_type:nN

```

```

311     { #1 }
312     \l_tmpa_tl
313     \str_if_eq:NNTF
314     \c_@@_option_type_counter_tl
315     \l_tmpa_tl
316     {
317         \@@_option_tl_to_csname:nN
318         { #1 }
319         \l_tmpa_tl
320         \tl_set:Nx
321         #2
322         { \the \cs:w \l_tmpa_tl \cs_end: } % noqa: W200
323     }
324     {
325         \@@_option_tl_to_csname:nN
326         { #1 }
327         \l_tmpa_tl
328         \tl_set:Nv
329         #2
330         { \l_tmpa_tl }
331     }
332 }
333 }
334 \cs_new:Nn \@@_option_tl_to_csname:nN
335 {
336     \tl_set:Nn
337     \l_tmpa_tl
338     { \str_uppercase:n { #1 } }
339     \tl_set:Nx
340     #2
341     {
342         markdownOption
343         \tl_head:f { \l_tmpa_tl }
344         \tl_tail:n { #1 }
345     }
346 }

```

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

```

347 \cs_new:Nn \@@_with_various_cases:nn
348 {
349     \seq_clear:N
350     \l_tmpa_seq
351     \seq_map_inline:Nn
352     \g_@@_cases_seq
353     {

```

```

354     \tl_set:Nn
355     \l_tmpa_tl
356     { #1 }
357     \use:c { ##1 }
358     \l_tmpa_tl
359     \seq_put_right:NV
360     \l_tmpa_seq
361     \l_tmpa_tl
362   }
363   \seq_map_inline:Nn
364   \l_tmpa_seq
365   { #2 }
366 }

```

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

```

367 \seq_new:N \g_@@_cases_seq
368 \cs_new:Nn \@@_camel_case:N % noqa: w401
369 {
370   \regex_replace_all:nnN
371   { _ ([a-z]) }
372   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
373   #1
374   \tl_set:Nx
375   #1
376   { #1 }
377 }
378 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
379 \cs_new:Nn \@@_snake_case:N % noqa: w401
380 {
381   \regex_replace_all:nnN
382   { ([a-z])([A-Z]) }
383   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
384   #1
385   \tl_set:Nx
386   #1
387   { #1 }
388 }
389 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 General Behavior

`eagerCache=true, false`

default: `true`

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

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

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

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

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

```
390 \@@_add_lua_option:nnn
391   { eagerCache }
392   { boolean }
393   { true }
394 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 just means that the version `experimental` of the theme `witiko/markdown/defaults` will be loaded and warnings for hard-deprecated features will become errors. However, the effects may extend to other areas in the future as well.

`false` Experimental features will be disabled.

```
395 \@@_add_lua_option:nnn
396   { experimental }
397   { boolean }
398   { false }
399 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.

```
400 \@@_add_lua_option:nnn
401   { singletonCache }
402   { boolean }
403   { true }
404 defaultOptions.singletonCache = true
405 local singletonCache = {
406   convert = nil,
407   options = nil,
408 }
```

`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.

```
409 \@@_add_lua_option:nnn
410   { unicodeNormalization }
411   { boolean }
412   { true }
413 defaultOptions.unicodeNormalization = true
```

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

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

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

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

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

418 defaultOptions.unicodeNormalizationForm = "nfc"
```

2.1.5 File and Directory Names

`cacheDir=<path>` default: `.`

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

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

```
419 \str_new:N
420 \g_@@_unquoted_jobname_str
421 \str_gset:NV
422 \g_@@_unquoted_jobname_str
423 \c_sys_jobname_str
424 \bool_new:N
425 \g_@@_jobname_quoted_bool
```

```

426 \regex_replace_all:nnNTF
427 { \A ("|') ( .* ) ("|') \Z }
428 { \2 }
429 \g_@@_unquoted_jobname_str
430 {
431     \bool_gset_true:N
432     \g_@@_jobname_quoted_bool
433 }
434 {
435     \bool_gset_false:N
436     \g_@@_jobname_quoted_bool
437 }
438 \@@_add_lua_option:nnn
439 { cacheDir }
440 { path }
441 {
442     \markdownOptionOutputDir
443     / _markdown_
444     \str_use:N
445     \g_@@_unquoted_jobname_str
446 }
447 defaultOptions.cacheDir = "."

```

`contentBlocksLanguageMap`=*<filename>*

default: `markdown-languages.json`

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

```

448 \@@_add_lua_option:nnn
449 { contentBlocksLanguageMap }
450 { path }
451 { markdown-languages.json }
452 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.

```

453 \@@_add_lua_option:nnn
454 { debugExtensionsFileName }
455 { path }
456 {
457   \markdownOptionOutputDir
458   /
459   \str_use:N
460   \g_@@_unquoted_jobname_str
461   .debug-extensions.json
462 }
463 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

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

```

464 \@@_add_lua_option:nnn
465 { frozenCacheFileName }
466 { path }
467 { \markdownOptionCacheDir / frozenCache.tex }
468 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

2.1.6 Parser Options

`autoIdentifiers`=true, false default: false

true Enable the Pandoc auto identifiers syntax extension⁹:

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

```
# 123 Sesame Street
```

false Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```
469 \@@_add_lua_option:nnn
```

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

```
470 { autoIdentifiers }
471 { boolean }
472 { false }

473 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.

```
474 \@@_add_lua_option:nnn
475 { blankBeforeBlockquote }
476 { boolean }
477 { false }

478 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.

```
479 \@@_add_lua_option:nnn
480 { blankBeforeCodeFence }
481 { boolean }
482 { false }

483 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.

```
484 \@@_add_lua_option:nnn
485 { blankBeforeDivFence }
486 { boolean }
487 { false }

488 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.

```
489 \@@_add_lua_option:nnn
490 { blankBeforeHeading }
491 { boolean }
492 { false }
493 defaultOptions.blankBeforeHeading = false
```

`blankBeforeList=true, false` default: false

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

```
494 \@@_add_lua_option:nnn
495 { blankBeforeList }
496 { boolean }
497 { false }
498 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

- `true` Enable the Pandoc bracketed span syntax extension¹⁰:

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

- `false` Disable the Pandoc bracketed span syntax extension.

```
499 \@@_add_lua_option:nnn
500 { bracketedSpans }
501 { boolean }
502 { false }
503 defaultOptions.bracketedSpans = false
```

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

`breakableBlockquotes=true, false`

default: true

- `true` A blank line separates block quotes.
- `false` Blank lines in the middle of a block quote are ignored.

```
504 \@@_add_lua_option:nnn
505 { breakableBlockquotes }
506 { boolean }
507 { true }

508 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.

```
509 \@@_add_lua_option:nnn
510 { citationNbsps }
511 { boolean }
512 { true }

513 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: false

- `true` Enable the Pandoc citation syntax extension¹¹:

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,

¹¹See <https://pandoc.org/MANUAL.html#extension-citations>.


```
chap. 1]. Here is one with the name of the author
suppressed -@doe99.
```

`false` Disable the Pandoc citation syntax extension.

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

518 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.``
```

```
519 \@@_add_lua_option:nnn
520   { codeSpans }
521   { boolean }
522   { true }

523 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

`true`

: Enable the iA Writer content blocks syntax extension [5]:

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

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

```
524 \@@_add_lua_option:nnn
525   { contentBlocks }
526   { boolean }
527   { false }

528 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
```

```
529 \@@_add_lua_option:nnn
530   { contentLevel }
531   { string }
532   { block }

533 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.

```
534 \@@_add_lua_option:nnn
535   { debugExtensions }
536   { boolean }
537   { false }

538 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.

```
539 \@@_add_lua_option:nnn
540   { definitionLists }
541   { boolean }
542   { false }

543 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: false

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

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

```
544 \@@_add_lua_option:nnn
545   { ensureJekyllData }
546   { boolean }
547   { false }

548 defaultOptions.ensureJekyllData = false
```

`expectJekyllData=true, false`

default: `false`

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

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

`true` When the `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}

```

```

549 \@@_add_lua_option:nnn
550   { expectJekyllData }
551   { boolean }
552   { false }

553 defaultOptions.expectJekyllData = false

```

`extensions=<filenames>`

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

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

```

local strike_through = {
  api_version = 2,
  grammar_version = 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:

```
554 metadata.user_extension_api_version = 2
555 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.

```
556 \cs_generate_variant:Nn
557   \@@_add_lua_option:nnn
558   { nnV }
559 \@@_add_lua_option:nnV
560   { extensions }
561   { clist }
562   \c_empty_clist
563 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: `false`

`true` Enable the Pandoc fancy list syntax extension<sup>12</sup>:

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

`false` Disable the Pandoc fancy list syntax extension.

```
564 \@@_add_lua_option:nnn
565   { fancyLists }
566   { boolean }
567   { false }
568 defaultOptions.fancyLists = false
```

---

<sup>12</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

fencedCode=true, false

default: true

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

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

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

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

```
569 \@@_add_lua_option:nnn
570 { fencedCode }
571 { boolean }
572 { true }

573 defaultOptions.fencedCode = true
```

fencedCodeAttributes=true, false

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>13</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.

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_code\\_attributes](https://pandoc.org/MANUAL.html#extension-fenced_code_attributes).

```

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

578 defaultOptions.fencedCodeAttributes = false

```

`fencedDivs=true, false`

default: false

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

```

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

And another.
:::::

```

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

```

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

583 defaultOptions.fencedDivs = false

```

`finalizeCache=true, false`

default: false

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

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

```

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

588 defaultOptions.finalizeCache = false

```

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



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

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

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

```
589 \@@_add_lua_option:nmn
590 { frozenCacheCounter }
591 { counter }
592 { 0 }

593 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers`=true, false default: false

**true** Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>15</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`.

```
594 \@@_add_lua_option:nmn
595 { gfmAutoIdentifiers }
596 { boolean }
597 { false }

598 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.

---

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

```

599 \@@_add_lua_option:nnn
600   { hashEnumerators }
601   { boolean }
602   { false }

603 defaultOptions.hashEnumerators = false

```

`headerAttributes=true, false`

default: false

**true** Enable the assignment of HTML attributes to headings:

```

# My first heading {#foo}

## My second heading ##   {#bar .baz}

Yet another heading   {key=value}
=====

```

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

```

604 \@@_add_lua_option:nnn
605   { headerAttributes }
606   { boolean }
607   { false }

608 defaultOptions.headerAttributes = false

```

`html=true, false`

default: true

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

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

```

609 \@@_add_lua_option:nnn
610   { html }
611   { boolean }
612   { true }

613 defaultOptions.html = true

```

`hybrid=true, false`

default: `false`

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

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

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

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

```
Here is a mathematical formula:  
  
/math-formula.tex
```

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

```
`$H_2 O$`{=tex} is a liquid.  
  
Here is a mathematical formula:  
... {=tex}  
\[distance[i] =  
    \begin{dcases}  
        a & b \\  
        c & d  
    \end{dcases}  
\]
```

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

```
$H_2 O$ is a liquid.

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

```
614 \@@_add_lua_option:nnn
615   { hybrid }
616   { boolean }
617   { false }

618 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: false

**true** Enable the Pandoc inline code span attribute extension<sup>16</sup>:

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

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

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

623 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

**true** Enable the Pandoc inline note syntax extension<sup>17</sup>:

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

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

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

`false` Disable the Pandoc inline note syntax extension.

```
624 \@@_add_lua_option:nnn
625 { inlineNotes }
626 { boolean }
627 { false }
628 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>18</sup> for entering metadata in YAML:

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

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

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

```
629 \@@_add_lua_option:nnn
630 { jeekyllData }
631 { boolean }
632 { false }
633 defaultOptions.jekyllData = false
```

`linkAttributes=true, false` default: false

`true` Enable the Pandoc link and image attribute syntax extension<sup>19</sup>:

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

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

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).

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

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

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

638 defaultOptions.linkAttributes = false
```

`lineBlocks=true, false` default: false

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

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

`false` Disable the Pandoc line block syntax extension.

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

643 defaultOptions.lineBlocks = false
```

`mark=true, false` default: false

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

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

`false` Disable the Pandoc mark syntax extension.

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

648 defaultOptions.mark = false
```

---

<sup>20</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

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

`notes=true, false`

default: false

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

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

[^1]: Here is the note.

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

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

        { some.code }

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

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

`false` Disable the Pandoc note syntax extension.

```
649 \@@_add_lua_option:nnm
650 { notes }
651 { boolean }
652 { false }

653 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.

---

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

```

654 \@@_add_lua_option:nnn
655   { pipeTables }
656   { boolean }
657   { false }

658 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.

```

659 \@@_add_lua_option:nnn
660   { preserveTabs }
661   { boolean }
662   { true }

663 defaultOptions.preserveTabs = true

```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>23</sup>:

```

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

```

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

```

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

```

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

**false** Disable the Pandoc raw attribute syntax extension.

<sup>23</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](https://pandoc.org/MANUAL.html#extension-raw_attribute).



```

664 \@@_add_lua_option:nnn
665   { rawAttribute }
666   { boolean }
667   { false }

668 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false`

default: false

`true` Enable relative references<sup>24</sup> in autolinks:

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

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

`false` Disable relative references in autolinks.

```

669 \@@_add_lua_option:nnn
670   { relativeReferences }
671   { boolean }
672   { false }

673 defaultOptions.relativeReferences = false

```

`shiftHeadings=<shift amount>`

default: 0

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

```

674 \@@_add_lua_option:nnn
675   { shiftHeadings }
676   { number }
677   { 0 }

678 defaultOptions.shiftHeadings = 0

```

<sup>24</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

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

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

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

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

```
679 \@@_add_lua_option:nnn
680 { slice }
681 { slice }
682 { ^~$ }
683 defaultOptions.slice = "^ $"
```

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

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

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

```
684 \@@_add_lua_option:nnn
685 { smartEllipses }
686 { boolean }
687 { false }
688 defaultOptions.smartEllipses = false
```

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

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

**false** Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOliItem` T<sub>E</sub>X macro.

```
689 \@@_add_lua_option:nnn
690 { startNumber }
691 { boolean }
692 { true }
693 defaultOptions.startNumber = true
```

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

**true** Enable the Pandoc strike-through syntax extension<sup>25</sup>:

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

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

```
694 \@@_add_lua_option:nnn
695 { strikeThrough }
696 { boolean }
697 { false }
698 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.

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

---

<sup>25</sup>See <https://pandoc.org/MANUAL.html#extension-strikeout>.

`subscripts=true, false` default: false

`true` Enable the Pandoc subscript syntax extension<sup>26</sup>:

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

`false` Disable the Pandoc subscript syntax extension.

```
704 \@@_add_lua_option:nnn
705 { subscripts }
706 { boolean }
707 { false }

708 defaultOptions.subscripts = false
```

`superscripts=true, false` default: false

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

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

`false` Disable the Pandoc superscript syntax extension.

```
709 \@@_add_lua_option:nnn
710 { superscripts }
711 { boolean }
712 { false }

713 defaultOptions.superscripts = false
```

`tableAttributes=true, false` default: false

`true`

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

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

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

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

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

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

```
714 \@@_add_lua_option:nnn
715   { tableAttributes }
716   { boolean }
717   { false }

718 defaultOptions.tableAttributes = false
```

**tableCaptions=true, false**

default: false

**true**

: Enable the Pandoc table caption syntax extension<sup>28</sup> for pipe tables (see the **pipeTables** option).

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

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

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

```
719 \@@_add_lua_option:nnn
720   { tableCaptions }
721   { boolean }
722   { false }

723 defaultOptions.tableCaptions = false
```

**taskLists=true, false**

default: false

**true**      Enable the Pandoc task list syntax extension<sup>29</sup>:

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

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

<sup>28</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>29</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

```

724 \@_add_lua_option:nnn
725   { taskLists }
726   { boolean }
727   { false }

728 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

**true** Strip TeX-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 TeX-style comments.

```

729 \@_add_lua_option:nnn
730   { texComments }
731   { boolean }
732   { false }

733 defaultOptions.texComments = false

```

`texMathDollars=true, false`

default: false

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

```

inline math: $E=mc^2$

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

```

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

```

734 \@_add_lua_option:nnn
735   { texMathDollars }
736   { boolean }
737   { false }

738 defaultOptions.texMathDollars = false

```

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

`texMathDoubleBackslash=true, false` default: false

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

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

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

```
739 \\@@_add_lua_option:nnn
740 { texMathDoubleBackslash }
741 { boolean }
742 { false }

743 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

```
744 \\@@_add_lua_option:nnn
745 { texMathSingleBackslash }
746 { boolean }
747 { false }

748 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 renders that may produce different output than lists that are not tight:

---

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

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

```

- This is
- a tight
- unordered list.

- This is

  not a tight

- unordered list.

```

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

```

749 \@@_add_lua_option:nmn
750 { tightLists }
751 { boolean }
752 { true }

753 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.

```

754 \@@_add_lua_option:nmn
755 { underscores }
756 { boolean }
757 { true }
758 \ExplSyntaxOff

759 defaultOptions.underscores = true

```

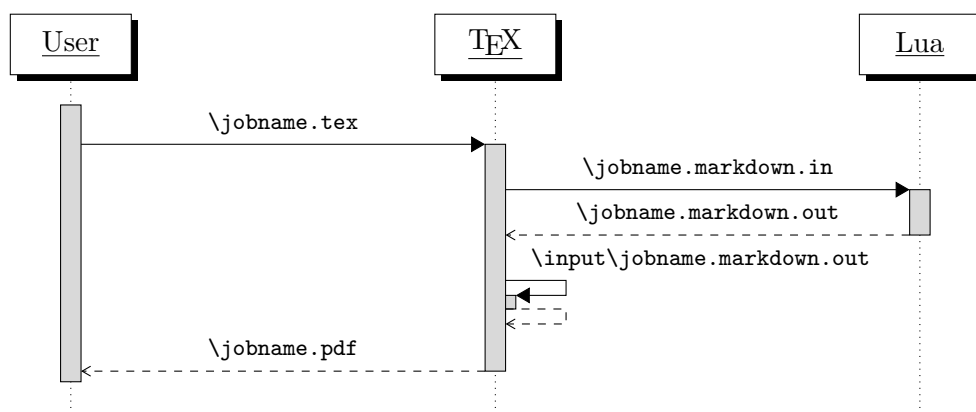


### 2.1.7 Command-Line Interface

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

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

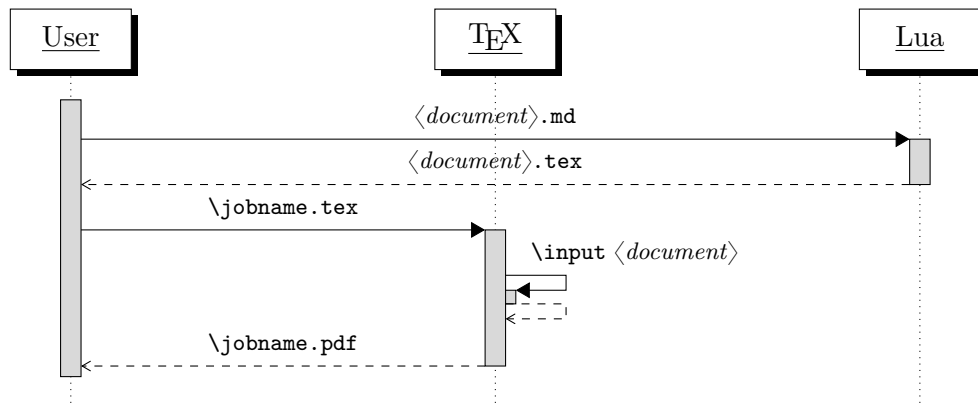
A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to  $\text{\TeX}$  is also provided, see Figure 3.



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

```

760 .TH MARKDOWN2TEX 1 "(((LASTMODIFIED)))"
761 .SH NAME
762 markdown2tex \- convert .md files to .tex
763 .SH SYNOPSIS
</lua-cli-manpage> <*lua-cli>
764 local HELP_STRING = "Usage: " .. [[
</lua-cli> <*lua-cli,lua-cli-manpage>
765 markdown2tex [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
766
</lua-cli,lua-cli-manpage> <*lua-cli-manpage>
767 .SH DESCRIPTION
  
```



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

```

768 % \end{macrocode}
769 </lua-cli-manpage>
770 <*lua-cli, lua-cli-manpage>
771 % \begin{macrocode}
772 OPTIONS are documented in Section 2.2.1 of the Markdown Package User
773 Manual (https://ctan.org/pkg/markdown).
774
775 When OUTPUT_FILE is unspecified, the result of the conversion will be
776 written to the standard output. When INPUT_FILE is also unspecified, the
777 result of the conversion will be read from the standard input.
778 % \end{macrocode}
779 </lua-cli, lua-cli-manpage>
780 <*lua-cli>
781 % \begin{macrocode}
782
783 Report bugs to: witiko@mail.muni.cz
784 Markdown package home page: <https://github.com/witiko/markdown>]]
785
786 local VERSION_STRING = [[
787 markdown2tex (Markdown) ]] .. metadata.version .. [[
788
789 Copyright (C) ]] .. table.concat(metadata.copyright,
790                                     "\nCopyright (C) ") .. [[
791
792 License: ]] .. metadata.license
793
794 local function warn(s)
795   io.stderr:write("Warning: " .. s .. "\n")
796 end
  
```

```

797
798 local function error(s)
799   io.stderr:write("Error: " .. s .. "\n")
800   os.exit(1)
801 end

```

To make it easier to copy-and-paste options from Pandoc [6] 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 [7] also show that `snake_case` is faster to read than camelCase.

```

802 local function camel_case(option_name)
803   local cased_option_name = option_name:gsub("_(%l)", function(match)
804     return match:sub(2, 2):upper()
805   end)
806   return cased_option_name
807 end
808
809 local function snake_case(option_name)
810   local cased_option_name = option_name:gsub("%l%u", function(match)
811     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
812   end)
813   return cased_option_name
814 end
815
816 local cases = {camel_case, snake_case}
817 local various_case_options = {}
818 for option_name, _ in pairs(defaultOptions) do
819   for _, case in ipairs(cases) do
820     various_case_options[case(option_name)] = option_name
821   end
822 end
823
824 local process_options = true
825 local options = {}
826 local input_filename
827 local output_filename
828 for i = 1, #arg do
829   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.

```

830     if arg[i] == "--" then
831       process_options = false
832       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.

```

833     elseif arg[i]:match("=") then
834         local key, value = arg[i]:match("(.-)=(.*)")
835         if defaultOptions[key] == nil and
836            various_case_options[key] ~= nil then
837             key = various_case_options[key]
838         end

```

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

```

839         local default_type = type(defaultOptions[key])
840         if default_type == "boolean" then
841             options[key] = (value == "true")
842         elseif default_type == "number" then
843             options[key] = tonumber(value)
844         elseif default_type == "table" then
845             options[key] = {}
846             for item in value:gmatch("[^,]+") do
847                 table.insert(options[key], item)
848             end
849         else
850             if default_type ~= "string" then
851                 if default_type == "nil" then
852                     warn('Option "' .. key .. '" not recognized.')
853                 else
854                     warn('Option "' .. key .. '" type not recognized, ' ..
855                         'please file a report to the package maintainer.')
856                 end
857                 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
858                     key .. '" as a string.')
859             end
860             options[key] = value
861         end
862         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.

```

863     elseif arg[i] == "--help" or arg[i] == "-h" then
864         print(HELP_STRING)
865         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.

```

866     elseif arg[i] == "--version" or arg[i] == "-v" then
867         print(VERSION_STRING)
868         os.exit()
869     end
870 end

```

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

```

871 if input_filename == nil then
872     input_filename = arg[i]

```

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

```

873 elseif output_filename == nil then
874     output_filename = arg[i]
875 else
876     error('Unexpected argument: "' .. arg[i] .. "'.')
877 end
878 ::continue::
879 end

```

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

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

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

```
\input hello
```

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

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

```

880 \def\markdownLastModified{((LASTMODIFIED))}%
881 \def\markdownVersion{((VERSION))}%

```

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

```
\input markdown
```

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

```
882 \let\markdownBegin\relax
883 \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  $\TeX$  [8, 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  $\TeX$  code, the characters `c`, `e`, and `f` will not appear in the output.

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

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

The following example plain  $\TeX$  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.

```

884 \let\uyamlBegin\uyamlBegin\relax
885 \def\uyamlEnd{\uyamlEnd\endgroup}

```

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

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

```

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

```

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

```

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

```

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

```

886 \let\markinline\relax

```

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

```

\input markdown
\markinline{ _Hello_ world }
\bye

```

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

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

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

### 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` T<sub>E</sub>X primitive to include T<sub>E</sub>X documents. The `\markdownInput` macro accepts a single parameter with the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X.

```
887 \let\markdownInput\relax
```

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

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

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

You can use the `\yamlInput` macro to include YAML documents, similarly to how you might use the `\input` T<sub>E</sub>X primitive to include T<sub>E</sub>X 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 T<sub>E</sub>X.

```
888 \def\yamlInput#1{%
889   \begingroup
890   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
891   \markdownInput{#1}%
892   \endgroup
893 }%
```

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

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



```
\input markdown
\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.

```
894 \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.

```
895 \ExplSyntaxOn
896 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
897 \cs_generate_variant:Nn
898   \tl_const:Nn
899   { NV }
900 \tl_if_exist:NF
901   \c_@@_top_layer_tl
902   {
903     \tl_const:NV
904       \c_@@_top_layer_tl
905       \c_@@_option_layer_plain_tex_tl
906   }
```

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

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

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

```

908 \prop_new:N \g_@@_plain_tex_option_types_prop
909 \prop_new:N \g_@@_default_plain_tex_options_prop
910 \seq_gput_right:NV
911   \g_@@_option_layers_seq
912   \c_@@_option_layer_plain_tex_tl
913 \cs_new:Nn
914   \@@_add_plain_tex_option:nnn
915   {
916     \@@_add_option:Vnnn
917     \c_@@_option_layer_plain_tex_tl
918     { #1 }
919     { #2 }
920     { #3 }
921   }

```

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

```

922 \cs_new:Nn
923   \@@_setup:n
924   {
925     \keys_set:nn
926       { markdown/options }
927       { #1 }
928   }
929 \cs_gset_eq:NN
930   \markdownSetup
931   \@@_setup:n

```

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

```

932 \cs_gset_eq:NN
933   \yamlSetup
934   \markdownSetup

```

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

```

935 \prg_new_conditional:Nnn % noqa: w401
936   \@@_if_option:n
937   { TF, T, F }
938   {

```

```

939 \@@_get_option_type:nN
940   { #1 }
941   \l_tmpa_tl
942 \str_if_eq:NNF
943   \l_tmpa_tl
944   \c_@@_option_type_boolean_tl
945   {
946     \msg_error:nxxx
947     { markdown }
948     { expected-boolean-option }
949     { #1 }
950     { \l_tmpa_tl }
951   }
952 \@@_get_option_value:nN
953   { #1 }
954   \l_tmpa_tl
955 \str_if_eq:NNTF
956   \l_tmpa_tl
957   \c_@@_option_value_true_tl
958   { \prg_return_true: }
959   { \prg_return_false: }
960 }
961 \msg_new:nnn
962 { markdown }
963 { expected-boolean-option }
964 {
965   Option~#1~has~type~#2,~
966   but~a~boolean~was~expected.
967 }
968 \let
969 \markdownIfOption
970 \@@_if_option:nTF

```

### 2.2.2.1 Finalizing and Freezing the Cache

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

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

```

971 \@@_add_plain_tex_option:nnn
972   { frozenCache }

```

```

973 { boolean }
974 { 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 T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X 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 T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`.

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

```

975 \tl_set:Nn
976   \l_tmpa_tl
977   {
978     \str_use:N
979     \g_@@_unquoted_jobname_str
980     .markdown.in
981   }
982 \bool_if:NT
983   \g_@@_jobname_quoted_bool
984   {
985     \tl_put_left:Nn
986       \l_tmpa_tl
987       { " }
988     \tl_put_right:Nn
989       \l_tmpa_tl
990       { " }
991   }
992 \cs_generate_variant:Nn
993   \@@_add_plain_tex_option:nnn
994   { nnV }
995 \@@_add_plain_tex_option:nnV
996   { inputTempFileName }
997   { path }
998   \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 T<sub>E</sub>X implementation. The option defaults to `.` or, since T<sub>E</sub>X Live 2024, to the value of the `-output-directory` option of your T<sub>E</sub>X engine.

In MikTeX, this automatic detection is currently only supported with LuaTeX<sup>33</sup>. If you need to use MikTeX and cannot use LuaTeX, you can either a) fix the automatic detection by setting the environmental variable `TEXMF_OUTPUT_DIRECTORY` manually or by setting the `\markdownOptionOutputDir` option manually.

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

```

999 \@@_add_plain_tex_option:nnn
1000 { outputDir }
1001 { path }
1002 { . }

```

### 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 TeX formats such as L<sup>A</sup>T<sub>E</sub>X and ConTeXt. 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 TeX formats should only use the plain TeX default definitions or whether they should also use the format-specific default definitions. Whereas plain TeX 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 TeX default definitions load no extra resources and are rather stable, whereas format-specific default definitions load extra resources and are subject to a more rapid change.

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

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

Here is how you would enable the macro in a ConTeXt 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.

```
1003 \@@_add_plain_tex_option:nnn
```

<sup>33</sup>See <https://github.com/MiKTeX/miktex/issues/1630>.

```

1004 { plain }
1005 { boolean }
1006 { 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.

```

1007 \@@_add_plain_tex_option:nnn
1008 { noDefaults }
1009 { boolean }
1010 { 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 [9] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

1011 \seq_gput_right:Nn
1012 \g_@@_plain_tex_options_seq
1013 { stripPercentSigns }
1014 \prop_gput:Nnn
1015 \g_@@_plain_tex_option_types_prop
1016 { stripPercentSigns }
1017 { boolean }
1018 \prop_gput:Nnx
1019 \g_@@_default_plain_tex_options_prop
1020 { stripPercentSigns }
1021 { false }

```

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

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

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

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

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

```
1022 \cs_new:Nn
1023   \@@_define_option_commands_and_keyvals:
1024   {
1025     \seq_map_inline:Nn
1026       \g_@@_option_layers_seq
1027       {
1028         \seq_map_inline:cn
1029           { g_@@_ ##1 _options_seq }
1030           {
1031             \@@_define_option_command:n
1032               { #####1 }
```

To make it easier to copy-and-paste options from Pandoc [6] 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 [7] also show that `snake_case` is faster to read than `camelCase`.

```
1033           \@@_with_various_cases:nn
1034             { #####1 }
1035             {
1036               \@@_define_option_keyval:nnn
1037                 { ##1 }
1038                 { #####1 }
1039                 { #####1 }
1040             }
1041         }
1042     }
1043 }
1044 \cs_new:Nn
1045   \@@_define_option_command:n
1046   {
```

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

```

1047 \str_if_eq:nnTF
1048   { #1 }
1049   { outputDir }
1050   { \@@_define_option_command_output_dir: }
1051   {

```

Do not override options defined before loading the package.

```

1052     \@@_option_tl_to_csname:nN
1053     { #1 }
1054     \l_tmpa_tl
1055     \cs_if_exist:cF
1056     { \l_tmpa_tl }
1057     {
1058         \@@_get_default_option_value:nN
1059         { #1 }
1060         \l_tmpa_tl
1061         \@@_set_option_value:nV
1062         { #1 }
1063         \l_tmpa_tl
1064     }
1065 }
1066 }
1067 \ExplSyntaxOff
1068 \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 [1, 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.

```

1069 \ExplSyntaxOn
1070 \cs_new:Nn
1071   \@@_define_option_command_output_dir:
1072   {
1073     \cs_if_free:NT
1074       \markdownOptionOutputDir
1075     {
1076       \bool_if:nTF
1077         {
1078           \cs_if_exist_p:N
1079             \luabridge_tl_set:Nn &&

```



```

1080      (
1081      \int_compare_p:nNn
1082      { \g_luabridge_method_int }
1083      =
1084      { \c_luabridge_method_directlua_int } ||
1085      \sys_if_shell_unrestricted_p:
1086      )
1087    }
1088    {

```

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.

```

1089      \group_begin:
1090      \cctab_select:N
1091      \c_str_cctab
1092      \luabridge_tl_set:Nn
1093      \l_tmpa_tl
1094      {
1095      print(
1096      (status.output_directory)
1097      or~os.getenv("TEXMF_OUTPUT_DIRECTORY")
1098      or~".")
1099      )
1100      }
1101      \tl_gset:NV
1102      \markdownOptionOutputDir
1103      \l_tmpa_tl
1104      \group_end:
1105    }
1106    {
1107      \tl_gset:Nn
1108      \markdownOptionOutputDir
1109      { . }
1110    }
1111  }
1112 }
1113 \cs_new:Nn
1114 \@@_set_option_value:nn
1115 {
1116   \@@_define_option:n
1117   { #1 }
1118   \@@_get_option_type:nN
1119   { #1 }
1120   \l_tmpa_tl
1121   \str_if_eq:NNTF
1122   \c_@@_option_type_counter_tl
1123   \l_tmpa_tl

```

```

1124     {
1125     \@@_option_tl_to_csname:nN
1126     { #1 }
1127     \l_tmpa_tl
1128     \int_gset:cn
1129     { \l_tmpa_tl }
1130     { #2 }
1131     }
1132     {
1133     \@@_option_tl_to_csname:nN
1134     { #1 }
1135     \l_tmpa_tl
1136     \cs_set:cpn
1137     { \l_tmpa_tl }
1138     { #2 }
1139     }
1140   }
1141 \cs_generate_variant:Nn
1142 \@@_set_option_value:mn
1143 { nV }
1144 \cs_new:Nn
1145 \@@_define_option:n
1146 {
1147   \@@_option_tl_to_csname:nN
1148   { #1 }
1149   \l_tmpa_tl
1150   \cs_if_free:cT
1151   { \l_tmpa_tl }
1152   {
1153     \@@_get_option_type:nN
1154     { #1 }
1155     \l_tmpb_tl
1156     \str_if_eq:NNT
1157     \c_@@_option_type_counter_tl
1158     \l_tmpb_tl
1159     {
1160       \@@_option_tl_to_csname:nN
1161       { #1 }
1162       \l_tmpa_tl
1163       \int_new:c
1164       { \l_tmpa_tl }
1165     }
1166   }
1167 }
1168 \cs_new:Nn
1169 \@@_define_option_keyval:nnn
1170 {

```

```

1171 \prop_get:cnN
1172   { g_@@_ #1 _option_types_prop }
1173   { #2 }
1174   \l_tmpa_tl
1175 \str_if_eq:VVTF
1176   \l_tmpa_tl
1177   \c_@@_option_type_boolean_tl
1178   {
1179     \keys_define:nn
1180       { markdown/options }
1181     {

```

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

```

1182         #3 .code:n = {
1183           \tl_set:Nx
1184             \l_tmpa_tl
1185             {
1186               \str_case:nnF
1187                 { ##1 }
1188                 {
1189                   { yes } { true }
1190                   { no } { false }
1191                 }
1192                 { ##1 }
1193             }
1194           \@@_set_option_value:nV
1195             { #2 }
1196             \l_tmpa_tl
1197         },
1198         #3 .default:n = { true },
1199     }
1200 }
1201 {
1202   \keys_define:nn
1203     { markdown/options }
1204     {
1205       #3 .code:n = {
1206         \@@_set_option_value:nn
1207           { #2 }
1208           { ##1 }
1209       },
1210     }
1211 }

```

For options of type **clist**, we assume that  $\langle key \rangle$  is a regular English noun in plural (such as **extensions**) and we also define the  $\langle singular\ key \rangle = \langle value \rangle$  interface, where  $\langle singular\ key \rangle$  is  $\langle key \rangle$  after stripping the trailing -s (such as **extension**). Rather

than setting the option to  $\langle value \rangle$ , this interface appends  $\langle value \rangle$  to the current value as the rightmost item in the list.

```

1212   \str_if_eq:VVT
1213     \l_tmpa_tl
1214     \c_@@_option_type_clist_tl
1215     {
1216       \tl_set:Nn
1217         \l_tmpa_tl
1218         { #3 }
1219       \tl_reverse:N
1220         \l_tmpa_tl
1221       \str_if_eq:enF
1222         {
1223           \tl_head:V
1224             \l_tmpa_tl
1225         }
1226         { s }
1227         {
1228           \msg_error:nnn
1229             { markdown }
1230             { malformed-name-for-clist-option }
1231             { #3 }
1232         }
1233       \tl_set:Nx
1234         \l_tmpa_tl
1235         {
1236           \tl_tail:V
1237             \l_tmpa_tl
1238         }
1239       \tl_reverse:N
1240         \l_tmpa_tl
1241       \tl_put_right:Nn
1242         \l_tmpa_tl
1243         {
1244           .code:n = {
1245             \@@_get_option_value:nN
1246               { #2 }
1247               \l_tmpa_tl
1248             \clist_set:NV
1249               \l_tmpa_clist
1250               { \l_tmpa_tl , { ##1 } }
1251             \@@_set_option_value:nV
1252               { #2 }
1253               \l_tmpa_clist
1254           }
1255         }
1256       \keys_define:nV

```

```

1257         { markdown/options }
1258         \l_tmpa_tl
1259     }
1260 }
1261 \cs_generate_variant:Nn
1262   \clist_set:Nn
1263   { NV }
1264 \cs_generate_variant:Nn
1265   \keys_define:nn
1266   { nV }
1267 \prg_generate_conditional_variant:Nnn
1268   \str_if_eq:nn
1269   { en }
1270   { p, F }
1271 \msg_new:nnn
1272   { markdown }
1273   { malformed-name-for-clist-option }
1274   {
1275     Clist-option-name~#1~does~not~end~with~-s.
1276   }

```

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.

```

1277 \str_if_eq:VVT
1278   \c_@@_top_layer_tl
1279   \c_@@_option_layer_plain_tex_tl
1280   {
1281     \@@_define_option_commands_and_keyvals:
1282   }
1283 \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 TeX document (further referred to as a *theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [10, 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  $\langle theme\ author\rangle/\langle theme\ purpose\rangle/\langle private\ naming\ scheme\rangle$ , where the *private naming scheme* may contain additional forward slashes. For example, a theme by a user `witiko` for the MU theme of the Beamer document class may have the name `witiko/beamer/MU`.

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

If  $\@ \langle theme\ version\rangle$  is specified after  $\langle theme\ name\rangle$ , then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If  $\@ \langle theme\ version\rangle$  is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [11].

```

1284 \ExplSyntaxOn
1285 \keys_define:nn
1286   { markdown/options }
1287   {
1288     theme .code:n = {
1289       \@_set_theme:n
1290       { #1 }
1291     },
1292     import .code:n = {
1293       \tl_set:Nn
1294       \l_tmpa_tl
1295       { #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.

```

1296     \tl_replace_all:NnV
1297     \l_tmpa_tl
1298     { / }
1299     \c_backslash_str
1300     \keys_set:nV
1301     { markdown/options/import }
1302     \l_tmpa_tl
1303   },
1304 }

```

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.

```

1305 \seq_new:N
1306   \g_@@_theme_names_seq
1307 \seq_new:N
1308   \g_@@_theme_versions_seq
1309 \tl_new:N
1310   \g_@@_current_theme_tl
1311 \tl_gset:Nn
1312   \g_@@_current_theme_tl
1313   { }
1314 \seq_gput_right:NV
1315   \g_@@_theme_names_seq
1316   \g_@@_current_theme_tl
1317 \cs_new:Npn
1318   \markdownThemeVersion
1319   { }
1320 \seq_gput_right:NV
1321   \g_@@_theme_versions_seq
1322   \g_@@_current_theme_tl
1323 \cs_new:Nn
1324   \@@_set_theme:n
1325   {

```

First, we validate the theme name.

```

1326   \str_if_in:nnF
1327     { #1 }
1328     { / }
1329     {
1330       \msg_error:nnn
1331         { markdown }
1332         { unqualified-theme-name }
1333         { #1 }
1334     }
1335   \str_if_in:nnT
1336     { #1 }
1337     { _ }
1338     {
1339       \msg_error:nnn
1340         { markdown }
1341         { underscores-in-theme-name }
1342         { #1 }
1343     }

```

Next, we extract the theme version.

```

1344   \str_if_in:nnTF
1345     { #1 }
1346     { @ }
1347     {
1348       \regex_extract_once:nnN

```

```

1349     { (.* ) @ (.* ) }
1350     { #1 }
1351     \l_tmpa_seq
1352     \seq_gpop_left:NN
1353     \l_tmpa_seq
1354     \l_tmpa_tl
1355     \seq_gpop_left:NN
1356     \l_tmpa_seq
1357     \l_tmpa_tl
1358     \tl_gset:NV
1359     \g_@@_current_theme_tl
1360     \l_tmpa_tl
1361     \seq_gpop_left:NN
1362     \l_tmpa_seq
1363     \l_tmpa_tl
1364     \cs_gset:Npe
1365     \markdownThemeVersion
1366     {
1367         \tl_use:N
1368         \l_tmpa_tl
1369     }
1370 }
1371 {
1372     \tl_gset:Nn
1373     \g_@@_current_theme_tl
1374     { #1 }
1375     \cs_gset:Npn
1376     \markdownThemeVersion
1377     { latest }
1378 }

```

Next, we munge the theme name.

```

1379     \str_set:NV
1380         \l_tmpa_str
1381         \g_@@_current_theme_tl
1382     \str_replace_all:Nnn
1383         \l_tmpa_str
1384         { / }
1385         { _ }

```

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

```

1386     \tl_set:NV
1387         \l_tmpa_tl
1388         \g_@@_current_theme_tl
1389     \tl_put_right:Nn
1390         \g_@@_current_theme_tl
1391         { / }

```



```

1392 \seq_gput_right:NV
1393 \g_@@_theme_names_seq
1394 \g_@@_current_theme_tl
1395 \seq_gput_right:NV
1396 \g_@@_theme_versions_seq
1397 \markdownThemeVersion
1398 \@@_load_theme:VeV
1399 \l_tmpa_tl
1400 { \markdownThemeVersion }
1401 \l_tmpa_str

```

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

```

1402 \seq_gpop_right:NN
1403 \g_@@_theme_names_seq
1404 \l_tmpa_tl
1405 \seq_get_right:NN
1406 \g_@@_theme_names_seq
1407 \l_tmpa_tl
1408 \tl_gset:NV
1409 \g_@@_current_theme_tl
1410 \l_tmpa_tl
1411 \seq_gpop_right:NN
1412 \g_@@_theme_versions_seq
1413 \l_tmpa_tl
1414 \seq_get_right:NN
1415 \g_@@_theme_versions_seq
1416 \l_tmpa_tl
1417 \cs_gset:Npe
1418 \markdownThemeVersion
1419 {
1420 \tl_use:N
1421 \l_tmpa_tl
1422 }
1423 }
1424 \msg_new:nnnn
1425 { markdown }
1426 { unqualified-theme-name }
1427 { Won't-load-theme-with-unqualified-name-#1 }
1428 { Theme-names-must-contain-at-least-one-forward-slash }
1429 \msg_new:nnnn
1430 { markdown }
1431 { underscores-in-theme-name }
1432 { Won't-load-theme-with-an-underscore-in-its-name-#1 }
1433 { Theme-names-must-not-contain-underscores-in-their-names }
1434 \cs_generate_variant:Nn
1435 \tl_replace_all:Nnn

```

```

1436 { NnV }
1437 \cs_generate_variant:Nn
1438 \cs_gset:Npn
1439 { Npe }

```

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

```

1440 \prop_new:N
1441 \g_@@_plain_tex_built_in_themes_prop

```

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

**witiko/diagrams** A theme that typesets fenced code blocks with the infostrings `dot`, `mermaid`, and `plantuml` as figures with diagrams produced with the command `dot` from Graphviz tools, the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`, and the command `plantuml` from the package PlantUML, respectively. The key-value attribute `caption` can be used to specify the caption of the figure. The remaining attributes are treated as image attributes.

```

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

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

  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];

```

```

    infix [label = "Infix"];
    mterms [label = "M-Terms"];
}
...

``` mermaid {caption="An example mindmap" width=9cm #mermaid}
mindmap
  root )base-idea(
    sub<br/>idea 1
      ((?))
    sub<br/>idea 2
      ((?))
    sub<br/>idea 3
      ((?))
    sub<br/>idea 4
      ((?))
  ...

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

' Diagram title
title Simple Request-Response Flow

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

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

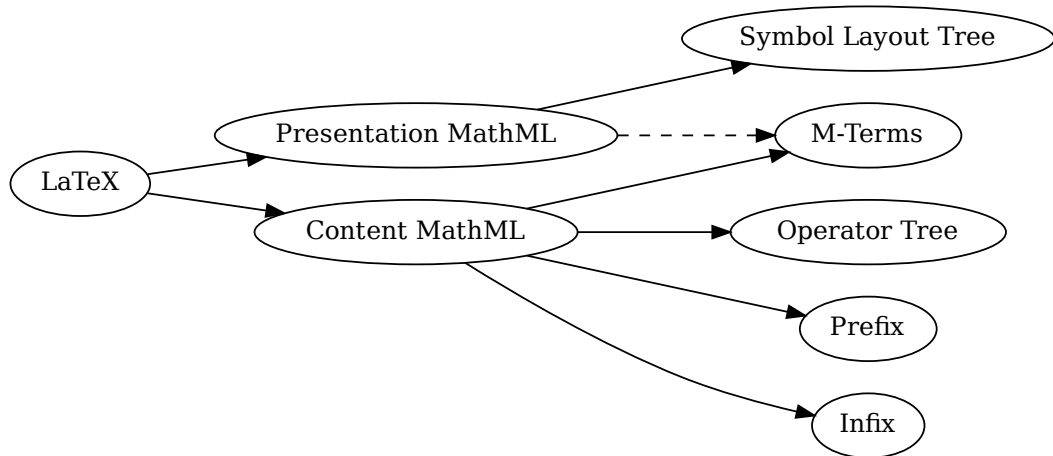
```

```

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  $\TeX$  option is enabled.

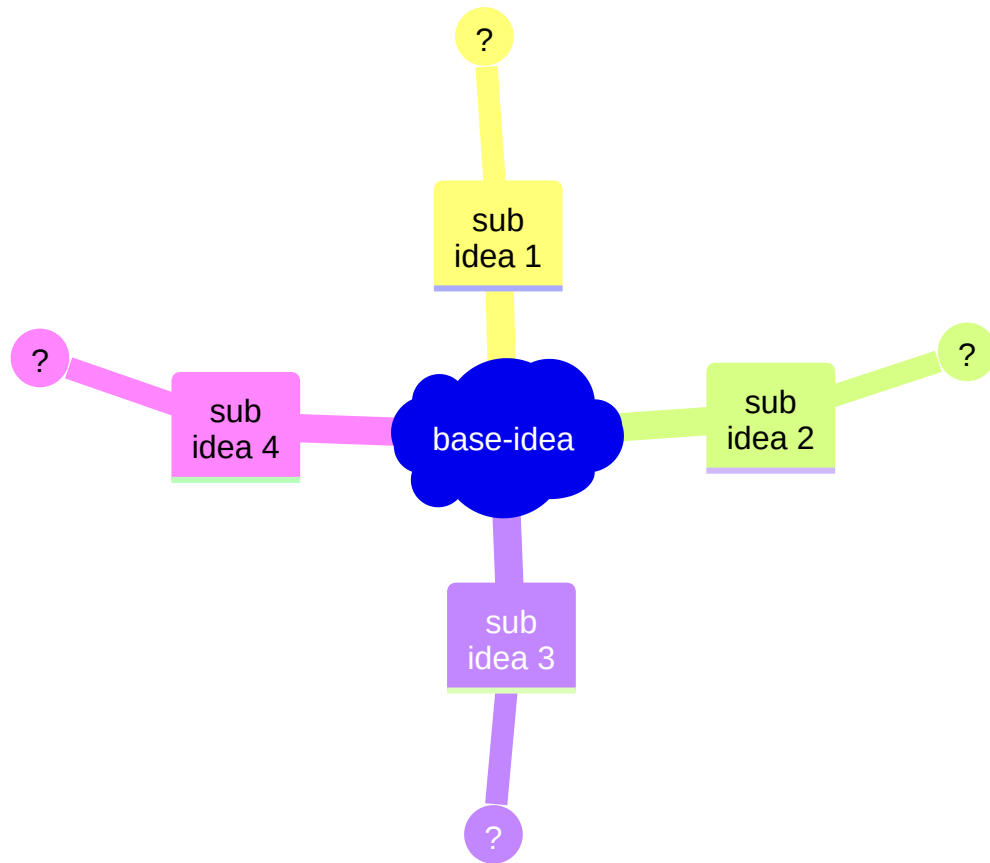
**witiko/graphicx/http** A theme that adds support for downloading images whose URL has the http or https protocol.

```

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

```

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



**Figure 5: An example mindmap**

## Simple Request-Response Flow

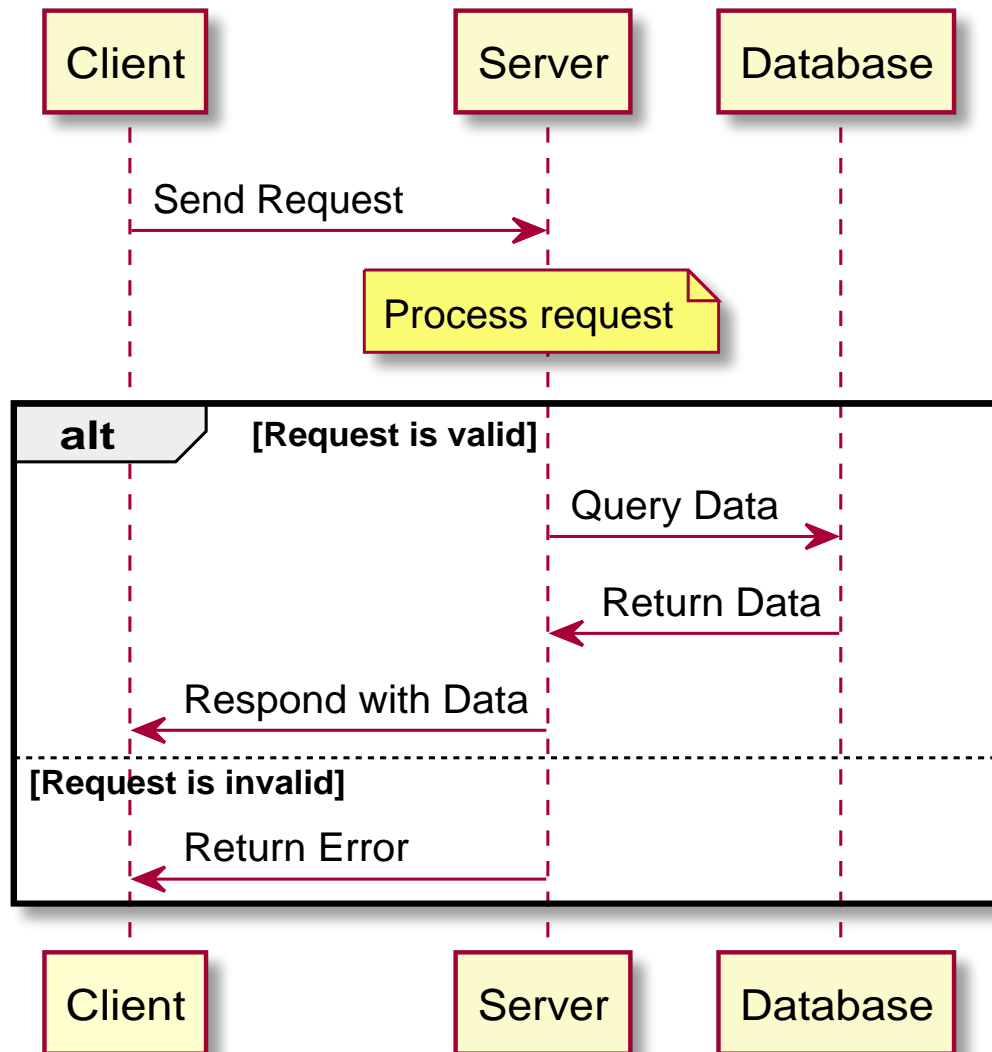


Figure 6: An example UML sequence diagram

```

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

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

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

```



## Chapter 1

# Introduction

### 1.1 Section

#### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

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

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

**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  $\TeX$  theme with the default definitions of token renderer prototypes for plain  $\TeX$ . This theme is loaded automatically together with the package and explicitly loading it has no effect.

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

## 2.2.4 Snippets

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

```
1442 \prop_new:N
1443   \g_@@_snippets_prop
1444 \cs_new:Nn
1445   \@@_setup_snippet:nn
1446   {
1447     \tl_if_empty:nT
1448       { #1 }
1449       {
1450         \msg_error:nnn
1451           { markdown }
1452           { empty-snippet-name }
1453           { #1 }
1454       }
1455     \tl_set:NV
1456       \l_tmpa_tl
1457       \g_@@_current_theme_tl
1458     \tl_put_right:Nn
1459       \l_tmpa_tl
1460       { #1 }
1461     \@@_if_snippet_exists:nT
1462       { #1 }
1463       {
1464         \msg_warning:nnV
1465           { markdown }
1466           { redefined-snippet }
1467           \l_tmpa_tl
1468       }
1469     \keys_precompile:nnN
1470       { markdown/options }
1471       { #2 }
1472       \l_tmpb_tl
1473     \prop_gput:NVV
1474       \g_@@_snippets_prop
1475       \l_tmpa_tl
1476       \l_tmpb_tl
1477   }
1478 \cs_gset_eq:NN
1479   \markdownSetupSnippet
1480   \@@_setup_snippet:nn
```



```

1481 \msg_new:nnnn
1482   { markdown }
1483   { empty-snippet-name }
1484   { Empty~snippet~name~#1 }
1485   { Pick~a~non~empty~name~for~your~snippet }
1486 \msg_new:nnn
1487   { markdown }
1488   { redefined-snippet }
1489   { Redefined~snippet~#1 }

```

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

```

1490 \tl_new:N
1491   \l_@@_current_snippet_tl
1492 \prg_new_conditional:Nnn
1493   \@@_if_snippet_exists:n
1494   { TF, T }
1495   {
1496     \tl_set:NV
1497       \l_@@_current_snippet_tl
1498       \g_@@_current_theme_tl
1499     \tl_put_right:Nn
1500       \l_@@_current_snippet_tl
1501       { #1 }
1502     \prop_if_in:NVTF
1503       \g_@@_snippets_prop
1504       \l_@@_current_snippet_tl
1505       { \prg_return_true: }
1506       { \prg_return_false: }
1507   }
1508 \cs_gset_eq:NN
1509   \markdownIfSnippetExists
1510   \@@_if_snippet_exists:nTF

```

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

```

1511 \keys_define:nn
1512   { markdown/options }
1513   {
1514     snippet .code:n = {
1515       \tl_set:NV
1516         \l_tmpa_tl
1517         \g_@@_current_theme_tl
1518       \tl_put_right:Nn
1519         \l_tmpa_tl
1520         { #1 }
1521       \@@_if_snippet_exists:nTF
1522         { #1 }
1523     }

```

```

1524     \prop_get:NVN
1525     \g_@@_snippets_prop
1526     \l_tmpa_tl
1527     \l_tmpb_tl
1528     \tl_use:N
1529     \l_tmpb_tl
1530   }
1531   {
1532     \msg_error:nnV
1533     { markdown }
1534     { undefined-snippet }
1535     \l_tmpa_tl
1536   }
1537 }
1538 }
1539 \msg_new:nnn
1540 { markdown }
1541 { undefined-snippet }
1542 { Can't-invoke-undefined-snippet~#1 }
1543 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

If the `romanNumerals` snippet were defined in the `jdoe/lists` theme, we could im-

port the `jdoo/lists` theme and use the qualified name `jdoo/lists/romanNumerals` to invoke the snippet:

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

Alternatively, we can use the extended variant of the `import` 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 = {
    jdoo/lists = romanNumerals,
  },
}
\begin{markdown}[snippet=romanNumerals]

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

```
1544 \ExplSyntaxOn
1545 \tl_new:N
1546   \l_@@_import_current_theme_tl
1547 \keys_define:nn
1548   { markdown/options/import }
1549   {
```

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

```
1550     unknown .default:n = {},
1551     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.

```
1552     \tl_set_eq:NN
```

```

1553     \l_@@_import_current_theme_tl
1554     \l_keys_key_str
1555     \tl_replace_all:NVn
1556     \l_@@_import_current_theme_tl
1557     \c_backslash_str
1558     { / }

```

Here, we import the snippets.

```

1559     \clist_map_inline:nn
1560     { #1 }
1561     {
1562         \regex_extract_once:nnNTF
1563         { ^(.*)\s+as\s+(.*)$ }
1564         { ##1 }
1565         \l_tmpa_seq
1566         {
1567             \seq_pop:NN
1568             \l_tmpa_seq
1569             \l_tmpa_tl
1570             \seq_pop:NN
1571             \l_tmpa_seq
1572             \l_tmpa_tl
1573             \seq_pop:NN
1574             \l_tmpa_seq
1575             \l_tmpb_tl
1576         }
1577         {
1578             \tl_set:Nn
1579             \l_tmpa_tl
1580             { ##1 }
1581             \tl_set:Nn
1582             \l_tmpb_tl
1583             { ##1 }
1584         }
1585         \tl_put_left:Nn
1586         \l_tmpa_tl
1587         { / }
1588         \tl_put_left:NV
1589         \l_tmpa_tl
1590         \l_@@_import_current_theme_tl
1591         \@@_setup_snippet:Vx
1592         \l_tmpb_tl
1593         { snippet = { \l_tmpa_tl } }
1594     }

```

Here, we load the theme.

```

1595     \@@_set_theme:V
1596     \l_@@_import_current_theme_tl

```

```

1597     },
1598   }
1599   \cs_generate_variant:Nn
1600     \tl_replace_all:Nnn
1601     { NVn }
1602   \cs_generate_variant:Nn
1603     \@@_set_theme:n
1604     { V }
1605   \cs_generate_variant:Nn
1606     \@@_setup_snippet:nn
1607     { 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.

```
1608 \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.

```
1609 \prop_new:N \g_@@_renderer_arities_prop
1610 \ExplSyntaxOff
```

### 2.2.5.1 Attribute Renderers

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

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

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

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

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

```

1611 \ExplSyntaxOn
1612 \cs_gset_protected:Npn
1613   \markdownRendererAttributeIdentifier
1614   {
1615     \markdownRendererAttributeIdentifierPrototype
1616   }
1617 \seq_gput_right:Nn
1618   \g_@@_renderers_seq
1619   { attributeIdentifier }
1620 \prop_gput:Nnn
1621   \g_@@_renderer_arities_prop
1622   { attributeIdentifier }
1623   { 1 }
1624 \cs_gset_protected:Npn
1625   \markdownRendererAttributeClassName
1626   {
1627     \markdownRendererAttributeClassNamePrototype
1628   }
1629 \seq_gput_right:Nn
1630   \g_@@_renderers_seq
1631   { attributeClassName }
1632 \prop_gput:Nnn
1633   \g_@@_renderer_arities_prop
1634   { attributeClassName }
1635   { 1 }
1636 \cs_gset_protected:Npn
1637   \markdownRendererAttributeKeyValue
1638   {
1639     \markdownRendererAttributeKeyValuePrototype
1640   }
1641 \seq_gput_right:Nn
1642   \g_@@_renderers_seq
1643   { attributeKeyValue }
1644 \prop_gput:Nnn
1645   \g_@@_renderer_arities_prop
1646   { attributeKeyValue }
1647   { 2 }
1648 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1649 \ExplSyntaxOn
1650 \cs_gset_protected:Npn
1651   \markdownRendererBlockQuoteBegin
1652   {
1653     \markdownRendererBlockQuoteBeginPrototype
1654   }
1655 \seq_gput_right:Nn
1656   \g_@@_renderers_seq
1657   { blockQuoteBegin }
1658 \prop_gput:Nnn
1659   \g_@@_renderer_arities_prop
1660   { blockQuoteBegin }
1661   { 0 }
1662 \ExplSyntaxOff

```

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

```

1663 \ExplSyntaxOn
1664 \cs_gset_protected:Npn
1665   \markdownRendererBlockQuoteEnd
1666   {
1667     \markdownRendererBlockQuoteEndPrototype
1668   }
1669 \seq_gput_right:Nn
1670   \g_@@_renderers_seq
1671   { blockQuoteEnd }
1672 \prop_gput:Nnn
1673   \g_@@_renderer_arities_prop
1674   { blockQuoteEnd }
1675   { 0 }
1676 \ExplSyntaxOff

```

### 2.2.5.3 Bracketed Spans Attribute Context Renderers

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

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

```

1677 \ExplSyntaxOn
1678 \cs_gset_protected:Npn
1679   \markdownRendererBracketedSpanAttributeContextBegin
1680   {
1681     \markdownRendererBracketedSpanAttributeContextBeginPrototype
1682   }
1683 \seq_gput_right:Nn

```



```

1684 \g_@@_renderers_seq
1685 { bracketedSpanAttributeContextBegin }
1686 \prop_gput:Nnn
1687 \g_@@_renderer_arities_prop
1688 { bracketedSpanAttributeContextBegin }
1689 { 0 }
1690 \cs_gset_protected:Npn
1691 \markdownRendererBracketedSpanAttributeContextEnd
1692 {
1693   \markdownRendererBracketedSpanAttributeContextEndPrototype
1694 }
1695 \seq_gput_right:Nn
1696 \g_@@_renderers_seq
1697 { bracketedSpanAttributeContextEnd }
1698 \prop_gput:Nnn
1699 \g_@@_renderer_arities_prop
1700 { bracketedSpanAttributeContextEnd }
1701 { 0 }
1702 \ExplSyntaxOff

```

#### 2.2.5.4 Bullet List Renderers

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

```

1703 \ExplSyntaxOn
1704 \cs_gset_protected:Npn
1705 \markdownRendererUlBegin
1706 {
1707   \markdownRendererUlBeginPrototype
1708 }
1709 \seq_gput_right:Nn
1710 \g_@@_renderers_seq
1711 { ulBegin }
1712 \prop_gput:Nnn
1713 \g_@@_renderer_arities_prop
1714 { ulBegin }
1715 { 0 }
1716 \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.

```

1717 \ExplSyntaxOn
1718 \cs_gset_protected:Npn

```

```

1719 \markdownRendererUlBeginTight
1720 {
1721   \markdownRendererUlBeginTightPrototype
1722 }
1723 \seq_gput_right:Nn
1724 \g_@@_renderers_seq
1725 { ulBeginTight }
1726 \prop_gput:Nnn
1727 \g_@@_renderer_arities_prop
1728 { ulBeginTight }
1729 { 0 }
1730 \ExplSyntaxOff

```

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

```

1731 \ExplSyntaxOn
1732 \cs_gset_protected:Npn
1733 \markdownRendererUlItem
1734 {
1735   \markdownRendererUlItemPrototype
1736 }
1737 \seq_gput_right:Nn
1738 \g_@@_renderers_seq
1739 { ulItem }
1740 \prop_gput:Nnn
1741 \g_@@_renderer_arities_prop
1742 { ulItem }
1743 { 0 }
1744 \ExplSyntaxOff

```

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

```

1745 \ExplSyntaxOn
1746 \cs_gset_protected:Npn
1747 \markdownRendererUlItemEnd
1748 {
1749   \markdownRendererUlItemEndPrototype
1750 }
1751 \seq_gput_right:Nn
1752 \g_@@_renderers_seq
1753 { ulItemEnd }
1754 \prop_gput:Nnn
1755 \g_@@_renderer_arities_prop
1756 { ulItemEnd }
1757 { 0 }
1758 \ExplSyntaxOff

```

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

```

1759 \ExplSyntaxOn
1760 \cs_gset_protected:Npn
1761   \markdownRendererUlEnd
1762   {
1763     \markdownRendererUlEndPrototype
1764   }
1765 \seq_gput_right:Nn
1766   \g_@@_renderers_seq
1767   { ulEnd }
1768 \prop_gput:Nnn
1769   \g_@@_renderer_arities_prop
1770   { ulEnd }
1771   { 0 }
1772 \ExplSyntaxOff

```

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

```

1773 \ExplSyntaxOn
1774 \cs_gset_protected:Npn
1775   \markdownRendererUlEndTight
1776   {
1777     \markdownRendererUlEndTightPrototype
1778   }
1779 \seq_gput_right:Nn
1780   \g_@@_renderers_seq
1781   { ulEndTight }
1782 \prop_gput:Nnn
1783   \g_@@_renderer_arities_prop
1784   { ulEndTight }
1785   { 0 }
1786 \ExplSyntaxOff

```

### 2.2.5.5 Citation Renderers

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

```

1787 \ExplSyntaxOn
1788 \cs_gset_protected:Npn
1789   \markdownRendererCite
1790   {
1791     \markdownRendererCitePrototype
1792   }
1793 \seq_gput_right:Nn
1794   \g_@@_renderers_seq
1795   { cite }
1796 \prop_gput:Nnn
1797   \g_@@_renderer_arities_prop
1798   { cite }
1799   { 1 }
1800 \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.

```

1801 \ExplSyntaxOn
1802 \cs_gset_protected:Npn
1803   \markdownRendererTextCite
1804   {
1805     \markdownRendererTextCitePrototype
1806   }
1807 \seq_gput_right:Nn
1808   \g_@@_renderers_seq
1809   { textCite }
1810 \prop_gput:Nnn
1811   \g_@@_renderer_arities_prop
1812   { textCite }
1813   { 1 }
1814 \ExplSyntaxOff

```

### 2.2.5.6 Code Block Renderers

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

```

1815 \ExplSyntaxOn
1816 \cs_gset_protected:Npn
1817   \markdownRendererInputVerbatim
1818   {
1819     \markdownRendererInputVerbatimPrototype
1820   }
1821 \seq_gput_right:Nn

```

```

1822 \g_@@_renderers_seq
1823 { inputVerbatim }
1824 \prop_gput:Nnn
1825 \g_@@_renderer_arities_prop
1826 { inputVerbatim }
1827 { 1 }
1828 \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.

```

1829 \ExplSyntaxOn
1830 \cs_gset_protected:Npn
1831 \markdownRendererInputFencedCode
1832 {
1833 \markdownRendererInputFencedCodePrototype
1834 }
1835 \seq_gput_right:Nn
1836 \g_@@_renderers_seq
1837 { inputFencedCode }
1838 \prop_gput:Nnn
1839 \g_@@_renderer_arities_prop
1840 { inputFencedCode }
1841 { 3 }
1842 \ExplSyntaxOff

```

### 2.2.5.7 Code Span Renderer

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

```

1843 \ExplSyntaxOn
1844 \cs_gset_protected:Npn
1845 \markdownRendererCodeSpan
1846 {
1847 \markdownRendererCodeSpanPrototype
1848 }
1849 \seq_gput_right:Nn
1850 \g_@@_renderers_seq
1851 { codeSpan }
1852 \prop_gput:Nnn
1853 \g_@@_renderer_arities_prop
1854 { codeSpan }
1855 { 1 }
1856 \ExplSyntaxOff

```

### 2.2.5.8 Code Span Attribute Context Renderers

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

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

```
1857 \ExplSyntaxOn
1858 \cs_gset_protected:Npn
1859   \markdownRendererCodeSpanAttributeContextBegin
1860   {
1861     \markdownRendererCodeSpanAttributeContextBeginPrototype
1862   }
1863 \seq_gput_right:Nn
1864   \g_@@_renderers_seq
1865   { codeSpanAttributeContextBegin }
1866 \prop_gput:Nnn
1867   \g_@@_renderer_arities_prop
1868   { codeSpanAttributeContextBegin }
1869   { 0 }
1870 \cs_gset_protected:Npn
1871   \markdownRendererCodeSpanAttributeContextEnd
1872   {
1873     \markdownRendererCodeSpanAttributeContextEndPrototype
1874   }
1875 \seq_gput_right:Nn
1876   \g_@@_renderers_seq
1877   { codeSpanAttributeContextEnd }
1878 \prop_gput:Nnn
1879   \g_@@_renderer_arities_prop
1880   { codeSpanAttributeContextEnd }
1881   { 0 }
1882 \ExplSyntaxOff
```

### 2.2.5.9 Content Block Renderers

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

```
1883 \ExplSyntaxOn
1884 \cs_gset_protected:Npn
1885   \markdownRendererContentBlock
1886   {
1887     \markdownRendererContentBlockPrototype
1888   }
1889 \seq_gput_right:Nn
```

```

1890 \g_@@_renderers_seq
1891 { contentBlock }
1892 \prop_gput:Nnn
1893 \g_@@_renderer_arities_prop
1894 { contentBlock }
1895 { 4 }
1896 \ExplSyntaxOff

```

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

```

1897 \ExplSyntaxOn
1898 \cs_gset_protected:Npn
1899 \markdownRendererContentBlockOnlineImage
1900 {
1901   \markdownRendererContentBlockOnlineImagePrototype
1902 }
1903 \seq_gput_right:Nn
1904 \g_@@_renderers_seq
1905 { contentBlockOnlineImage }
1906 \prop_gput:Nnn
1907 \g_@@_renderer_arities_prop
1908 { contentBlockOnlineImage }
1909 { 4 }
1910 \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>34</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s, s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local  $\text{T}\text{E}\text{X}$  directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [5] is a good starting point.

```

1911 \ExplSyntaxOn
1912 \cs_gset_protected:Npn

```

---

<sup>34</sup>Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

1913 \markdownRendererContentBlockCode
1914 {
1915   \markdownRendererContentBlockCodePrototype
1916 }
1917 \seq_gput_right:Nn
1918 \g_@@_renderers_seq
1919 { contentBlockCode }
1920 \prop_gput:Nnn
1921 \g_@@_renderer_arities_prop
1922 { contentBlockCode }
1923 { 5 }
1924 \ExplSyntaxOff

```

### 2.2.5.10 Definition List Renderers

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

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

```

1925 \ExplSyntaxOn
1926 \cs_gset_protected:Npn
1927 \markdownRendererDlBegin
1928 {
1929   \markdownRendererDlBeginPrototype
1930 }
1931 \seq_gput_right:Nn
1932 \g_@@_renderers_seq
1933 { dlBegin }
1934 \prop_gput:Nnn
1935 \g_@@_renderer_arities_prop
1936 { dlBegin }
1937 { 0 }
1938 \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.

```

1939 \ExplSyntaxOn
1940 \cs_gset_protected:Npn
1941 \markdownRendererDlBeginTight
1942 {
1943   \markdownRendererDlBeginTightPrototype
1944 }
1945 \seq_gput_right:Nn

```



```

1946 \g_@@_renderers_seq
1947 { dlBeginTight }
1948 \prop_gput:Nnn
1949 \g_@@_renderer_arities_prop
1950 { dlBeginTight }
1951 { 0 }
1952 \ExplSyntaxOff

```

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

```

1953 \ExplSyntaxOn
1954 \cs_gset_protected:Npn
1955 \markdownRendererDlItem
1956 {
1957 \markdownRendererDlItemPrototype
1958 }
1959 \seq_gput_right:Nn
1960 \g_@@_renderers_seq
1961 { dlItem }
1962 \prop_gput:Nnn
1963 \g_@@_renderer_arities_prop
1964 { dlItem }
1965 { 1 }
1966 \ExplSyntaxOff

```

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

```

1967 \ExplSyntaxOn
1968 \cs_gset_protected:Npn
1969 \markdownRendererDlItemEnd
1970 {
1971 \markdownRendererDlItemEndPrototype
1972 }
1973 \seq_gput_right:Nn
1974 \g_@@_renderers_seq
1975 { dlItemEnd }
1976 \prop_gput:Nnn
1977 \g_@@_renderer_arities_prop
1978 { dlItemEnd }
1979 { 0 }
1980 \ExplSyntaxOff

```

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

```

1981 \ExplSyntaxOn
1982 \cs_gset_protected:Npn
1983 \markdownRendererDlDefinitionBegin

```

```

1984 {
1985   \markdownRendererDlDefinitionBeginPrototype
1986 }
1987 \seq_gput_right:Nn
1988   \g_@@_renderers_seq
1989   { dlDefinitionBegin }
1990 \prop_gput:Nnn
1991   \g_@@_renderer_arities_prop
1992   { dlDefinitionBegin }
1993   { 0 }
1994 \ExplSyntaxOff

```

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

```

1995 \ExplSyntaxOn
1996 \cs_gset_protected:Npn
1997   \markdownRendererDlDefinitionEnd
1998   {
1999     \markdownRendererDlDefinitionEndPrototype
2000   }
2001 \seq_gput_right:Nn
2002   \g_@@_renderers_seq
2003   { dlDefinitionEnd }
2004 \prop_gput:Nnn
2005   \g_@@_renderer_arities_prop
2006   { dlDefinitionEnd }
2007   { 0 }
2008 \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.

```

2009 \ExplSyntaxOn
2010 \cs_gset_protected:Npn
2011   \markdownRendererDlEnd
2012   {
2013     \markdownRendererDlEndPrototype
2014   }
2015 \seq_gput_right:Nn
2016   \g_@@_renderers_seq
2017   { dlEnd }
2018 \prop_gput:Nnn
2019   \g_@@_renderer_arities_prop
2020   { dlEnd }
2021   { 0 }
2022 \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.

```

2023 \ExplSyntaxOn
2024 \cs_gset_protected:Npn
2025   \markdownRendererDlEndTight
2026   {
2027     \markdownRendererDlEndTightPrototype
2028   }
2029 \seq_gput_right:Nn
2030   \g_@@_renderers_seq
2031   { dlEndTight }
2032 \prop_gput:Nnn
2033   \g_@@_renderer_arities_prop
2034   { dlEndTight }
2035   { 0 }
2036 \ExplSyntaxOff

```

#### 2.2.5.11 Ellipsis Renderer

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

```

2037 \ExplSyntaxOn
2038 \cs_gset_protected:Npn
2039   \markdownRendererEllipsis
2040   {
2041     \markdownRendererEllipsisPrototype
2042   }
2043 \seq_gput_right:Nn
2044   \g_@@_renderers_seq
2045   { ellipsis }
2046 \prop_gput:Nnn
2047   \g_@@_renderer_arities_prop
2048   { ellipsis }
2049   { 0 }
2050 \ExplSyntaxOff

```

#### 2.2.5.12 Emphasis Renderers

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

```

2051 \ExplSyntaxOn
2052 \cs_gset_protected:Npn

```

```

2053 \markdownRendererEmphasis
2054 {
2055   \markdownRendererEmphasisPrototype
2056 }
2057 \seq_gput_right:Nn
2058 \g_@@_renderers_seq
2059 { emphasis }
2060 \prop_gput:Nnn
2061 \g_@@_renderer_arities_prop
2062 { emphasis }
2063 { 1 }
2064 \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.

```

2065 \ExplSyntaxOn
2066 \cs_gset_protected:Npn
2067   \markdownRendererStrongEmphasis
2068   {
2069     \markdownRendererStrongEmphasisPrototype
2070   }
2071 \seq_gput_right:Nn
2072 \g_@@_renderers_seq
2073 { strongEmphasis }
2074 \prop_gput:Nnn
2075 \g_@@_renderer_arities_prop
2076 { strongEmphasis }
2077 { 1 }
2078 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

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

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

```

2079 \ExplSyntaxOn
2080 \cs_gset_protected:Npn
2081   \markdownRendererFencedCodeAttributeContextBegin
2082   {
2083     \markdownRendererFencedCodeAttributeContextBeginPrototype
2084   }
2085 \seq_gput_right:Nn
2086 \g_@@_renderers_seq

```

```

2087 { fencedCodeAttributeContextBegin }
2088 \prop_gput:Nnn
2089 \g_@@_renderer_arities_prop
2090 { fencedCodeAttributeContextBegin }
2091 { 0 }
2092 \cs_gset_protected:Npn
2093 \markdownRendererFencedCodeAttributeContextEnd
2094 {
2095   \markdownRendererFencedCodeAttributeContextEndPrototype
2096 }
2097 \seq_gput_right:Nn
2098 \g_@@_renderers_seq
2099 { fencedCodeAttributeContextEnd }
2100 \prop_gput:Nnn
2101 \g_@@_renderer_arities_prop
2102 { fencedCodeAttributeContextEnd }
2103 { 0 }
2104 \ExplSyntaxOff

```

#### 2.2.5.14 Fenced Div Attribute Context Renderers

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

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

```

2105 \ExplSyntaxOn
2106 \cs_gset_protected:Npn
2107 \markdownRendererFencedDivAttributeContextBegin
2108 {
2109   \markdownRendererFencedDivAttributeContextBeginPrototype
2110 }
2111 \seq_gput_right:Nn
2112 \g_@@_renderers_seq
2113 { fencedDivAttributeContextBegin }
2114 \prop_gput:Nnn
2115 \g_@@_renderer_arities_prop
2116 { fencedDivAttributeContextBegin }
2117 { 0 }
2118 \cs_gset_protected:Npn
2119 \markdownRendererFencedDivAttributeContextEnd
2120 {
2121   \markdownRendererFencedDivAttributeContextEndPrototype
2122 }
2123 \seq_gput_right:Nn
2124 \g_@@_renderers_seq
2125 { fencedDivAttributeContextEnd }
2126 \prop_gput:Nnn

```

```

2127 \g_@@_renderer_arities_prop
2128 { fencedDivAttributeContextEnd }
2129 { 0 }
2130 \ExplSyntaxOff

```

### 2.2.5.15 Header Attribute Context Renderers

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

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

```

2131 \ExplSyntaxOn
2132 \cs_gset_protected:Npn
2133 \markdownRendererHeaderAttributeContextBegin
2134 {
2135   \markdownRendererHeaderAttributeContextBeginPrototype
2136 }
2137 \seq_gput_right:Nn
2138 \g_@@_renderers_seq
2139 { headerAttributeContextBegin }
2140 \prop_gput:Nnn
2141 \g_@@_renderer_arities_prop
2142 { headerAttributeContextBegin }
2143 { 0 }
2144 \cs_gset_protected:Npn
2145 \markdownRendererHeaderAttributeContextEnd
2146 {
2147   \markdownRendererHeaderAttributeContextEndPrototype
2148 }
2149 \seq_gput_right:Nn
2150 \g_@@_renderers_seq
2151 { headerAttributeContextEnd }
2152 \prop_gput:Nnn
2153 \g_@@_renderer_arities_prop
2154 { headerAttributeContextEnd }
2155 { 0 }
2156 \ExplSyntaxOff

```

### 2.2.5.16 Heading Renderers

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

```

2157 \ExplSyntaxOn
2158 \cs_gset_protected:Npn
2159 \markdownRendererHeadingOne

```

```

2160 {
2161   \markdownRendererHeadingOnePrototype
2162 }
2163 \seq_gput_right:Nn
2164 \g_@@_renderers_seq
2165 { headingOne }
2166 \prop_gput:Nnn
2167 \g_@@_renderer_arities_prop
2168 { headingOne }
2169 { 1 }
2170 \ExplSyntaxOff

```

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

```

2171 \ExplSyntaxOn
2172 \cs_gset_protected:Npn
2173   \markdownRendererHeadingTwo
2174   {
2175     \markdownRendererHeadingTwoPrototype
2176   }
2177 \seq_gput_right:Nn
2178 \g_@@_renderers_seq
2179 { headingTwo }
2180 \prop_gput:Nnn
2181 \g_@@_renderer_arities_prop
2182 { headingTwo }
2183 { 1 }
2184 \ExplSyntaxOff

```

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

```

2185 \ExplSyntaxOn
2186 \cs_gset_protected:Npn
2187   \markdownRendererHeadingThree
2188   {
2189     \markdownRendererHeadingThreePrototype
2190   }
2191 \seq_gput_right:Nn
2192 \g_@@_renderers_seq
2193 { headingThree }
2194 \prop_gput:Nnn
2195 \g_@@_renderer_arities_prop
2196 { headingThree }
2197 { 1 }
2198 \ExplSyntaxOff

```

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

```
2199 \ExplSyntaxOn
2200 \cs_gset_protected:Npn
2201   \markdownRendererHeadingFour
2202   {
2203     \markdownRendererHeadingFourPrototype
2204   }
2205 \seq_gput_right:Nn
2206   \g_@@_renderers_seq
2207   { headingFour }
2208 \prop_gput:Nnn
2209   \g_@@_renderer_arities_prop
2210   { headingFour }
2211   { 1 }
2212 \ExplSyntaxOff
```

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

```
2213 \ExplSyntaxOn
2214 \cs_gset_protected:Npn
2215   \markdownRendererHeadingFive
2216   {
2217     \markdownRendererHeadingFivePrototype
2218   }
2219 \seq_gput_right:Nn
2220   \g_@@_renderers_seq
2221   { headingFive }
2222 \prop_gput:Nnn
2223   \g_@@_renderer_arities_prop
2224   { headingFive }
2225   { 1 }
2226 \ExplSyntaxOff
```

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

```
2227 \ExplSyntaxOn
2228 \cs_gset_protected:Npn
2229   \markdownRendererHeadingSix
2230   {
2231     \markdownRendererHeadingSixPrototype
2232   }
2233 \seq_gput_right:Nn
2234   \g_@@_renderers_seq
2235   { headingSix }
2236 \prop_gput:Nnn
```



```

2237 \g_@@_renderer_arities_prop
2238 { headingSix }
2239 { 1 }
2240 \ExplSyntaxOff

```

### 2.2.5.17 Inline HTML Comment Renderer

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

```

2241 \ExplSyntaxOn
2242 \cs_gset_protected:Npn
2243 \markdownRendererInlineHtmlComment
2244 {
2245 \markdownRendererInlineHtmlCommentPrototype
2246 }
2247 \seq_gput_right:Nn
2248 \g_@@_renderers_seq
2249 { inlineHtmlComment }
2250 \prop_gput:Nnn
2251 \g_@@_renderer_arities_prop
2252 { inlineHtmlComment }
2253 { 1 }
2254 \ExplSyntaxOff

```

### 2.2.5.18 HTML Tag and Element Renderers

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

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

```

2255 \ExplSyntaxOn
2256 \cs_gset_protected:Npn
2257 \markdownRendererInlineHtmlTag
2258 {
2259 \markdownRendererInlineHtmlTagPrototype
2260 }
2261 \seq_gput_right:Nn
2262 \g_@@_renderers_seq
2263 { inlineHtmlTag }
2264 \prop_gput:Nnn

```

```

2265 \g_@@_renderer_arities_prop
2266 { inlineHtmlTag }
2267 { 1 }
2268 \cs_gset_protected:Npn
2269 \markdownRendererInputBlockHtmlElement
2270 {
2271   \markdownRendererInputBlockHtmlElementPrototype
2272 }
2273 \seq_gput_right:Nn
2274 \g_@@_renderers_seq
2275 { inputBlockHtmlElement }
2276 \prop_gput:Nnn
2277 \g_@@_renderer_arities_prop
2278 { inputBlockHtmlElement }
2279 { 1 }
2280 \ExplSyntaxOff

```

### 2.2.5.19 Image Renderer

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

```

2281 \ExplSyntaxOn
2282 \cs_gset_protected:Npn
2283 \markdownRendererImage
2284 {
2285   \markdownRendererImagePrototype
2286 }
2287 \seq_gput_right:Nn
2288 \g_@@_renderers_seq
2289 { image }
2290 \prop_gput:Nnn
2291 \g_@@_renderer_arities_prop
2292 { image }
2293 { 4 }
2294 \ExplSyntaxOff

```

### 2.2.5.20 Image Attribute Context Renderers

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

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

```

2295 \ExplSyntaxOn
2296 \cs_gset_protected:Npn

```

```

2297 \markdownRendererImageAttributeContextBegin
2298 {
2299   \markdownRendererImageAttributeContextBeginPrototype
2300 }
2301 \seq_gput_right:Nn
2302 \g_@@_renderers_seq
2303 { imageAttributeContextBegin }
2304 \prop_gput:Nnn
2305 \g_@@_renderer_arities_prop
2306 { imageAttributeContextBegin }
2307 { 0 }
2308 \cs_gset_protected:Npn
2309 \markdownRendererImageAttributeContextEnd
2310 {
2311   \markdownRendererImageAttributeContextEndPrototype
2312 }
2313 \seq_gput_right:Nn
2314 \g_@@_renderers_seq
2315 { imageAttributeContextEnd }
2316 \prop_gput:Nnn
2317 \g_@@_renderer_arities_prop
2318 { imageAttributeContextEnd }
2319 { 0 }
2320 \ExplSyntaxOff

```

### 2.2.5.21 Interblock Separator Renderers

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

```

2321 \ExplSyntaxOn
2322 \cs_gset_protected:Npn
2323 \markdownRendererInterblockSeparator
2324 {
2325   \markdownRendererInterblockSeparatorPrototype
2326 }
2327 \seq_gput_right:Nn
2328 \g_@@_renderers_seq
2329 { interblockSeparator }
2330 \prop_gput:Nnn
2331 \g_@@_renderer_arities_prop
2332 { interblockSeparator }
2333 { 0 }
2334 \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.

```
2335 \ExplSyntaxOn
2336 \cs_gset_protected:Npn
2337   \markdownRendererParagraphSeparator
2338   {
2339     \markdownRendererParagraphSeparatorPrototype
2340   }
2341 \seq_gput_right:Nn
2342   \g_@@_renderers_seq
2343   { paragraphSeparator }
2344 \prop_gput:Nnn
2345   \g_@@_renderer_arities_prop
2346   { paragraphSeparator }
2347   { 0 }
2348 \ExplSyntaxOff
```

#### 2.2.5.22 Line Block Renderers

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

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

```
2349 \ExplSyntaxOn
2350 \cs_gset_protected:Npn
2351   \markdownRendererLineBlockBegin
2352   {
2353     \markdownRendererLineBlockBeginPrototype
2354   }
2355 \seq_gput_right:Nn
2356   \g_@@_renderers_seq
2357   { lineBlockBegin }
2358 \prop_gput:Nnn
2359   \g_@@_renderer_arities_prop
2360   { lineBlockBegin }
2361   { 0 }
2362 \cs_gset_protected:Npn
2363   \markdownRendererLineBlockEnd
2364   {
2365     \markdownRendererLineBlockEndPrototype
2366   }
2367 \seq_gput_right:Nn
2368   \g_@@_renderers_seq
2369   { lineBlockEnd }
```

```

2370 \prop_gput:Nnn
2371   \g_@@_renderer_arities_prop
2372   { lineBlockEnd }
2373   { 0 }
2374 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2375 \ExplSyntaxOn
2376 \cs_gset_protected:Npn
2377   \markdownRendererSoftLineBreak
2378   {
2379     \markdownRendererSoftLineBreakPrototype
2380   }
2381 \seq_gput_right:Nn
2382   \g_@@_renderers_seq
2383   { softLineBreak }
2384 \prop_gput:Nnn
2385   \g_@@_renderer_arities_prop
2386   { softLineBreak }
2387   { 0 }
2388 \ExplSyntaxOff

```

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

```

2389 \ExplSyntaxOn
2390 \cs_gset_protected:Npn
2391   \markdownRendererHardLineBreak
2392   {
2393     \markdownRendererHardLineBreakPrototype
2394   }
2395 \seq_gput_right:Nn
2396   \g_@@_renderers_seq
2397   { hardLineBreak }
2398 \prop_gput:Nnn
2399   \g_@@_renderer_arities_prop
2400   { hardLineBreak }
2401   { 0 }
2402 \ExplSyntaxOff

```

### 2.2.5.24 Link Renderer

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

```

2403 \ExplSyntaxOn
2404 \cs_gset_protected:Npn
2405   \markdownRendererLink
2406   {
2407     \markdownRendererLinkPrototype
2408   }
2409 \seq_gput_right:Nn
2410   \g_@@_renderers_seq
2411   { link }
2412 \prop_gput:Nnn
2413   \g_@@_renderer_arities_prop
2414   { link }
2415   { 4 }
2416 \ExplSyntaxOff

```

### 2.2.5.25 Link Attribute Context Renderers

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

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

```

2417 \ExplSyntaxOn
2418 \cs_gset_protected:Npn
2419   \markdownRendererLinkAttributeContextBegin
2420   {
2421     \markdownRendererLinkAttributeContextBeginPrototype
2422   }
2423 \seq_gput_right:Nn
2424   \g_@@_renderers_seq
2425   { linkAttributeContextBegin }
2426 \prop_gput:Nnn
2427   \g_@@_renderer_arities_prop
2428   { linkAttributeContextBegin }
2429   { 0 }
2430 \cs_gset_protected:Npn
2431   \markdownRendererLinkAttributeContextEnd
2432   {
2433     \markdownRendererLinkAttributeContextEndPrototype
2434   }
2435 \seq_gput_right:Nn
2436   \g_@@_renderers_seq
2437   { linkAttributeContextEnd }
2438 \prop_gput:Nnn
2439   \g_@@_renderer_arities_prop
2440   { linkAttributeContextEnd }
2441   { 0 }

```

2442 \ExplSyntaxOff

### 2.2.5.26 Marked Text Renderer

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

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

```
2443 \ExplSyntaxOn
2444 \cs_gset_protected:Npn
2445   \markdownRendererMark
2446   {
2447     \markdownRendererMarkPrototype
2448   }
2449 \seq_gput_right:Nn
2450   \g_@@_renderers_seq
2451   { mark }
2452 \prop_gput:Nnn
2453   \g_@@_renderer_arities_prop
2454   { mark }
2455   { 1 }
2456 \ExplSyntaxOff
```

### 2.2.5.27 Markdown Document Renderers

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

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

```
2457 \ExplSyntaxOn
2458 \cs_gset_protected:Npn
2459   \markdownRendererDocumentBegin
2460   {
2461     \markdownRendererDocumentBeginPrototype
2462   }
2463 \seq_gput_right:Nn
2464   \g_@@_renderers_seq
2465   { documentBegin }
2466 \prop_gput:Nnn
2467   \g_@@_renderer_arities_prop
2468   { documentBegin }
2469   { 0 }
2470 \cs_gset_protected:Npn
2471   \markdownRendererDocumentEnd
```

```

2472 {
2473   \markdownRendererDocumentEndPrototype
2474 }
2475 \seq_gput_right:Nn
2476   \g_@@_renderers_seq
2477   { documentEnd }
2478 \prop_gput:Nnn
2479   \g_@@_renderer_arities_prop
2480   { documentEnd }
2481   { 0 }
2482 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2483 \ExplSyntaxOn
2484 \cs_gset_protected:Npn
2485   \markdownRendererNbsp
2486   {
2487     \markdownRendererNbspPrototype
2488   }
2489 \seq_gput_right:Nn
2490   \g_@@_renderers_seq
2491   { nbsp }
2492 \prop_gput:Nnn
2493   \g_@@_renderer_arities_prop
2494   { nbsp }
2495   { 0 }
2496 \ExplSyntaxOff

```

### 2.2.5.29 Note Renderer

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

```

2497 \def\markdownRendererNote{%
2498   \markdownRendererNotePrototype}%
2499 \ExplSyntaxOn
2500 \seq_gput_right:Nn
2501   \g_@@_renderers_seq
2502   { note }
2503 \prop_gput:Nnn
2504   \g_@@_renderer_arities_prop
2505   { note }
2506   { 1 }
2507 \ExplSyntaxOff

```



### 2.2.5.30 Ordered List Renderers

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

```
2508 \ExplSyntaxOn
2509 \cs_gset_protected:Npn
2510   \markdownRendererOlBegin
2511   {
2512     \markdownRendererOlBeginPrototype
2513   }
2514 \seq_gput_right:Nn
2515   \g_@@_renderers_seq
2516   { olBegin }
2517 \prop_gput:Nnn
2518   \g_@@_renderer_arities_prop
2519   { olBegin }
2520   { 0 }
2521 \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.

```
2522 \ExplSyntaxOn
2523 \cs_gset_protected:Npn
2524   \markdownRendererOlBeginTight
2525   {
2526     \markdownRendererOlBeginTightPrototype
2527   }
2528 \seq_gput_right:Nn
2529   \g_@@_renderers_seq
2530   { olBeginTight }
2531 \prop_gput:Nnn
2532   \g_@@_renderer_arities_prop
2533   { olBeginTight }
2534   { 0 }
2535 \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`).

```

2536 \ExplSyntaxOn
2537 \cs_gset_protected:Npn
2538   \markdownRendererFancyOlBegin
2539   {
2540     \markdownRendererFancyOlBeginPrototype
2541   }
2542 \seq_gput_right:Nn
2543   \g_@@_renderers_seq
2544   { fancyOlBegin }
2545 \prop_gput:Nnn
2546   \g_@@_renderer_arities_prop
2547   { fancyOlBegin }
2548   { 2 }
2549 \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.

```

2550 \ExplSyntaxOn
2551 \cs_gset_protected:Npn
2552   \markdownRendererFancyOlBeginTight
2553   {
2554     \markdownRendererFancyOlBeginTightPrototype
2555   }
2556 \seq_gput_right:Nn
2557   \g_@@_renderers_seq
2558   { fancyOlBeginTight }
2559 \prop_gput:Nnn
2560   \g_@@_renderer_arities_prop
2561   { fancyOlBeginTight }
2562   { 2 }
2563 \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.

```

2564 \ExplSyntaxOn
2565 \cs_gset_protected:Npn
2566   \markdownRendererOlItem
2567   {
2568     \markdownRendererOlItemPrototype
2569   }
2570 \seq_gput_right:Nn

```

```

2571 \g_@@_renderers_seq
2572 { olItem }
2573 \prop_gput:Nnn
2574 \g_@@_renderer_arities_prop
2575 { olItem }
2576 { 0 }
2577 \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.

```

2578 \ExplSyntaxOn
2579 \cs_gset_protected:Npn
2580 \markdownRendererOlItemEnd
2581 {
2582   \markdownRendererOlItemEndPrototype
2583 }
2584 \seq_gput_right:Nn
2585 \g_@@_renderers_seq
2586 { olItemEnd }
2587 \prop_gput:Nnn
2588 \g_@@_renderer_arities_prop
2589 { olItemEnd }
2590 { 0 }
2591 \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.

```

2592 \ExplSyntaxOn
2593 \cs_gset_protected:Npn
2594 \markdownRendererOlItemWithNumber
2595 {
2596   \markdownRendererOlItemWithNumberPrototype
2597 }
2598 \seq_gput_right:Nn
2599 \g_@@_renderers_seq
2600 { olItemWithNumber }
2601 \prop_gput:Nnn
2602 \g_@@_renderer_arities_prop
2603 { olItemWithNumber }
2604 { 1 }
2605 \ExplSyntaxOff

```

The `\markdownRendererFancy01Item` 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.

```

2606 \ExplSyntaxOn
2607 \cs_gset_protected:Npn
2608   \markdownRendererFancy01Item
2609   {
2610     \markdownRendererFancy01ItemPrototype
2611   }
2612 \seq_gput_right:Nn
2613   \g_@@_renderers_seq
2614   { fancy01Item }
2615 \prop_gput:Nnn
2616   \g_@@_renderer_arities_prop
2617   { fancy01Item }
2618   { 0 }
2619 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemEnd` macro represents the end of an item in a fancy ordered list. This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

2620 \ExplSyntaxOn
2621 \cs_gset_protected:Npn
2622   \markdownRendererFancy01ItemEnd
2623   {
2624     \markdownRendererFancy01ItemEndPrototype
2625   }
2626 \seq_gput_right:Nn
2627   \g_@@_renderers_seq
2628   { fancy01ItemEnd }
2629 \prop_gput:Nnn
2630   \g_@@_renderer_arities_prop
2631   { fancy01ItemEnd }
2632   { 0 }
2633 \ExplSyntaxOff

```

The `\markdownRendererFancy01ItemWithNumber` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` and `fancyLists` options are enabled. The macro receives a single numeric argument that corresponds to the item number.

```

2634 \ExplSyntaxOn
2635 \cs_gset_protected:Npn
2636   \markdownRendererFancy01ItemWithNumber
2637   {
2638     \markdownRendererFancy01ItemWithNumberPrototype
2639   }

```

```

2640 \seq_gput_right:Nn
2641   \g_@@_renderers_seq
2642   { fancyO1ItemWithNumber }
2643 \prop_gput:Nnn
2644   \g_@@_renderer_arities_prop
2645   { fancyO1ItemWithNumber }
2646   { 1 }
2647 \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.

```

2648 \ExplSyntaxOn
2649 \cs_gset_protected:Npn
2650   \markdownRendererO1End
2651   {
2652     \markdownRendererO1EndPrototype
2653   }
2654 \seq_gput_right:Nn
2655   \g_@@_renderers_seq
2656   { olEnd }
2657 \prop_gput:Nnn
2658   \g_@@_renderer_arities_prop
2659   { olEnd }
2660   { 0 }
2661 \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.

```

2662 \ExplSyntaxOn
2663 \cs_gset_protected:Npn
2664   \markdownRendererO1EndTight
2665   {
2666     \markdownRendererO1EndTightPrototype
2667   }
2668 \seq_gput_right:Nn
2669   \g_@@_renderers_seq
2670   { olEndTight }
2671 \prop_gput:Nnn
2672   \g_@@_renderer_arities_prop
2673   { olEndTight }
2674   { 0 }
2675 \ExplSyntaxOff

```

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

```

2676 \ExplSyntaxOn
2677 \cs_gset_protected:Npn
2678   \markdownRendererFancyO1End
2679   {
2680     \markdownRendererFancyO1EndPrototype
2681   }
2682 \seq_gput_right:Nn
2683   \g_@@_renderers_seq
2684   { fancyO1End }
2685 \prop_gput:Nnn
2686   \g_@@_renderer_arities_prop
2687   { fancyO1End }
2688   { 0 }
2689 \ExplSyntaxOff

```

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

```

2690 \ExplSyntaxOn
2691 \cs_gset_protected:Npn
2692   \markdownRendererFancyO1EndTight
2693   {
2694     \markdownRendererFancyO1EndTightPrototype
2695   }
2696 \seq_gput_right:Nn
2697   \g_@@_renderers_seq
2698   { fancyO1EndTight }
2699 \prop_gput:Nnn
2700   \g_@@_renderer_arities_prop
2701   { fancyO1EndTight }
2702   { 0 }
2703 \ExplSyntaxOff

```

### 2.2.5.31 Raw Content Renderers

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

```

2704 \ExplSyntaxOn

```

```

2705 \cs_gset_protected:Npn
2706   \markdownRendererInputRawInline
2707   {
2708     \markdownRendererInputRawInlinePrototype
2709   }
2710 \seq_gput_right:Nn
2711   \g_@@_renderers_seq
2712   { inputRawInline }
2713 \prop_gput:Nnn
2714   \g_@@_renderer_arities_prop
2715   { inputRawInline }
2716   { 2 }
2717 \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.

```

2718 \ExplSyntaxOn
2719 \cs_gset_protected:Npn
2720   \markdownRendererInputRawBlock
2721   {
2722     \markdownRendererInputRawBlockPrototype
2723   }
2724 \seq_gput_right:Nn
2725   \g_@@_renderers_seq
2726   { inputRawBlock }
2727 \prop_gput:Nnn
2728   \g_@@_renderer_arities_prop
2729   { inputRawBlock }
2730   { 2 }
2731 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2732 \ExplSyntaxOn
2733 \cs_gset_protected:Npn
2734   \markdownRendererSectionBegin
2735   {
2736     \markdownRendererSectionBeginPrototype
2737   }
2738 \seq_gput_right:Nn
2739   \g_@@_renderers_seq
2740   { sectionBegin }
2741 \prop_gput:Nnn

```

```

2742 \g_@@_renderer_arities_prop
2743 { sectionBegin }
2744 { 0 }
2745 \cs_gset_protected:Npn
2746 \markdownRendererSectionEnd
2747 {
2748   \markdownRendererSectionEndPrototype
2749 }
2750 \seq_gput_right:Nn
2751 \g_@@_renderers_seq
2752 { sectionEnd }
2753 \prop_gput:Nnn
2754 \g_@@_renderer_arities_prop
2755 { sectionEnd }
2756 { 0 }
2757 \ExplSyntaxOff

```

### 2.2.5.33 Replacement Character Renderers

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

```

2758 \ExplSyntaxOn
2759 \cs_gset_protected:Npn
2760 \markdownRendererReplacementCharacter
2761 {
2762   \markdownRendererReplacementCharacterPrototype
2763 }
2764 \seq_gput_right:Nn
2765 \g_@@_renderers_seq
2766 { replacementCharacter }
2767 \prop_gput:Nnn
2768 \g_@@_renderer_arities_prop
2769 { replacementCharacter }
2770 { 0 }
2771 \ExplSyntaxOff

```

### 2.2.5.34 Special Character Renderers

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

```

2772 \ExplSyntaxOn
2773 \cs_gset_protected:Npn
2774 \markdownRendererLeftBrace
2775 {
2776   \markdownRendererLeftBracePrototype
2777 }

```



```

2778 \seq_gput_right:Nn
2779   \g_@@_renderers_seq
2780   { leftBrace }
2781 \prop_gput:Nnn
2782   \g_@@_renderer_arities_prop
2783   { leftBrace }
2784   { 0 }
2785 \cs_gset_protected:Npn
2786   \markdownRendererRightBrace
2787   {
2788     \markdownRendererRightBracePrototype
2789   }
2790 \seq_gput_right:Nn
2791   \g_@@_renderers_seq
2792   { rightBrace }
2793 \prop_gput:Nnn
2794   \g_@@_renderer_arities_prop
2795   { rightBrace }
2796   { 0 }
2797 \cs_gset_protected:Npn
2798   \markdownRendererDollarSign
2799   {
2800     \markdownRendererDollarSignPrototype
2801   }
2802 \seq_gput_right:Nn
2803   \g_@@_renderers_seq
2804   { dollarSign }
2805 \prop_gput:Nnn
2806   \g_@@_renderer_arities_prop
2807   { dollarSign }
2808   { 0 }
2809 \cs_gset_protected:Npn
2810   \markdownRendererPercentSign
2811   {
2812     \markdownRendererPercentSignPrototype
2813   }
2814 \seq_gput_right:Nn
2815   \g_@@_renderers_seq
2816   { percentSign }
2817 \prop_gput:Nnn
2818   \g_@@_renderer_arities_prop
2819   { percentSign }
2820   { 0 }
2821 \cs_gset_protected:Npn
2822   \markdownRendererAmpersand
2823   {
2824     \markdownRendererAmpersandPrototype

```

```

2825 }
2826 \seq_gput_right:Nn
2827 \g_@@_renderers_seq
2828 { ampersand }
2829 \prop_gput:Nnn
2830 \g_@@_renderer_arities_prop
2831 { ampersand }
2832 { 0 }
2833 \cs_gset_protected:Npn
2834 \markdownRendererUnderscore
2835 {
2836   \markdownRendererUnderscorePrototype
2837 }
2838 \seq_gput_right:Nn
2839 \g_@@_renderers_seq
2840 { underscore }
2841 \prop_gput:Nnn
2842 \g_@@_renderer_arities_prop
2843 { underscore }
2844 { 0 }
2845 \cs_gset_protected:Npn
2846 \markdownRendererHash
2847 {
2848   \markdownRendererHashPrototype
2849 }
2850 \seq_gput_right:Nn
2851 \g_@@_renderers_seq
2852 { hash }
2853 \prop_gput:Nnn
2854 \g_@@_renderer_arities_prop
2855 { hash }
2856 { 0 }
2857 \cs_gset_protected:Npn
2858 \markdownRendererCircumflex
2859 {
2860   \markdownRendererCircumflexPrototype
2861 }
2862 \seq_gput_right:Nn
2863 \g_@@_renderers_seq
2864 { circumflex }
2865 \prop_gput:Nnn
2866 \g_@@_renderer_arities_prop
2867 { circumflex }
2868 { 0 }
2869 \cs_gset_protected:Npn
2870 \markdownRendererBackslash
2871 {

```

```

2872     \markdownRendererBackslashPrototype
2873   }
2874 \seq_gput_right:Nn
2875   \g_@@_renderers_seq
2876   { backslash }
2877 \prop_gput:Nnn
2878   \g_@@_renderer_arities_prop
2879   { backslash }
2880   { 0 }
2881 \cs_gset_protected:Npn
2882   \markdownRendererTilde
2883   {
2884     \markdownRendererTildePrototype
2885   }
2886 \seq_gput_right:Nn
2887   \g_@@_renderers_seq
2888   { tilde }
2889 \prop_gput:Nnn
2890   \g_@@_renderer_arities_prop
2891   { tilde }
2892   { 0 }
2893 \cs_gset_protected:Npn
2894   \markdownRendererPipe
2895   {
2896     \markdownRendererPipePrototype
2897   }
2898 \seq_gput_right:Nn
2899   \g_@@_renderers_seq
2900   { pipe }
2901 \prop_gput:Nnn
2902   \g_@@_renderer_arities_prop
2903   { pipe }
2904   { 0 }
2905 \ExplSyntaxOff

```

### 2.2.5.35 Strike-Through Renderer

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

```

2906 \ExplSyntaxOn
2907 \cs_gset_protected:Npn
2908   \markdownRendererStrikeThrough
2909   {
2910     \markdownRendererStrikeThroughPrototype
2911   }

```

```

2912 \seq_gput_right:Nn
2913   \g_@@_renderers_seq
2914   { strikeThrough }
2915 \prop_gput:Nnn
2916   \g_@@_renderer_arities_prop
2917   { strikeThrough }
2918   { 1 }
2919 \ExplSyntaxOff

```

### 2.2.5.36 Subscript Renderer

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

```

2920 \ExplSyntaxOn
2921 \cs_gset_protected:Npn
2922   \markdownRendererSubscript
2923   {
2924     \markdownRendererSubscriptPrototype
2925   }
2926 \seq_gput_right:Nn
2927   \g_@@_renderers_seq
2928   { subscript }
2929 \prop_gput:Nnn
2930   \g_@@_renderer_arities_prop
2931   { subscript }
2932   { 1 }

```

### 2.2.5.37 Superscript Renderer

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

```

2933 \cs_gset_protected:Npn
2934   \markdownRendererSuperscript
2935   {
2936     \markdownRendererSuperscriptPrototype
2937   }
2938 \seq_gput_right:Nn
2939   \g_@@_renderers_seq
2940   { superscript }
2941 \prop_gput:Nnn
2942   \g_@@_renderer_arities_prop
2943   { superscript }
2944   { 1 }
2945 \ExplSyntaxOff

```

### 2.2.5.38 Table Attribute Context Renderers

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

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

```
2946 \ExplSyntaxOn
2947 \cs_gset_protected:Npn
2948   \markdownRendererTableAttributeContextBegin
2949   {
2950     \markdownRendererTableAttributeContextBeginPrototype
2951   }
2952 \seq_gput_right:Nn
2953   \g_@@_renderers_seq
2954   { tableAttributeContextBegin }
2955 \prop_gput:Nnn
2956   \g_@@_renderer_arities_prop
2957   { tableAttributeContextBegin }
2958   { 0 }
2959 \cs_gset_protected:Npn
2960   \markdownRendererTableAttributeContextEnd
2961   {
2962     \markdownRendererTableAttributeContextEndPrototype
2963   }
2964 \seq_gput_right:Nn
2965   \g_@@_renderers_seq
2966   { tableAttributeContextEnd }
2967 \prop_gput:Nnn
2968   \g_@@_renderer_arities_prop
2969   { tableAttributeContextEnd }
2970   { 0 }
2971 \ExplSyntaxOff
```

### 2.2.5.39 Table Renderer

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

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

- `r` – The corresponding column is right-aligned.

```

2972 \ExplSyntaxOn
2973 \cs_gset_protected:Npn
2974   \markdownRendererTable
2975   {
2976     \markdownRendererTablePrototype
2977   }
2978 \seq_gput_right:Nn
2979   \g_@@_renderers_seq
2980   { table }
2981 \prop_gput:Nnn
2982   \g_@@_renderer_arities_prop
2983   { table }
2984   { 3 }
2985 \ExplSyntaxOff

```

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

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

```

2986 \ExplSyntaxOn
2987 \cs_gset_protected:Npn
2988   \markdownRendererInlineMath
2989   {
2990     \markdownRendererInlineMathPrototype
2991   }
2992 \seq_gput_right:Nn
2993   \g_@@_renderers_seq
2994   { inlineMath }
2995 \prop_gput:Nnn
2996   \g_@@_renderer_arities_prop
2997   { inlineMath }
2998   { 1 }
2999 \cs_gset_protected:Npn
3000   \markdownRendererDisplayMath
3001   {
3002     \markdownRendererDisplayMathPrototype
3003   }
3004 \seq_gput_right:Nn
3005   \g_@@_renderers_seq
3006   { displayMath }
3007 \prop_gput:Nnn
3008   \g_@@_renderer_arities_prop

```

```

3009 { displayMath }
3010 { 1 }
3011 \ExplSyntaxOff

```

### 2.2.5.41 Thematic Break Renderer

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

```

3012 \ExplSyntaxOn
3013 \cs_gset_protected:Npn
3014   \markdownRendererThematicBreak
3015   {
3016     \markdownRendererThematicBreakPrototype
3017   }
3018 \seq_gput_right:Nn
3019   \g_@@_renderers_seq
3020   { thematicBreak }
3021 \prop_gput:Nnn
3022   \g_@@_renderer_arities_prop
3023   { thematicBreak }
3024   { 0 }
3025 \ExplSyntaxOff

```

### 2.2.5.42 Tickbox Renderers

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

```

3026 \ExplSyntaxOn
3027 \cs_gset_protected:Npn
3028   \markdownRendererTickedBox
3029   {
3030     \markdownRendererTickedBoxPrototype
3031   }
3032 \seq_gput_right:Nn
3033   \g_@@_renderers_seq
3034   { tickedBox }
3035 \prop_gput:Nnn
3036   \g_@@_renderer_arities_prop
3037   { tickedBox }
3038   { 0 }
3039 \cs_gset_protected:Npn
3040   \markdownRendererHalfTickedBox
3041   {

```

```

3042     \markdownRendererHalfTickedBoxPrototype
3043   }
3044 \seq_gput_right:Nn
3045   \g_@@_renderers_seq
3046   { halfTickedBox }
3047 \prop_gput:Nnn
3048   \g_@@_renderer_arities_prop
3049   { halfTickedBox }
3050   { 0 }
3051 \cs_gset_protected:Npn
3052   \markdownRendererUntickedBox
3053   {
3054     \markdownRendererUntickedBoxPrototype
3055   }
3056 \seq_gput_right:Nn
3057   \g_@@_renderers_seq
3058   { untickedBox }
3059 \prop_gput:Nnn
3060   \g_@@_renderer_arities_prop
3061   { untickedBox }
3062   { 0 }
3063 \ExplSyntaxOff

```

#### 2.2.5.43 Warning and Error Renderers

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

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

```

3064 \ExplSyntaxOn
3065 \cs_gset_protected:Npn
3066   \markdownRendererWarning
3067   {
3068     \markdownRendererWarningPrototype
3069   }
3070 \cs_gset_protected:Npn
3071   \markdownRendererError
3072   {

```



```

3073     \markdownRendererErrorPrototype
3074   }
3075 \seq_gput_right:Nn
3076   \g_@@_renderers_seq
3077   { warning }
3078 \prop_gput:Nnn
3079   \g_@@_renderer_arities_prop
3080   { warning }
3081   { 4 }
3082 \seq_gput_right:Nn
3083   \g_@@_renderers_seq
3084   { error }
3085 \prop_gput:Nnn
3086   \g_@@_renderer_arities_prop
3087   { error }
3088   { 4 }
3089 \ExplSyntaxOff

```

#### 2.2.5.44 YAML Metadata Renderers

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

```

3090 \ExplSyntaxOn
3091 \cs_gset_protected:Npn
3092   \markdownRendererJekyllDataBegin
3093   {
3094     \markdownRendererJekyllDataBeginPrototype
3095   }
3096 \seq_gput_right:Nn
3097   \g_@@_renderers_seq
3098   { jekyllDataBegin }
3099 \prop_gput:Nnn
3100   \g_@@_renderer_arities_prop
3101   { jekyllDataBegin }
3102   { 0 }
3103 \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.

```

3104 \ExplSyntaxOn
3105 \cs_gset_protected:Npn
3106   \markdownRendererJekyllDataEnd
3107   {
3108     \markdownRendererJekyllDataEndPrototype
3109   }

```

```

3110 \seq_gput_right:Nn
3111   \g_@@_renderers_seq
3112   { jekyllDataEnd }
3113 \prop_gput:Nnn
3114   \g_@@_renderer_arities_prop
3115   { jekyllDataEnd }
3116   { 0 }
3117 \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.

```

3118 \ExplSyntaxOn
3119 \cs_gset_protected:Npn
3120   \markdownRendererJekyllDataMappingBegin
3121   {
3122     \markdownRendererJekyllDataMappingBeginPrototype
3123   }
3124 \seq_gput_right:Nn
3125   \g_@@_renderers_seq
3126   { jekyllDataMappingBegin }
3127 \prop_gput:Nnn
3128   \g_@@_renderer_arities_prop
3129   { jekyllDataMappingBegin }
3130   { 2 }
3131 \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.

```

3132 \ExplSyntaxOn
3133 \cs_gset_protected:Npn
3134   \markdownRendererJekyllDataMappingEnd
3135   {
3136     \markdownRendererJekyllDataMappingEndPrototype
3137   }
3138 \seq_gput_right:Nn
3139   \g_@@_renderers_seq
3140   { jekyllDataMappingEnd }
3141 \prop_gput:Nnn
3142   \g_@@_renderer_arities_prop
3143   { jekyllDataMappingEnd }
3144   { 0 }
3145 \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.

```

3146 \ExplSyntaxOn
3147 \cs_gset_protected:Npn
3148   \markdownRendererJekyllDataSequenceBegin
3149   {
3150     \markdownRendererJekyllDataSequenceBeginPrototype
3151   }
3152 \seq_gput_right:Nn
3153   \g_@@_renderers_seq
3154   { jekyllDataSequenceBegin }
3155 \prop_gput:Nnn
3156   \g_@@_renderer_arities_prop
3157   { jekyllDataSequenceBegin }
3158   { 2 }
3159 \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.

```

3160 \ExplSyntaxOn
3161 \cs_gset_protected:Npn
3162   \markdownRendererJekyllDataSequenceEnd
3163   {
3164     \markdownRendererJekyllDataSequenceEndPrototype
3165   }
3166 \seq_gput_right:Nn
3167   \g_@@_renderers_seq
3168   { jekyllDataSequenceEnd }
3169 \prop_gput:Nnn
3170   \g_@@_renderer_arities_prop
3171   { jekyllDataSequenceEnd }
3172   { 0 }
3173 \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.

```

3174 \ExplSyntaxOn
3175 \cs_gset_protected:Npn

```

```

3176 \markdownRendererJekyllDataBoolean
3177 {
3178   \markdownRendererJekyllDataBooleanPrototype
3179 }
3180 \seq_gput_right:Nn
3181 \g_@@_renderers_seq
3182 { jekyllDataBoolean }
3183 \prop_gput:Nnn
3184 \g_@@_renderer_arities_prop
3185 { jekyllDataBoolean }
3186 { 2 }
3187 \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.

```

3188 \ExplSyntaxOn
3189 \cs_gset_protected:Npn
3190 \markdownRendererJekyllDataNumber
3191 {
3192   \markdownRendererJekyllDataNumberPrototype
3193 }
3194 \seq_gput_right:Nn
3195 \g_@@_renderers_seq
3196 { jekyllDataNumber }
3197 \prop_gput:Nnn
3198 \g_@@_renderer_arities_prop
3199 { jekyllDataNumber }
3200 { 2 }
3201 \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  $\TeX$  characters in the string have been replaced by  $\TeX$  macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\TeX$ , 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.

```

3202 \ExplSyntaxOn
3203 \cs_gset_protected:Npn
3204   \markdownRendererJekyllDataTypographicString
3205   {
3206     \markdownRendererJekyllDataTypographicStringPrototype
3207   }
3208 \cs_gset_protected:Npn
3209   \markdownRendererJekyllDataProgrammaticString
3210   {
3211     \markdownRendererJekyllDataProgrammaticStringPrototype
3212   }
3213 \seq_gput_right:Nn
3214   \g_@@_renderers_seq
3215   { jekyllDataTypographicString }
3216 \prop_gput:Nnn
3217   \g_@@_renderer_arities_prop
3218   { jekyllDataTypographicString }
3219   { 2 }
3220 \seq_gput_right:Nn
3221   \g_@@_renderers_seq
3222   { jekyllDataProgrammaticString }
3223 \prop_gput:Nnn
3224   \g_@@_renderer_arities_prop
3225   { jekyllDataProgrammaticString }
3226   { 2 }
3227 \ExplSyntaxOff

```

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

```

3228 \ExplSyntaxOn
3229 \cs_gset:Npn
3230   \markdownRendererJekyllDataTypographicString
3231   {
3232     \cs_if_exist:NTF
3233       \markdownRendererJekyllDataString
3234       {
3235         \@@_if_option:nTF
3236           { experimental }
3237           {
3238             \markdownError
3239             {
3240               The~jekyllDataString~renderer~has~been~deprecated,~

```

```

3241         to-be-removed-in-Markdown-4.0.0
3242     }
3243 }
3244 {
3245     \markdownWarning
3246     {
3247         The~jekyllDataString~renderer~has~been~deprecated,~
3248         to-be-removed-in-Markdown-4.0.0
3249     }
3250     \markdownRendererJekyllDataString
3251 }
3252 }
3253 {
3254     \cs_if_exist:NTF
3255     \markdownRendererJekyllDataStringPrototype
3256     {
3257         \@@_if_option:nTF
3258         { experimental }
3259         {
3260             \markdownError
3261             {
3262                 The~jekyllDataString~renderer~prototype~
3263                 has~been~deprecated,~
3264                 to-be-removed-in-Markdown-4.0.0
3265             }
3266         }
3267         {
3268             \markdownWarning
3269             {
3270                 The~jekyllDataString~renderer~prototype~
3271                 has~been~deprecated,~
3272                 to-be-removed-in-Markdown-4.0.0
3273             }
3274             \markdownRendererJekyllDataStringPrototype
3275         }
3276     }
3277 }
3278     \markdownRendererJekyllDataTypographicStringPrototype
3279 }
3280 }
3281 }
3282 \seq_gput_right:Nn
3283   \g_@@_renderers_seq
3284   { jekyllDataString }
3285 \prop_gput:Nnn
3286   \g_@@_renderer_arities_prop
3287   { jekyllDataString }

```

```

3288 { 2 }
3289 \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.

```

3290 \ExplSyntaxOn
3291 \cs_gset_protected:Npn
3292   \markdownRendererJekyllDataEmpty
3293   {
3294     \markdownRendererJekyllDataEmptyPrototype
3295   }
3296 \seq_gput_right:Nn
3297   \g_@@_renderers_seq
3298   { jekyllDataEmpty }
3299 \prop_gput:Nnn
3300   \g_@@_renderer_arities_prop
3301   { jekyllDataEmpty }
3302   { 1 }
3303 \ExplSyntaxOff

```

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

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

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

```

3304 \ExplSyntaxOn
3305 \cs_new:Nn \@@_define_renderers:
3306   {
3307     \seq_map_inline:Nn
3308       \g_@@_renderers_seq
3309       {
3310         \@@_define_renderer:n
3311         { ##1 }
3312       }
3313   }
3314 \cs_new:Nn \@@_define_renderer:n
3315   {

```

```

3316 \@@_renderer_tl_to_csname:nN
3317   { #1 }
3318   \l_tmpa_tl
3319 \prop_get:NnN
3320   \g_@@_renderer_arities_prop
3321   { #1 }
3322   \l_tmpb_tl
3323 \@@_define_renderer:ncV
3324   { #1 }
3325   { \l_tmpa_tl }
3326   \l_tmpb_tl
3327 }
3328 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3329 {
3330   \tl_set:Nn
3331     \l_tmpa_tl
3332     { \str_uppercase:n { #1 } }
3333   \tl_set:Nx
3334     #2
3335     {
3336       markdownRenderer
3337       \tl_head:f { \l_tmpa_tl }
3338       \tl_tail:n { #1 }
3339     }
3340 }
3341 \tl_new:N
3342   \l_@@_renderer_definition_tl
3343 \bool_new:N
3344   \g_@@_appending_renderer_bool
3345 \bool_new:N
3346   \g_@@_unprotected_renderer_bool
3347 \cs_new:Nn \@@_define_renderer:nNn
3348 {
3349   \keys_define:nn
3350     { markdown/options/renderers }
3351     {
3352       #1 .code:n = {
3353         \tl_set:Nn
3354           \l_@@_renderer_definition_tl
3355           { ##1 }
3356         \regex_replace_all:nnN
3357           { \cP\#0 }
3358           { #1 }
3359         \l_@@_renderer_definition_tl
3360         \bool_if:NT
3361           \g_@@_appending_renderer_bool
3362         {

```



```

3363         \@@_tl_set_from_cs:NNn
3364         \l_tmpa_tl
3365         #2
3366         { #3 }
3367         \tl_put_left:NV
3368         \l_@@_renderer_definition_tl
3369         \l_tmpa_tl
3370     }
3371     \bool_if:NTF
3372     \g_@@_unprotected_renderer_bool
3373     {
3374         \tl_set:Nn
3375         \l_tmpa_tl
3376         { \cs_set:Npn }
3377     }
3378     {
3379         \tl_set:Nn
3380         \l_tmpa_tl
3381         { \cs_set_protected:Npn }
3382     }
3383     \exp_last_unbraced:NNV
3384     \cs_generate_from_arg_count:NNnV
3385     #2
3386     \l_tmpa_tl
3387     { #3 }
3388     \l_@@_renderer_definition_tl
3389 },
3390 }

```

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.

```

3391     \str_if_eq:nnT
3392     { #1 }
3393     { jekyllDataString }
3394     {
3395         \cs_undefine:N
3396         #2
3397     }
3398 }

```

We define the function `\@@_tl_set_from_cs:NNn` [12]. 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.

```

3399 \cs_new_protected:Nn
3400 \@@_tl_set_from_cs:NNn

```

```

3401 {
3402   \tl_set:Nn
3403     \l_tmpa_tl
3404     { #2 }
3405   \int_step_inline:nn
3406     { #3 }
3407     {
3408       \exp_args:Nnc
3409         \tl_put_right:Nn
3410         \l_tmpa_tl
3411         { @@_tl_set_from_cs_parameter_ ##1 }
3412     }
3413   \exp_args:NNV
3414     \tl_set:No
3415     \l_tmpb_tl
3416     \l_tmpa_tl
3417   \regex_replace_all:nnN
3418     { \cP. }
3419     { \0\0 }
3420     \l_tmpb_tl
3421   \int_step_inline:nn
3422     { #3 }
3423     {
3424       \regex_replace_all:nnN
3425         { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3426         { \cP\# ##1 }
3427         \l_tmpb_tl
3428     }
3429   \tl_set:NV
3430     #1
3431     \l_tmpb_tl
3432 }
3433 \cs_generate_variant:Nn
3434   \@_define_renderer:nNn
3435   { ncV }
3436 \cs_generate_variant:Nn
3437   \cs_generate_from_arg_count:NNnn
3438   { NNnV }
3439 \cs_generate_variant:Nn
3440   \tl_put_left:Nn
3441   { Nv }
3442 \keys_define:nn
3443   { markdown/options }
3444   {
3445     renderers .code:n = {
3446       \bool_gset_false:N
3447         \g_@@_unprotected_renderer_bool

```

```

3448     \keys_set:nn
3449         { markdown/options/renderers }
3450         { #1 }
3451     },
3452     unprotectedRenderers .code:n = {
3453         \bool_gset_true:N
3454         \g_@@_unprotected_renderer_bool
3455         \keys_set:nn
3456         { markdown/options/renderers }
3457         { #1 }
3458     },
3459 }

```

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.
  }
}

```

```

3460 \tl_new:N
3461   \l_@@_renderer_glob_definition_tl
3462 \seq_new:N
3463   \l_@@_renderer_glob_results_seq
3464 \regex_const:Nn
3465   \c_@@_appending_key_regex
3466   { \s*+$ }
3467 \keys_define:nn
3468   { markdown/options/renderers }
3469   {
3470     unknown .code:n = {

```

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

```

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

```

```

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

```

```

3471      % TODO: Use `\\regex_if_match` in TeX Live 2025.
3472      \\regex_match:NVTF % noqa: w202
3473      \\c_@@_appending_key_regex
3474      \\l_keys_key_str
3475      {
3476          \\bool_gset_true:N
3477          \\g_@@_appending_renderer_bool
3478          \\tl_set:NV
3479          \\l_tmpa_tl
3480          \\l_keys_key_str
3481          \\regex_replace_once:NnN
3482          \\c_@@_appending_key_regex
3483          { }
3484          \\l_tmpa_tl
3485          \\tl_set:Nx
3486          \\l_tmpb_tl
3487          { { \\l_tmpa_tl } = }
3488          \\tl_put_right:Nn
3489          \\l_tmpb_tl
3490          { { #1 } }
3491          \\keys_set:nV
3492          { markdown/options/renderers }
3493          \\l_tmpb_tl
3494          \\bool_gset_false:N
3495          \\g_@@_appending_renderer_bool
3496      }

```

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

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

Wildcards and enumerations can be combined:

```
\markdownSetup{
  renderers = {
    *1Item(|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.
  }
}
```

```
3497     {
3498         \@@_glob_seq:VnN
3499         \l_keys_key_str
3500         { g_@@_renderers_seq }
3501         \l_@@_renderer_glob_results_seq
3502     \seq_if_empty:NTF
3503         \l_@@_renderer_glob_results_seq
3504     {
3505         \msg_error:nnV
3506         { markdown }
3507         { undefined-renderer }
3508         \l_keys_key_str
3509     }
3510     {
3511         \tl_set:Nn
```

```

3512         \l_@@_renderer_glob_definition_tl
3513         { \exp_not:n { #1 } }
3514     \seq_map_inline:Nn
3515     \l_@@_renderer_glob_results_seq
3516     {
3517         \tl_set:Nn
3518         \l_tmpa_tl
3519         { { ##1 } = }
3520         \tl_put_right:Nx
3521         \l_tmpa_tl
3522         { { \l_@@_renderer_glob_definition_tl } }
3523         \keys_set:nV
3524         { markdown/options/renderers }
3525         \l_tmpa_tl
3526     }
3527 }
3528 }
3529 },
3530 }
3531 \msg_new:nnn
3532 { markdown }
3533 { undefined-renderer }
3534 {
3535     Renderer~#1~is~undefined.
3536 }
3537 \cs_generate_variant:Nn
3538 \@@_glob_seq:nnN
3539 { VnN }
3540 \cs_generate_variant:Nn
3541 \cs_generate_from_arg_count:NNnn
3542 { cNVV }
3543 \cs_generate_variant:Nn
3544 \msg_error:nnn
3545 { nnV }
3546 \prg_generate_conditional_variant:Nnn
3547 % TODO: Use `regex_if_match` in TeX Live 2025.
3548 \regex_match:Nn % noqa: w202
3549 { NV }
3550 { TF }
3551 \prop_new:N
3552 \g_@@_glob_cache_prop
3553 \tl_new:N
3554 \l_@@_current_glob_tl
3555 \cs_new:Nn
3556 \@@_glob_seq:nnN
3557 {
3558     \tl_set:Nn

```

```

3559     \l_@@_current_glob_tl
3560     { ^ #1 $ }
3561   \prop_get:NeNTF
3562     \g_@@_glob_cache_prop
3563     { #2 / \l_@@_current_glob_tl }
3564     \l_tmpa_clist
3565     {
3566       \seq_set_from_clist:NN
3567         #3
3568         \l_tmpa_clist
3569     }
3570   {
3571     \seq_clear:N
3572       #3
3573     \regex_replace_all:nnN
3574       { \* }
3575       { .* }
3576       \l_@@_current_glob_tl
3577     \regex_set:NV
3578       \l_tmpa_regex
3579       \l_@@_current_glob_tl
3580     \seq_map_inline:cn
3581       { #2 }
3582     {
3583       % TODO: Use ` \regex_if_match ` in TeX Live 2025.
3584       \regex_match:NnT % noqa: w202
3585         \l_tmpa_regex
3586         { ##1 }
3587         {
3588           \seq_put_right:Nn
3589             #3
3590             { ##1 }
3591         }
3592     }
3593     \clist_set_from_seq:NN
3594       \l_tmpa_clist
3595       #3
3596     \prop_gput:NeV
3597       \g_@@_glob_cache_prop
3598       { #2 / \l_@@_current_glob_tl }
3599     \l_tmpa_clist
3600   }
3601 }
3602 \cs_generate_variant:Nn
3603   \regex_set:Nn
3604   { NV }
3605 \cs_generate_variant:Nn

```

```

3606 \prop_gput:Nnn
3607 { NeV }

```

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

```

3608 \str_if_eq:VVT
3609 \c_@@_top_layer_tl
3610 \c_@@_option_layer_plain_tex_tl
3611 {
3612   \@@_define_renderers:
3613 }
3614 \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 [2] for the module `markdown/jekyllData`.

```

3615 \ExplSyntaxOn
3616 \keys_define:nn
3617 { markdown/jekyllData }
3618 { }
3619 \ExplSyntaxOff

```

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

```

3620 \ExplSyntaxOn
3621 \@@_with_various_cases:nn
3622 { jekyllDataRenderers }
3623 {
3624   \keys_define:nn
3625   { markdown/options }
3626   {
3627     #1 .code:n = {
3628       \tl_set:Nn
3629       \l_tmpa_tl
3630       { ##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.

```

3631   \tl_replace_all:NnV
3632   \l_tmpa_tl

```



```

3633         { / }
3634         \c_backslash_str
3635         \keys_set:nV
3636         { markdown/options/jekyll-data-renderers }
3637         \l_tmpa_tl
3638     },
3639 }
3640 }
3641 \keys_define:nn
3642 { markdown/options/jekyll-data-renderers }
3643 {
3644     unknown .code:n = {
3645         \tl_set_eq:NN
3646         \l_tmpa_tl
3647         \l_keys_key_str
3648         \tl_replace_all:NVn
3649         \l_tmpa_tl
3650         \c_backslash_str
3651         { / }
3652         \tl_put_right:Nn
3653         \l_tmpa_tl
3654         {
3655             .code:n = { #1 }
3656         }
3657         \keys_define:nV
3658         { markdown/jekyllData }
3659         \l_tmpa_tl
3660     }
3661 }
3662 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [13] 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` de-

files unprotected functions, which are easier to expand and may be preferable for programming.

```

3663 \ExplSyntaxOn
3664 \cs_new:Nn \@@_define_renderer_prototypes:
3665 {
3666   \seq_map_inline:Nn
3667     \g_@@_renderers_seq
3668     {
3669       \@@_define_renderer_prototype:n
3670       { #1 }
3671     }
3672 }
3673 \cs_new:Nn \@@_define_renderer_prototype:n
3674 {
3675   \@@_renderer_prototype_tl_to_csname:nN
3676   { #1 }
3677   \l_tmpa_tl
3678   \prop_get:NnN
3679     \g_@@_renderer_arities_prop
3680     { #1 }
3681     \l_tmpb_tl
3682   \@@_define_renderer_prototype:ncV
3683   { #1 }
3684   { \l_tmpa_tl }
3685   \l_tmpb_tl
3686 }
3687 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3688 {
3689   \tl_set:Nn
3690     \l_tmpa_tl
3691     { \str_uppercase:n { #1 } }
3692   \tl_set:Nx
3693     #2
3694     {
3695       markdownRenderer
3696       \tl_head:f { \l_tmpa_tl }
3697       \tl_tail:n { #1 }
3698       Prototype
3699     }
3700 }
3701 \tl_new:N
3702   \l_@@_renderer_prototype_definition_tl
3703 \bool_new:N
3704   \g_@@_appending_renderer_prototype_bool
3705 \bool_new:N
3706   \g_@@_unprotected_renderer_prototype_bool
3707 \cs_new:Nn \@@_define_renderer_prototype:nNn

```

```

3708 {
3709   \keys_define:nn
3710     { markdown/options/renderer-prototypes }
3711     {
3712       #1 .code:n = {
3713         \tl_set:Nn
3714           \l_@@_renderer_prototype_definition_tl
3715           { ##1 }
3716         \regex_replace_all:nnN
3717           { \cP\#0 }
3718           { #1 }
3719         \l_@@_renderer_prototype_definition_tl
3720         \bool_if:NT
3721           \g_@@_appending_renderer_prototype_bool
3722           {
3723             \@@_tl_set_from_cs:NNn
3724             \l_tmpa_tl
3725             #2
3726             { #3 }
3727             \tl_put_left:NV
3728             \l_@@_renderer_prototype_definition_tl
3729             \l_tmpa_tl
3730           }
3731         \bool_if:NTF
3732           \g_@@_unprotected_renderer_prototype_bool
3733           {
3734             \tl_set:Nn
3735             \l_tmpa_tl
3736             { \cs_set:Npn }
3737           }
3738           {
3739             \tl_set:Nn
3740             \l_tmpa_tl
3741             { \cs_set_protected:Npn }
3742           }
3743         \exp_last_unbraced:NNV
3744         \cs_generate_from_arg_count:NNnV
3745         #2
3746         \l_tmpa_tl
3747         { #3 }
3748         \l_@@_renderer_prototype_definition_tl
3749       },
3750     }

```

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.

```

3751   \str_if_eq:nnF
3752     { #1 }
3753     { jekyllDataString }
3754     {
3755       \cs_if_free:NT
3756         #2
3757         {
3758           \cs_generate_from_arg_count:NNnn
3759             #2
3760             \cs_gset_protected:Npn
3761               { #3 }
3762               { }
3763           }
3764       }
3765   }
3766 \cs_generate_variant:Nn
3767   \@_define_renderer_prototype:nNn
3768   { 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.
  }
}

```

```

3769 \keys_define:nn
3770   { markdown/options/renderer-prototypes }
3771   {
3772     unknown .code:n = {

```

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

```

\markdownSetup{
  rendererPrototypes = {
    % Start with empty renderer prototypes.
    headerAttributeContextBegin = {},
    attributeClassName = {},

```

```

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

```

```

3773 % TODO: Use `\\regex_if_match` in TeX Live 2025.
3774 \\regex_match:NVTF % noqa: w202
3775 \\c_@@_appending_key_regex
3776 \\l_keys_key_str
3777 {
3778   \\bool_gset_true:N
3779   \\g_@@_appending_renderer_prototype_bool
3780   \\tl_set:NV
3781   \\l_tmpa_tl
3782   \\l_keys_key_str
3783   \\regex_replace_once:NnN
3784   \\c_@@_appending_key_regex
3785   { }
3786   \\l_tmpa_tl
3787   \\tl_set:Nx
3788   \\l_tmpb_tl
3789   { { \\l_tmpa_tl } = }
3790   \\tl_put_right:Nn
3791   \\l_tmpb_tl
3792   { { #1 } }
3793   \\keys_set:nV
3794   { markdown/options/renderer-prototypes }
3795   \\l_tmpb_tl
3796   \\bool_gset_false:N
3797   \\g_@@_appending_renderer_prototype_bool

```

3798

}

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.
}
```

```
3799      {
3800          \@_glob_seq:VnN
3801          \l_keys_key_str
3802          { g_@_renderers_seq }
3803          \l_@_renderer_glob_results_seq
3804          \seq_if_empty:NTF
3805          \l_@_renderer_glob_results_seq
3806          {
3807              \msg_error:nnV
3808              { markdown }
3809              { undefined-renderer-prototype }
3810              \l_keys_key_str
```

```

3811     }
3812     {
3813         \tl_set:Nn
3814         \l_@@_renderer_glob_definition_tl
3815         { \exp_not:n { #1 } }
3816         \seq_map_inline:Nn
3817         \l_@@_renderer_glob_results_seq
3818         {
3819             \tl_set:Nn
3820             \l_tmpa_tl
3821             { { ##1 } = }
3822             \tl_put_right:Nx
3823             \l_tmpa_tl
3824             { { \l_@@_renderer_glob_definition_tl } }
3825             \keys_set:nV
3826             { markdown/options/renderer-prototypes }
3827             \l_tmpa_tl
3828         }
3829     }
3830 }
3831 },
3832 }
3833 \msg_new:nnn
3834 { markdown }
3835 { undefined-renderer-prototype }
3836 {
3837     Renderer~prototype~#1~is~undefined.
3838 }
3839 \@@_with_various_cases:nn
3840 { rendererPrototypes }
3841 {
3842     \keys_define:nn
3843     { markdown/options }
3844     {
3845         #1 .code:n = {
3846             \bool_gset_false:N
3847             \g_@@_unprotected_renderer_prototype_bool
3848             \keys_set:nn
3849             { markdown/options/renderer-prototypes }
3850             { ##1 }
3851         },
3852     }
3853 }
3854 \@@_with_various_cases:nn
3855 { unprotectedRendererPrototypes }
3856 {
3857     \keys_define:nn

```

```

3858     { markdown/options }
3859     {
3860     #1 .code:n = {
3861         \bool_gset_true:N
3862         \g_@@_unprotected_renderer_prototype_bool
3863         \keys_set:nn
3864         { markdown/options/renderer-prototypes }
3865         { ##1 }
3866     },
3867     }
3868 }

```

If plain  $\text{T}_{\text{E}}\text{X}$  is the top layer, we use the `\@@_define_renderer_prototypes:` macro to define plain  $\text{T}_{\text{E}}\text{X}$  token renderer prototype macros and key–values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3869 \str_if_eq:VVT
3870 \c_@@_top_layer_tl
3871 \c_@@_option_layer_plain_tex_tl
3872 {
3873     \@@_define_renderer_prototypes:
3874 }
3875 \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  $\text{T}_{\text{E}}\text{X}$  engine that does not support direct Lua access is starting to buffer a text. The plain  $\text{T}_{\text{E}}\text{X}$  implementation changes the category code of plain  $\text{T}_{\text{E}}\text{X}$  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*.

```

3876 \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  $\text{T}_{\text{E}}\text{X}$  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.

```
3877 \let\markdownReadAndConvert\relax
3878 \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*.

```
3879 \catcode`\|=0\catcode`\=12%
3880 |gdef|markdownBegin{%
3881   |markdownReadAndConvert{\markdownEnd}%
3882                               {\|markdownEnd}}%
3883 |gdef|yamlBegin{%
3884   |begingroup
3885   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3886   |markdownReadAndConvert{\yamlEnd}%
3887                               {\|yamlEnd}}%
3888 |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.

```
3889 \ExplSyntaxOn
3890 \keys_define:nn
3891   { markdown/options }
3892   {
3893     code .code:n = { #1 },
3894   }
3895 \ExplSyntaxOff
```

This can be especially useful in snippets.

## 2.3 $\LaTeX$ Interface

The  $\LaTeX$  interface provides  $\LaTeX$  environments for the typesetting of markdown input from within  $\LaTeX$ , facilities for setting Lua, plain  $\TeX$ , and  $\LaTeX$  options used during the conversion from markdown to plain  $\TeX$ , and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain  $\TeX$  interface (see Section 2.2).

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

```
3896 \ExplSyntaxOn
```

```

3897 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3898 \cs_generate_variant:Nn
3899   \tl_const:Nn
3900   { NV }
3901 \tl_if_exist:NF
3902   \c_@@_top_layer_tl
3903   {
3904     \tl_const:NV
3905       \c_@@_top_layer_tl
3906       \c_@@_option_layer_latex_tl
3907   }
3908 \ExplSyntaxOff
3909 \input markdown/markdown

```

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

```

\usepackage[⟨options⟩]{markdown}

```

where `⟨options⟩` are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3). Note that `⟨options⟩` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

#### 2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain T<sub>E</sub>X interface.

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

```

3910 \newenvironment{markdown}\relax\relax
3911 \newenvironment{markdown*}[1]\relax\relax

```

Furthermore, both environments accept L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X code showcases the usage of the `markdown` and `markdown*` environments:

```
\documentclass{article}          \documentclass{article}
\usepackage{markdown}          \usepackage{markdown}
\begin{document}              \begin{document}
\begin{markdown}[smartEllipses] \begin{markdown*}{smartEllipses}
_Hello_ **world** ...        _Hello_ **world** ...
\end{markdown}                \end{markdown*}
\end{document}                \end{document}
```

You can't directly extend the `markdown` L<sup>A</sup>T<sub>E</sub>X environment by using it in other environments as follows:

```
\newenvironment{foo}%
    {code before \begin{markdown}[some, options]}%
    {\end{markdown} code after}
```

This is because the implementation looks for the literal string `\end{markdown}` to stop scanning the markdown text. However, you can work around this limitation by using the `\markdown` and `\markdownEnd` macros directly in the definition as follows:

```
\newenvironment{foo}%
    {code before \markdown[some, options]}%
    {\markdownEnd code after}
```

Specifically, the `\markdown` macro must appear at the end of the replacement before-text and must be followed by text that has not yet been ingested by T<sub>E</sub>X's input processor.

Furthermore, using the `\markdownEnd` macro 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` L<sup>A</sup>T<sub>E</sub>X environment.

Lastly, you can't nest the other environments. For example, the following definition would be incorrect:

```
\newenvironment{bar}{\begin{foo}}{\end{foo}}
```

In this example, you should use the `\markdown` macro directly in the definition of the environment `bar`:

```
\newenvironment{bar}{\markdown[some, options]}{\markdownEnd}
```

The `yaml` L<sup>A</sup>T<sub>E</sub>X environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain T<sub>E</sub>X interface.

```
3912 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as the only optional argument.

The `yaml` environment is subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

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

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

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

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

The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown

document to plain T<sub>E</sub>X. Unlike the `\markinline` macro provided by the plain T<sub>E</sub>X interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

### 2.3.1.2 Typesetting Markdown and YAML from external documents

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

The following example 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 T<sub>E</sub>X. Unlike the `\yamlInput` 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 YAML document.

The following example L<sup>A</sup>T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X hooks with the Markdown package

L<sup>A</sup>T<sub>E</sub>X provides an intricate hook management system that allows users to insert extra material before and after certain T<sub>E</sub>X macros and L<sup>A</sup>T<sub>E</sub>X environments, among other things. [14, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before T<sub>E</sub>X commands and before/after L<sup>A</sup>T<sub>E</sub>X 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 “`markdownfoo emphasis: _bar_ baz!<markdown>`”, 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 “`markdownfoo emphasis_bar_/emphasis baz!/markdown`”, as expected.

However, the default renderer for emphasis uses currying and calls the renderer prototype in a way that prevents the use of hooks to insert extra material after the renderer, see Section 2.2.5.12. In such a case, you would need to redefine the renderer in a way that does not use currying before you would be able to use hooks to insert extra material after it.

Hooks also cannot be used to insert extra material after renderers with a variable number of parameters such as the renderer for tables, see Section 2.2.5.39.

### 2.3.3 Options

The 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.

L<sup>A</sup>T<sub>E</sub>X options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain T<sub>E</sub>X interface (see Sections 2.2.5 and 2.2.6).

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

#### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [15, Section 5.1], which supports the `finalizcache` and `frozencache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain T<sub>E</sub>X options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

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

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

```
3913 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}  
3914 \DeclareOption{frozencache}{\markdownSetup{frozenCache}}
```

### 2.3.3.2 Generating Plain T<sub>E</sub>X Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

If L<sup>A</sup>T<sub>E</sub>X 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.

```
3915 \ExplSyntaxOn
3916 \str_if_eq:VVT
3917   \c_@@_top_layer_tl
3918   \c_@@_option_layer_latex_tl
3919   {
3920     \@@_define_option_commands_and_keyvals:
3921     \@@_define_renderers:
3922     \@@_define_renderer_prototypes:
3923   }
3924 \ExplSyntaxOff
```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of plain T<sub>E</sub>X 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 L<sup>A</sup>T<sub>E</sub>X, we expand on the concept of themes by allowing a theme to be a full-blown L<sup>A</sup>T<sub>E</sub>X package. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a L<sup>A</sup>T<sub>E</sub>X package named `markdowntheme<munged theme name>.sty` 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 L<sup>A</sup>T<sub>E</sub>X-specific `.sty 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 `.sty theme file` using the `\markdownLoadPlainTeXTheme` macro.

If the L<sup>A</sup>T<sub>E</sub>X option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out



order until after the Markdown  $\LaTeX$  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}
```

```
3925 \newif\ifmarkdownLaTeXLoaded
3926   \markdownLaTeXLoadedfalse
```

Due to limitations of  $\LaTeX$ , themes may not be loaded after the beginning of a  $\LaTeX$  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.

```
3927 \ExplSyntaxOn
3928 \prop_new:N
3929   \g_@@_latex_built_in_themes_prop
3930 \ExplSyntaxOff
```

Built-in  $\LaTeX$  themes provided with the Markdown package include:

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

```
3931 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

At the end of the  $\LaTeX$  module, we load the `witiko/markdown/defaults`  $\LaTeX$  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).

```
3932 \ExplSyntaxOn
3933 \str_if_eq:VVT
3934   \c_@@_top_layer_tl
3935   \c_@@_option_layer_latex_tl
3936   {
3937     \use:c
3938       { ExplSyntaxOff }
3939     \AtEndOfPackage
3940       {
3941         \@@_if_option:nF
3942           { noDefaults }
3943         {
```

```

3944         \@@_if_option:nTF
3945             { experimental }
3946         {
3947             \@@_setup:n
3948             { theme = witiko/markdown/defaults@experimental }
3949         }
3950         {
3951             \@@_setup:n
3952             { theme = witiko/markdown/defaults }
3953         }
3954     }
3955 }
3956 \use:c
3957     { ExplSyntaxOn }
3958 }
3959 \ExplSyntaxOff

```

Please, see Section 3.3.2 for implementation details of the built-in L<sup>A</sup>T<sub>E</sub>X themes.

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

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

```

3960 \ExplSyntaxOn
3961 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3962 \cs_generate_variant:Nn
3963     \tl_const:Nn
3964     { NV }
3965 \tl_if_exist:NF
3966     \c_@@_top_layer_tl
3967     {
3968         \tl_const:NV
3969             \c_@@_top_layer_tl
3970             \c_@@_option_layer_context_tl
3971     }
3972 \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).

```

3973 \writestatus{loading}{ConTEXt User Module / markdown}%
3974 \startmodule[markdown]
3975 \def\dospecials{\do\ \do\\\do{\do\}\do\$\do\&%
3976     \do#\do\^\do\_do\%do\~}%
3977 \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 `\inputting` 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.

```
3978 \let\startmarkdown\relax
3979 \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.

```
3980 \let\startyaml\relax
3981 \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.

```
3982 \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 `TeX` interface.

```
3983 \let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts `ConTeXt` interface options (see Section 2.4.2) as its optional argument. These options will only influence this `YAML` document.

The following example `ConTeXt` code showcases the usage of the `\inputyaml` macro:

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

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

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

## 2.4.2 Options

The `ConTeXt` options are represented by a comma-delimited list of `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` (or, equivalently, `<key>=yes`) if the `=<value>` part has been omitted.

`ConTeXt` options map directly to the options recognized by the plain `TeX` interface (see Section 2.2.2).

The `ConTeXt` 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}`.

```
3984 \ExplSyntaxOn
3985 \cs_new:Npn
3986   \setupmarkdown
3987   [ #1 ]
3988   {
3989     \@@_setup:n
```

```

3990     { #1 }
3991   }

```

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

```

3992 \cs_gset_eq:NN
3993   \setupyaml
3994   \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.

```

3995 \cs_new:Nn \@@_caseless:N % noqa: w401
3996   {
3997     \regex_replace_all:nnN
3998       { ([a-z])([A-Z]) }
3999       { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
4000     #1
4001     \tl_set:Nx
4002       #1
4003       { #1 }
4004   }
4005 \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.

```

4006 \str_if_eq:VVT
4007   \c_@@_top_layer_tl
4008   \c_@@_option_layer_context_tl
4009   {
4010     \@@_define_option_commands_and_keyvals:
4011     \@@_define_renderers:
4012     \@@_define_renderer_prototypes:
4013   }

```

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

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

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

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.

```
4014 \prop_new:N
4015 \g_@@_context_built_in_themes_prop
4016 \ExplSyntaxOff
```

Built-in ConTeXt themes provided with the Markdown package include:

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

```
4017 \startmodule[markdownthemewitiko_markdown_defaults]
4018 \unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConTeXt 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 TeX *token renderers* is performed by the Lua layer. The plain TeX layer provides default definitions for the token renderers. The L<sup>A</sup>T<sub>E</sub>X and ConTeXt layers correct idiosyncrasies of the respective TeX formats, and provide format-specific default definitions for the token renderers.

## 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain 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.

```
4019 local upper, format, length =
4020     string.upper, string.format, string.len
4021 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
4022     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
4023     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.

```
4024 local early_warnings = {}
4025 local unicode_data = require("markdown-unicode-data")
4026 if metadata.version ~= unicode_data.metadata.version then
4027     table.insert(
4028         early_warnings,
4029         "markdown.lua " .. metadata.version .. " used with " ..
4030         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
4031     )
4032 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).

```
4033 local function depth_first_search(node, path, visit, leave)
4034     assert(node._type == "intermediate")
4035     visit(node, path)
4036     for label, child in pairs(node) do
```



```

4037     if label == "_type" then
4038         goto continue
4039     end
4040     if child._type == "intermediate" then
4041         depth_first_search(child, path .. label, visit, leave)
4042     else
4043         assert(child._type == "leaf")
4044         visit(child, path)
4045     end
4046     ::continue::
4047 end
4048 leave(node, path)
4049 end

```

The function `serialize_byte_parser` produces the Lua code for a PEG parser that matches a single byte.

```

4050 local function serialize_byte_parser(byte)
4051     if byte == '"' then
4052         return "P('' .. byte .. '')"
4053     elseif byte == "\\\" then
4054         return 'P("\\\\\\"')
4055     else
4056         return 'P('' .. byte .. '')'
4057     end
4058 end

```

The function `serialize_byte_range_parser` produces the Lua code for a PEG parser that matches a contiguous range of bytes.

```

4059 local function serialize_byte_range_parser(start_byte, end_byte)
4060     assert(start_byte <= end_byte)
4061     if start_byte == "\\\" then
4062         start_byte = "\\\\"
4063     end
4064     if end_byte == "\\\" then
4065         end_byte = "\\\\"
4066     end
4067     if start_byte == '"' or end_byte == '"' then
4068         return "R('' .. start_byte .. end_byte .. '')"
4069     else
4070         return 'R('' .. start_byte .. end_byte .. '')'
4071     end
4072 end

```

The function `serialize_replacement` produces the Lua code for a string literal with one or more UTF-8-encoded Unicode characters.

```

4073 local function serialize_replacement(codepoints)
4074     assert(#codepoints > 0)
4075     local buffer = {''''}

```

```

4076 for _, codepoint in ipairs(codepoints) do
4077     local code = utf8.char(codepoint)
4078     for i = 1, #code do
4079         local byte = code:sub(i, i)
4080         if byte == "'" then
4081             table.insert(buffer, '\\\'')
4082         elseif byte == "\"" then
4083             table.insert(buffer, "\\\"")
4084         else
4085             table.insert(buffer, byte)
4086         end
4087     end
4088 end
4089 table.insert(buffer, '')
4090 return table.concat(buffer)
4091 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.

```

4092 local function read_decompositions(file)
4093     return function()
4094         local from_codepoint, mapping
4095         for line in file:lines() do
4096             from_codepoint, mapping
4097                 = line:match("^(%x+);[~;]*;%a*;%d+;%a*;( [<a>%x ]*)")
4098             assert(from_codepoint ~= nil)
4099             if #mapping > 0 then
4100                 break
4101             end
4102         end
4103         if #mapping == 0 then
4104             return nil
4105         end
4106         from_codepoint = tonumber(from_codepoint, 16)
4107         local to_codepoints, decomposition_type = {}, "canonical"
4108         for raw_codepoint in mapping:gmatch("%S+") do
4109             assert(#raw_codepoint > 0)
4110             if raw_codepoint:sub(1, 1) == "<" then
4111                 decomposition_type = "compatibility"
4112             else
4113                 local codepoint = tonumber(raw_codepoint, 16)
4114                 table.insert(to_codepoints, codepoint)
4115             end
4116         end
4117         assert(#to_codepoints > 0)
4118         return decomposition_type, from_codepoint, to_codepoints

```

```

4119 end
4120 end

```

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

```

4121 print("local P, R, Cc, C = lpeg.P, lpeg.R, lpeg.Cc, lpeg.C")
4122 print("local fail = P(false)")
4123 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 [16, 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`.

```

4124 ;(function()
4125   local pathname = assert(kpse.find_file("UnicodeData.txt"),
4126     [[Could not locate file "UnicodeData.txt"]])
4127   local file = assert(io.open(pathname, "r"),
4128     [[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.

```

4129   local decomposition_types = {"canonical", "compatibility"}
4130   local prefix_trees = {}
4131   for _, decomposition_type in ipairs(decomposition_types) do
4132     prefix_trees[decomposition_type] = {}
4133     for char_length = 1, 4 do
4134       prefix_trees[decomposition_type][char_length]
4135         = {_type = "intermediate"}
4136     end
4137   end
4138   for decomposition_type, from_codepoint, to_codepoints
4139     in read_decompositions(file) do
4140     local from_code = utf8.char(from_codepoint)
4141     local node = prefix_trees[decomposition_type][#from_code]
4142     for i = 1, #from_code do
4143       local from_byte = from_code:sub(i, i)
4144       if i < #from_code then
4145         if node[from_byte] == nil then
4146           node[from_byte] = {_type = "intermediate"}
4147         end
4148         node = node[from_byte]
4149       else
4150         table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4151       end

```

```

4152     end
4153 end
4154 assert(file:close())

```

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

```

4155 print("M.decomposition_mapping = {}".format(M.decomposition_mapping))
4156 for _, decomposition_type in ipairs(decomposition_types) do
4157     print("M.decomposition_mapping." .. decomposition_type .. " = {}".format(M.decomposition_mapping[decomposition_type]))
4158     for length, prefix_tree in ipairs(prefix_trees[decomposition_type]) do
4159         do
4160             local subparsers = {}
4161             depth_first_search(prefix_tree, "", function(node, path)
4162                 if node.type == "leaf" then
4163                     local from_byte, to_codepoints = table.unpack(node)
4164                     local suffix = serialize_byte_parser(from_byte)
4165                     .. "/" .. serialize_replacement(to_codepoints)
4166                     if subparsers[path] ~= nil then
4167                         subparsers[path] = subparsers[path] .. " + " .. suffix
4168                     else
4169                         subparsers[path] = suffix
4170                     end
4171                 end
4172             end, function(_, path)
4173                 if #path > 0 then
4174                     local byte = path:sub(#path, #path)
4175                     local parent_path = path:sub(1, #path-1)
4176                     local prefix = serialize_byte_parser(byte)
4177                     local suffix
4178                     if subparsers[path]:find("%+ ") then
4179                         suffix = prefix .. " * (" .. subparsers[path] .. ")"
4180                     else
4181                         suffix = prefix .. " * " .. subparsers[parent_path]
4182                     end
4183                     if subparsers[parent_path] ~= nil then
4184                         subparsers[parent_path] = subparsers[parent_path]
4185                         .. " + " .. suffix
4186                     else
4187                         subparsers[parent_path] = suffix
4188                     end
4189                 else
4190                     print(
4191                         "M.decomposition_mapping." .. decomposition_type
4192                         .. "[" .. length .. "]" = " .. (subparsers[path] or "fail")
4193                     )
4194                 end
4195             end)
4196         end
4197     end

```

```
4198 end)()
```

### 3.1.1.2 Hangul Syllable Decomposition

Low-level parsers that decompose UTF-8-encoded Unicode characters using the Hangul syllable decomposition [16, 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`.

```
4199 ;(function()
4200   local pathname = assert(kpse.find_file("HangulSyllableType.txt"),
4201     [[Could not locate file "HangulSyllableType.txt"]])
4202   local file = assert(io.open(pathname, "r"),
4203     [[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.

```
4204   local syllable_types = {"LV", "LVT"}
4205   local prefix_trees = {}
4206   for _, syllable_type in ipairs(syllable_types) do
4207     prefix_trees[syllable_type] = {}
4208     for char_length = 1, 4 do
4209       prefix_trees[syllable_type][char_length]
4210         = {_type = "intermediate"}
4211     end
4212   end
4213   for line in file:lines() do
4214     if #line == 0 or line:sub(1, 1) == "#" then
4215       goto continue
4216     end
4217     local codepoint, syllable_type = line:match("^([%x.]+)%s*;%s*(%a*)")
4218     assert(codepoint ~= nil)
4219     if prefix_trees[syllable_type] == nil then
4220       goto continue
4221     end
4222     local codepoint_start, codepoint_end
4223     if codepoint:find("%.%.") then
4224       codepoint_start, codepoint_end
4225         = codepoint:match("^(%x+)%.%.(%x+)$")
4226     else
4227       codepoint_start, codepoint_end = codepoint, codepoint
4228     end
4229     codepoint_start = tonumber(codepoint_start, 16)
4230     codepoint_end = tonumber(codepoint_end, 16)
4231     local prev_code, prev_leaf_node
4232     for codepoint = codepoint_start, codepoint_end do
4233       local code = utf8.char(codepoint)
```

```

4234     local node = prefix_trees[syllable_type][#code]
4235     for i = 1, #code do
4236         local byte = code:sub(i, i)
4237         if i < #code then
4238             if node[byte] == nil then
4239                 node[byte] = {_type = "intermediate"}
4240             end
4241             node = node[byte]
4242         else
4243             local leaf_node
4244             if (
4245                 prev_code ~= nil
4246                 and #prev_code == #code
4247                 and code:sub(1, #code - 1)
4248                     == prev_code:sub(1, #code - 1)
4249             ) then
4250                 assert(prev_leaf_node ~= nil)
4251                 leaf_node = prev_leaf_node
4252                 leaf_node[2] = byte
4253             else
4254                 leaf_node = {byte, byte, _type = "leaf"}
4255                 table.insert(node, leaf_node)
4256             end
4257             prev_code, prev_leaf_node = code, leaf_node
4258         end
4259     end
4260     end
4261     ::continue::
4262 end
4263 assert(file:close())

```

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

```

4264     print(string.format("local s_base = %d", tonumber("AC00", 16)))
4265     print(string.format("local l_base = %d", tonumber("1100", 16)))
4266     print(string.format("local v_base = %d", tonumber("1161", 16)))
4267     print(string.format("local t_base = %d", tonumber("11A7", 16)))
4268     print(string.format("local l_count = %d", 19))
4269     print(string.format("local v_count = %d", 21))
4270     print(string.format("local t_count = %d", 28))
4271     print("local n_count = v_count * t_count")
4272     print("local s_count = l_count * n_count")
4273
4274     print("local function hangul_decompose_LV(byte)")
4275     print("  local s = utf8.codepoint(byte)")
4276     print("  local s_index = s - s_base")
4277     print("  local l_index = s_index // n_count")

```

```

4278 print(" local v_index = (s_index % n_count) // t_count")
4279 print(" local l_part = l_base + l_index")
4280 print(" local v_part = v_base + v_index")
4281 print(" return utf8.char(l_part) .. utf8.char(v_part)")
4282 print("end")
4283
4284 print("local function hangul_decompose_LVT(byte)")
4285 print(" local s = utf8.codepoint(byte)")
4286 print(" local s_index = s - s_base")
4287 print(" local lv_index = (s_index // t_count) * t_count")
4288 print(" local t_index = s_index % t_count")
4289 print(" local lv_part = s_base + lv_index")
4290 print(" local t_part = t_base + t_index")
4291 print(" return utf8.char(lv_part) .. utf8.char(t_part)")
4292 print("end")
4293
4294 print("function M.hangul_compose(first_byte, second_byte)")
4295 print(" local last = utf8.codepoint(first_byte)")
4296 print(" local ch = utf8.codepoint(second_byte)")
4297 print(" local l_index = last - l_base")
4298 print(" if 0 <= l_index and l_index < l_count then")
4299 print("   local v_index = ch - v_base")
4300 print("   if 0 <= v_index and v_index < v_count then")
4301 print("     return utf8.char("
4302         s_base + (l_index * v_count + v_index) * t_count")
4303     ")")
4304 print("   end")
4305 print(" end")
4306 print(" local s_index = last - s_base")
4307 print(" if 0 <= s_index and s_index < s_count")
4308 print("   and s_index % t_count == 0 then")
4309 print("   local t_index = ch - t_base")
4310 print("   if 0 < t_index and t_index < t_count then")
4311 print("     return utf8.char(last + t_index)")
4312 print("   end")
4313 print(" end")
4314 print(" return nil")
4315 print("end")

```

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

```

4316 print("M.decomposition_mapping.hangul = {}")
4317 for _, syllable_type in ipairs(syllable_types) do
4318   print("M.decomposition_mapping.hangul." .. syllable_type .. " = {}")
4319   for length, prefix_tree in pairs(prefix_trees[syllable_type]) do
4320     local subparsers = {}
4321     depth_first_search(prefix_tree, "", function(node, path)
4322       if node._type == "leaf" then
4323         local start_byte, end_byte = table.unpack(node)

```

```

4324     local suffix
4325     if start_byte == end_byte then
4326         suffix = serialize_byte_parser(start_byte)
4327     else
4328         assert(start_byte < end_byte)
4329         suffix = serialize_byte_range_parser(start_byte, end_byte)
4330     end
4331     if subparsers[path] ~= nil then
4332         subparsers[path] = subparsers[path] .. " + " .. suffix
4333     else
4334         subparsers[path] = suffix
4335     end
4336 end
4337 end, function(_, path)
4338     if #path > 0 then
4339         local byte = path:sub(#path, #path)
4340         local parent_path = path:sub(1, #path-1)
4341         local prefix = serialize_byte_parser(byte)
4342         local suffix
4343         if subparsers[path]:find(" %+ ") then
4344             suffix = prefix .. " * (" .. subparsers[path] .. ")"
4345         else
4346             suffix = prefix .. " * " .. subparsers[path]
4347         end
4348         if subparsers[parent_path] ~= nil then
4349             subparsers[parent_path] = subparsers[parent_path]
4350                 .. " + " .. suffix
4351         else
4352             subparsers[parent_path] = suffix
4353         end
4354     else
4355         if subparsers[path] then
4356             print(
4357                 "M.decomposition_mapping.hangul." .. syllable_type
4358                 .. "[" .. length .. "] = C(" .. subparsers[path] .. ")"
4359                 .. " / hangul_decompose_" .. syllable_type
4360             )
4361         else
4362             print(
4363                 "M.decomposition_mapping.hangul." .. syllable_type
4364                 .. "[" .. length .. "] = fail"
4365             )
4366         end
4367     end
4368 end)
4369 end
4370 end

```



```
4371 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 [16, 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.

```
4372 ;(function()
4373   local pathname
4374   = assert(kpse.find_file("DerivedNormalizationProps.txt"),
4375     [[Could not locate file "DerivedNormalizationProps.txt"]])
4376   local file = assert(io.open(pathname, "r"),
4377     [[Could not open file "DerivedNormalizationProps.txt"]])
4378   local full_composition_exclusions = {}
4379   for line in file:lines() do
4380     if #line == 0 or line:sub(1, 1) == "#" then
4381       goto continue
4382     end
4383     local codepoint, property = line:match("^([%x.]+)%s*;%s*([%a_]+)")
4384     assert(codepoint ~= nil)
4385     if property ~= "Full_Composition_Exclusion" then
4386       goto continue
4387     end
4388     local codepoint_start, codepoint_end
4389     if codepoint:find("%.%.") then
4390       codepoint_start, codepoint_end
4391       = codepoint:match("^(%x+)%.%.(%x+)$")
4392     else
4393       codepoint_start, codepoint_end = codepoint, codepoint
4394     end
4395     codepoint_start = tonumber(codepoint_start, 16)
4396     codepoint_end = tonumber(codepoint_end, 16)
4397     for codepoint = codepoint_start, codepoint_end do
4398       full_composition_exclusions[codepoint] = true
4399     end
4400     ::continue::
4401   end
4402   assert(file:close())
```

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

```
4403   pathname = assert(kpse.find_file("UnicodeData.txt"),
4404     [[Could not locate file "UnicodeData.txt"]])
4405   file = assert(io.open(pathname, "r"),
4406     [[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.

```
4407 local prefix_trees = {starters = {}, both = {}}
4408 for first_char_length = 1, 4 do
4409   prefix_trees.starters[first_char_length]
4410     = {_type = "intermediate"}
4411   prefix_trees.both[first_char_length] = {}
4412   for second_char_length = 1, 4 do
4413     prefix_trees.both[first_char_length][second_char_length]
4414       = {_type = "intermediate"}
4415   end
4416 end
4417 local seen_starter_codes = {}
4418 for decomposition_type, to_codepoint, from_codepoints
4419   in read_decompositions(file) do
4420   if (
4421     decomposition_type ~= "canonical"
4422     or #from_codepoints ~= 2
4423     or full_composition_exclusions[to_codepoint]
4424   ) then
4425     goto continue
4426   end
4427   local starter_code = utf8.char(from_codepoints[1])
4428   local combining_character_code = utf8.char(from_codepoints[2])
4429   local starter_node = prefix_trees.starters[#starter_code]
4430   local both_node
4431     = prefix_trees.both[#starter_code][#combining_character_code]
4432   for i = 1, #starter_code do
4433     local from_byte = starter_code:sub(i, i)
4434     if both_node[from_byte] == nil then
4435       both_node[from_byte] = {_type = "intermediate"}
4436     end
4437     both_node = both_node[from_byte]
4438     if i < #starter_code then
4439       if starter_node[from_byte] == nil then
4440         starter_node[from_byte] = {_type = "intermediate"}
4441       end
4442       starter_node = starter_node[from_byte]
4443     elseif seen_starter_codes[starter_code] == nil then
4444       seen_starter_codes[starter_code] = true
4445       table.insert(starter_node, {from_byte, _type = "leaf"})
4446     end
4447   end
4448   for i = 1, #combining_character_code do
4449     local from_byte = combining_character_code:sub(i, i)
4450     if i < #combining_character_code then
4451       if both_node[from_byte] == nil then
```

```

4452         both_node[from_byte] = {_type = "intermediate"}
4453     end
4454     both_node = both_node[from_byte]
4455 else
4456     table.insert(
4457         both_node,
4458         {from_byte, to_codepoint, _type = "leaf"}
4459     )
4460 end
4461 end
4462 ::continue::
4463 end
4464 assert(file:close())

```

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

```

4465 print("M.composition_mapping = {starters = {}, both = {}}")
4466 for first_char_length = 1, 4 do
4467     local prefix_tree = prefix_trees.starters[first_char_length]
4468     local subparsers = {}
4469     depth_first_search(prefix_tree, "", function(node, path)
4470         if node._type == "leaf" then
4471             local from_byte = table.unpack(node)
4472             local suffix = serialize_byte_parser(from_byte)
4473             if subparsers[path] ~= nil then
4474                 subparsers[path] = subparsers[path] .. " + " .. suffix
4475             else
4476                 subparsers[path] = suffix
4477             end
4478         end
4479     end, function(_, path)
4480         if #path > 0 then
4481             local byte = path:sub(#path, #path)
4482             local parent_path = path:sub(1, #path-1)
4483             local prefix = serialize_byte_parser(byte)
4484             local suffix
4485             if subparsers[path]:find(" %+ ") then
4486                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4487             else
4488                 suffix = prefix .. " * " .. subparsers[path]
4489             end
4490             if subparsers[parent_path] ~= nil then
4491                 subparsers[parent_path] = subparsers[parent_path]
4492                     .. " + " .. suffix
4493             else
4494                 subparsers[parent_path] = suffix
4495             end
4496         else
4497             print(

```

```

4498         "M.composition_mapping.starters["
4499         .. first_char_length .. "] = " .. (subparsers[path] or "fail")
4500     )
4501     end
4502 end)
4503 print(
4504     string.format(
4505         "M.composition_mapping.both[%d] = {}",
4506         first_char_length
4507     )
4508 )
4509 for second_char_length = 1, 4 do
4510     prefix_tree
4511     = prefix_trees.both[first_char_length][second_char_length]
4512     subparsers = {}
4513     depth_first_search(prefix_tree, "", function(node, path)
4514         if node._type == "leaf" then
4515             local from_byte, to_codepoint = table.unpack(node)
4516             local suffix = serialize_byte_parser(from_byte)
4517             .. " / " .. serialize_replacement({to_codepoint})
4518             if subparsers[path] ~= nil then
4519                 subparsers[path] = subparsers[path] .. " + " .. suffix
4520             else
4521                 subparsers[path] = suffix
4522             end
4523         end
4524     end, function(_, path)
4525         if #path > 0 then
4526             local byte = path:sub(#path, #path)
4527             local parent_path = path:sub(1, #path-1)
4528             local prefix = serialize_byte_parser(byte)
4529             local suffix
4530             if subparsers[path]:find(" %+ ") then
4531                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4532             else
4533                 suffix = prefix .. " * " .. subparsers[path]
4534             end
4535             if subparsers[parent_path] ~= nil then
4536                 subparsers[parent_path] = subparsers[parent_path]
4537                 .. " + " .. suffix
4538             else
4539                 subparsers[parent_path] = suffix
4540             end
4541         else
4542             print(
4543                 "M.composition_mapping.both[" .. first_char_length .. "]["
4544                 .. second_char_length .. "] = "

```

```

4545         .. (subparsers[path] or "fail")
4546     )
4547     end
4548 end)
4549 end
4550 end
4551 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) [16, 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`.

```

4552 ;(function()
4553   local pathname = assert(kpse.find_file("CaseFolding.txt"),
4554     [[Could not locate file "CaseFolding.txt"]])
4555   local file = assert(io.open(pathname, "r"),
4556     [[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.

```

4557   local prefix_trees = {}
4558   for char_length = 1, 4 do
4559     prefix_trees[char_length] = {_type = "intermediate"}
4560   end
4561   for line in file:lines() do
4562     if #line == 0 or line:sub(1, 1) == "#" then
4563       goto continue
4564     end
4565     local raw_from_codepoint, status, raw_to_codepoints
4566       = line:match("^(%x+); ([CFST]); ([%x ]+);")
4567     assert(raw_from_codepoint ~= nil)
4568     assert(status ~= nil)
4569     assert(raw_to_codepoints ~= nil)
4570     if status ~= "C" and status ~= "F" then
4571       goto continue
4572     end
4573     local from_codepoint = tonumber(raw_from_codepoint, 16)
4574     local to_codepoints = {}
4575     for raw_codepoint in raw_to_codepoints:gmatch('%x+') do
4576       local codepoint = tonumber(raw_codepoint, 16)
4577       table.insert(to_codepoints, codepoint)
4578     end
4579     local from_code = utf8.char(from_codepoint)
4580     local node = prefix_trees[#from_code]

```

```

4581     for i = 1, #from_code do
4582         local from_byte = from_code:sub(i, i)
4583         if i < #from_code then
4584             if node[from_byte] == nil then
4585                 node[from_byte] = {_type = "intermediate"}
4586             end
4587             node = node[from_byte]
4588         else
4589             table.insert(node, {from_byte, to_codepoints, _type = "leaf"})
4590         end
4591     end
4592     ::continue::
4593 end
4594 assert(file:close())

```

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

```

4595 print("M.casefold_mapping = {}")
4596 for length, prefix_tree in pairs(prefix_trees) do
4597     local subparsers = {}
4598     depth_first_search(prefix_tree, "", function(node, path)
4599         if node._type == "leaf" then
4600             local from_byte, to_codepoints = table.unpack(node)
4601             local suffix = serialize_byte_parser(from_byte)
4602             .. " / " .. serialize_replacement(to_codepoints)
4603             if subparsers[path] ~= nil then
4604                 subparsers[path] = subparsers[path] .. " + " .. suffix
4605             else
4606                 subparsers[path] = suffix
4607             end
4608         end
4609     end, function(_, path)
4610         if #path > 0 then
4611             local byte = path:sub(#path, #path)
4612             local parent_path = path:sub(1, #path-1)
4613             local prefix = serialize_byte_parser(byte)
4614             local suffix
4615             if subparsers[path]:find(" %+ ") then
4616                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4617             else
4618                 suffix = prefix .. " * " .. subparsers[path]
4619             end
4620             if subparsers[parent_path] ~= nil then
4621                 subparsers[parent_path] = subparsers[parent_path]
4622                 .. " + " .. suffix
4623             else
4624                 subparsers[parent_path] = suffix
4625             end
4626         else

```

```

4627     print(
4628         "M.casefold_mapping[" .. length .. "] = "
4629         .. (subparsers[path] or "fail")
4630     )
4631     end
4632 end)
4633 end
4634 end)()

```

### 3.1.1.5 Character Categories

Low-level parsers of UTF-8-encoded Unicode characters from different general categories [16, 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`.

```

4635 ;(function()
4636     local pathname = assert(kpse.find_file("UnicodeData.txt"),
4637         [[Could not locate file "UnicodeData.txt"]])
4638     local file = assert(io.open(pathname, "r"),
4639         [[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.

```

4640     local categories = {"L", "N", "P", "Pc", "S", "Z"}
4641     local prefix_trees = {}
4642     for _, category in ipairs(categories) do
4643         prefix_trees[category] = {}
4644         for char_length = 1, 4 do
4645             prefix_trees[category][char_length] = {_type = "intermediate"}
4646         end
4647     end
4648     for line in file:lines() do
4649         local codepoint, full_category = line:match("^(%x+);[~;]*;(%a*)")
4650         assert(#full_category >= 1)
4651         local major_category = full_category:sub(1, 1)
4652         for _, category in ipairs({full_category, major_category}) do
4653             if prefix_trees[category] == nil then
4654                 goto continue
4655             end
4656             local code = utf8.char(tonumber(codepoint, 16))
4657             local node = prefix_trees[category][#code]
4658             for i = 1, #code do
4659                 local byte = code:sub(i, i)
4660                 if i < #code then
4661                     if node[byte] == nil then
4662                         node[byte] = {_type = "intermediate"}

```

```

4663         end
4664         node = node[byte]
4665     else
4666         table.insert(node, {byte, _type = "leaf"})
4667     end
4668 end
4669 ::continue::
4670 end
4671 end
4672 assert(file:close())

```

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

```

4673 print("M.categories = {}")
4674 for _, category in ipairs(categories) do
4675     print("M.categories." .. category .. " = {}")
4676     for length, prefix_tree in pairs(prefix_trees[category]) do
4677         local subparsers = {}
4678         depth_first_search(prefix_tree, "", function(node, path)
4679             if node._type == "leaf" then
4680                 local byte = node[1]
4681                 local suffix = serialize_byte_parser(byte)
4682                 if subparsers[path] ~= nil then
4683                     subparsers[path] = subparsers[path] .. " + " .. suffix
4684                 else
4685                     subparsers[path] = suffix
4686                 end
4687             end
4688         end, function(_, path)
4689             if #path > 0 then
4690                 local byte = path:sub(#path, #path)
4691                 local parent_path = path:sub(1, #path-1)
4692                 local prefix = serialize_byte_parser(byte)
4693                 local suffix
4694                 if subparsers[path]:find(" %+ ") then
4695                     suffix = prefix .. " * (" .. subparsers[path] .. ")"
4696                 else
4697                     suffix = prefix .. " * " .. subparsers[path]
4698                 end
4699                 if subparsers[parent_path] ~= nil then
4700                     subparsers[parent_path] = subparsers[parent_path]
4701                         .. " + " .. suffix
4702                 else
4703                     subparsers[parent_path] = suffix
4704                 end
4705             else
4706                 print(
4707                     "M.categories." .. category .. "[" .. length .. "] = "
4708                     .. (subparsers[path] or "fail")

```



```

4709         )
4710     end
4711 end)
4712 end
4713 end
4714 end)()

```

### 3.1.1.6 Canonical Ordering Classes

Low-level parsers of UTF-8-encoded Unicode characters from different character classes [16, 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`.

```

4715 ;(function()
4716     local pathname = assert(kpse.find_file("UnicodeData.txt"),
4717         [[Could not locate file "UnicodeData.txt"]])
4718     local file = assert(io.open(pathname, "r"),
4719         [[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.

```

4720     local prefix_trees = {}
4721     for char_length = 1, 4 do
4722         prefix_trees[char_length] = {_type = "intermediate"}
4723     end
4724     for line in file:lines() do
4725         local codepoint, combining_class
4726             = line:match("^(%x+);[~;]*;%a*(%d+)")
4727         combining_class = tonumber(combining_class)
4728         if combining_class == 0 then
4729             goto continue
4730         end
4731         local code = utf8.char(tonumber(codepoint, 16))
4732         local node = prefix_trees[#code]
4733         for i = 1, #code do
4734             local byte = code:sub(i, i)
4735             if i < #code then
4736                 if node[byte] == nil then
4737                     node[byte] = {_type = "intermediate"}
4738                 end
4739                 node = node[byte]
4740             else
4741                 table.insert(node, {byte, combining_class, _type = "leaf"})
4742             end
4743         end
4744     ::continue::

```

```

4745 end
4746 assert(file:close())

```

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

```

4747 print("M.ccc = {}")
4748 for length, prefix_tree in pairs(prefix_trees) do
4749     local subparsers = {}
4750     depth_first_search(prefix_tree, "", function(node, path)
4751         if node._type == "leaf" then
4752             local byte, combining_class = table.unpack(node)
4753             local suffix = serialize_byte_parser(byte)
4754             .. " * Cc(" .. tostring(combining_class) .. ")"
4755             if subparsers[path] ~= nil then
4756                 subparsers[path] = subparsers[path] .. " + " .. suffix
4757             else
4758                 subparsers[path] = suffix
4759             end
4760         end
4761     end, function(_, path)
4762         if #path > 0 then
4763             local byte = path:sub(#path, #path)
4764             local parent_path = path:sub(1, #path-1)
4765             local prefix = serialize_byte_parser(byte)
4766             local suffix
4767             if subparsers[path]:find(" %+ ") then
4768                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
4769             else
4770                 suffix = prefix .. " * " .. subparsers[path]
4771             end
4772             if subparsers[parent_path] ~= nil then
4773                 subparsers[parent_path] = subparsers[parent_path]
4774                 .. " + " .. suffix
4775             else
4776                 subparsers[parent_path] = suffix
4777             end
4778         else
4779             print(
4780                 "M.ccc[" .. length .. "] = " .. (subparsers[path] or "fail")
4781             )
4782         end
4783     end)
4784 end
4785 end()
4786 print("-- luacheck: pop")
4787 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.

```
4788 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.

```
4789 function util.err(msg, exit_code)
4790   io.stderr:write("markdown.lua: " .. msg .. "\n")
4791   os.exit(exit_code or 1)
4792 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.

```
4793 function util.cache(dir, string, salt, transform, suffix)
4794   local digest = md5.sumhexa(string .. (salt or ""))
4795   local name = util.pathname(dir, digest .. suffix)
4796   local file = io.open(name, "r")
4797   local result = nil
4798   if file == nil then -- If no cache entry exists, create a new one.
4799     file = assert(io.open(name, "w"),
4800       [[Could not open file ]] .. name .. [[ for writing]])
4801     result = string
4802     if transform ~= nil then
4803       result = transform(result)
4804     end
4805     assert(file:write(result))
4806     assert(file:close())
4807   end
4808   return name, result
4809 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.

```
4810 function util.cache_verbatim(dir, string)
4811   local name = util.cache(dir, string, nil, nil, ".verbatim")
4812   return name
4813 end
```

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

```

4814 function util.table_copy(t)
4815   local u = { }
4816   for k, v in pairs(t) do u[k] = v end
4817   return setmetatable(u, getmetatable(t))
4818 end

```

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

```

4819 function util.encode_json_string(s)
4820   s = s:gsub([[\\]], [[\\]])
4821   s = s:gsub([[\"]], [[\"]])
4822   return [[\"]] .. s .. [[\"]]
4823 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 [17, Chapter 21].

```

4824 function util.expand_tabs_in_line(s, tabstop)
4825   local tab = tabstop or 4
4826   local corr = 0
4827   return (s:gsub(")\t", function(p)
4828     local sp = tab - (p - 1 + corr) % tab
4829     corr = corr - 1 + sp
4830     return string.rep(" ", sp)
4831   end))
4832 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.

```

4833 function util.walk(t, f)
4834   local typ = type(t)
4835   if typ == "string" then
4836     f(t)
4837   elseif typ == "table" then
4838     local i = 1
4839     local n
4840     n = t[i]
4841     while n do
4842       util.walk(n, f)
4843       i = i + 1
4844       n = t[i]
4845     end
4846   elseif typ == "function" then
4847     local ok, val = pcall(t)
4848     if ok then
4849       util.walk(val, f)

```

```

4850     end
4851   else
4852     f(tostring(t))
4853   end
4854 end

```

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

```

4855 function util.flatten(ary)
4856   local new = {}
4857   for _,v in ipairs(ary) do
4858     if type(v) == "table" then
4859       for _,w in ipairs(util.flatten(v)) do
4860         new[#new + 1] = w
4861       end
4862     else
4863       new[#new + 1] = v
4864     end
4865   end
4866   return new
4867 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.

```

4868 function util.rope_to_string(rope)
4869   local buffer = {}
4870   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4871   return table.concat(buffer)
4872 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.

```

4873 function util.rope_last(rope)
4874   if #rope == 0 then
4875     return nil
4876   else
4877     local l = rope[#rope]
4878     if type(l) == "table" then
4879       return util.rope_last(l)
4880     else
4881       return l
4882     end
4883   end
4884 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  $1 \leq i \leq \#ary$ .

```

4885 function util.intersperse(ary, x)
4886   local new = {}
4887   local l = #ary
4888   for i,v in ipairs(ary) do
4889     local n = #new
4890     new[n + 1] = v
4891     if i ~= l then
4892       new[n + 2] = x
4893     end
4894   end
4895   return new
4896 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  $1 \leq i \leq \#ary$ .

```

4897 function util.map(ary, f)
4898   local new = {}
4899   for i,v in ipairs(ary) do
4900     new[i] = f(v)
4901   end
4902   return new
4903 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.

```

4904 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

4905   local char_escapes_list = ""
4906   for i,_ in pairs(char_escapes) do
4907     char_escapes_list = char_escapes_list .. i
4908   end

```

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

```

4909   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.

```

4910  if string_escapes then
4911      for k,v in pairs(string_escapes) do
4912          escapable = P(k) / v + escapable
4913      end
4914  end

```

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

```

4915  local escape_string = Cs((escapable + any)^0)

```

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

```

4916  return function(s)
4917      return lpeg.match(escape_string, s)
4918  end
4919 end

```

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

```

4920 function util.pathname(dir, file)
4921  if #dir == 0 then
4922      return file
4923  else
4924      return dir .. "/" .. file
4925  end
4926 end

```

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

```

4927 function util.salt(options)
4928  local opt_string = {}
4929  for k, _ in pairs(defaultOptions) do
4930      local v = options[k]
4931      if type(v) == "table" then
4932          for _, i in ipairs(v) do
4933              opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4934          end

```

The `cacheDir` option is disregarded.

```

4935      elseif k ~= "cacheDir" then
4936          opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4937      end
4938  end
4939  table.sort(opt_string)
4940  local salt = table.concat(opt_string, ",")
4941      .. "," .. metadata.version
4942  return salt
4943 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.

```
4944 util.warning = (function()
4945   local function warning(s)
4946     io.stderr:write("Warning: " .. s .. "\n")
4947   end
4948   for _, message in ipairs(early_warnings) do
4949     warning(message)
4950   end
4951   return warning
4952 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.

```
4953 util.casefold = (function()
4954   local fail, any = P(false), P(1)
4955   local eof = -any
```

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

```
4956   local fold_character = fail
4957   for n = 1, 4 do
4958     fold_character
4959       = fold_character
4960       + unicode_data.casefold_mapping[n]
4961   end
4962   fold_character
4963     = fold_character
4964     + C(any)
```

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

```
4965   local fold_string = Ct(fold_character~0) * eof
4966   return function(s, form)
4967     local result = table.concat(lpeg.match(fold_string, s))
4968     assert(result ~= nil)
```

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

```
4969     if form == "nfd" or form == "nfkd" then
4970       result = util.normalize(result, form)
4971       result = table.concat(lpeg.match(fold_string, result))
4972       assert(result ~= nil)
4973       result = util.normalize(result, form)
4974     end
```



```

4975     return result
4976   end
4977 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.

```

4978 util.canonically_order = (function()
4979   local fail, any = P(false), P(1)
4980   local eof = -any
4981   local cont = R("\128\191")
4982   local utf8_character
4983     = R("\0\127")
4984     + R("\194\223") * cont
4985     + R("\224\239") * cont * cont
4986     + R("\240\244") * cont * cont * cont

```

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

```

4987   local classify_character = fail
4988   for n = 1, 4 do
4989     classify_character
4990       = classify_character
4991       + unicode_data.ccc[n]
4992   end
4993   classify_character
4994     = classify_character
4995     + utf8_character * Cc(0)

```

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

```

4996   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.

```

4997   return function(s)
4998     local s_len = utf8.len(s)
4999     if s == false or s_len <= 1 then
5000       return s
5001     end

```

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

```

5002     local classes = lpeg.match(classify_string, s)
5003     if classes == nil then
5004       return s
5005     end
5006     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.

```
5007     local non_starter_ranges = {}
5008     local first_non_starter, last_non_starter = nil, nil
5009     for i = 1, #classes do
5010         if first_non_starter == nil then
5011             if classes[i] ~= 0 then
5012                 first_non_starter, last_non_starter = i, i
5013             end
5014         else
5015             if classes[i] == 0 then
5016                 table.insert(
5017                     non_starter_ranges,
5018                     {first_non_starter, last_non_starter}
5019                 )
5020                 first_non_starter, last_non_starter = nil, nil
5021             else
5022                 last_non_starter = i
5023             end
5024         end
5025     end
5026     if first_non_starter ~= nil then
5027         table.insert(
5028             non_starter_ranges,
5029             {first_non_starter, last_non_starter}
5030         )
5031     end
5032     if #non_starter_ranges == 0 then
5033         return s
5034     end
5035     local max_range_length = 0
5036     for _, range in ipairs(non_starter_ranges) do
5037         local range_start, range_end = table.unpack(range)
5038         local range_length = range_end - range_start + 1
5039         if range_length > max_range_length then
5040             max_range_length = range_length
5041         end
5042     end
5043     if max_range_length <= 1 then
5044         return s
5045     end
```

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

```
5046     local buffer = {}
5047     for _, code in utf8.codes(s) do
5048         local char = utf8.char(code)
```

```

5049     table.insert(buffer, char)
5050   end
5051   assert(#buffer == s_len)

```

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

```

5052   for _, range in ipairs(non_starter_ranges) do
5053     local range_start, range_end = table.unpack(range)
5054     local range_length = range_end - range_start + 1
5055     for _ = 1, range_length - 1 do
5056       local swapped = false
5057       for i = range_start, range_end - 1 do
5058         local j = i + 1
5059         if classes[i] > classes[j] then
5060           classes[i], classes[j] = classes[j], classes[i]
5061           buffer[i], buffer[j] = buffer[j], buffer[i]
5062           swapped = true
5063         end
5064       end
5065       if not swapped then
5066         break
5067       end
5068     end
5069   end

```

Finally, concatenate the buffer and return an ordered string.

```

5070     return table.concat(buffer, "")
5071   end
5072 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.

```

5073 util.decompose = (function()
5074   local fail, any = P(false), P(1)
5075   local eof = -any
5076   local decomposition_types = {"canonical", "compatibility"}

```

First, define parsers that will decompose a character.

```

5077   local decompose_character = {}
5078   for _, decomposition_type in ipairs(decomposition_types) do
5079     decompose_character[decomposition_type] = fail
5080     for n = 1, 4 do
5081       decompose_character[decomposition_type]
5082         = decompose_character[decomposition_type]
5083         + unicode_data.decomposition_mapping[decomposition_type][n]
5084     end

```

```

5085     decompose_character[decomposition_type]
5086     = decompose_character[decomposition_type]
5087     + C(any)
5088 end
5089 local hangul = unicode_data.decomposition_mapping.hangul
5090 decompose_character.hangul = {}
5091 for syllable_type, _ in pairs(hangul) do
5092     decompose_character.hangul[syllable_type] = fail
5093     for n = 1, 4 do
5094         decompose_character.hangul[syllable_type]
5095         = decompose_character.hangul[syllable_type]
5096         + hangul[syllable_type][n]
5097     end
5098     decompose_character.hangul[syllable_type]
5099     = decompose_character.hangul[syllable_type]
5100     + C(any)
5101 end

```

Next, define a parser that will decompose a string.

```

5102 local decompose_string = {}
5103 for _, decomposition_type in ipairs(decomposition_types) do
5104     decompose_string[decomposition_type]
5105     = Ct(decompose_character[decomposition_type]^0) * eof
5106 end
5107 decompose_string.hangul = {}
5108 for syllable_type, _ in pairs(hangul) do
5109     decompose_string.hangul[syllable_type]
5110     = Ct(decompose_character.hangul[syllable_type]^0) * eof
5111 end
5112 local function _decompose(s, parser)
5113     assert(s ~= nil)
5114     local result = table.concat(lpeg.match(parser, s), "")
5115     assert(result ~= nil)
5116     return result
5117 end
5118 return function(s, decomposition_type)
5119     assert(
5120         decomposition_type == "canonical"
5121         or decomposition_type == "compatibility"
5122     )
5123     local prev_s
5124     local next_s = s
5125     repeat
5126         prev_s = next_s
5127         local function decompose(...) next_s = _decompose(next_s, ...) end
5128         decompose(decompose_string.canonical)
5129         if decomposition_type == "compatibility" then
5130             decompose(decompose_string.compatibility)

```

```

5131     end
5132     decompose(decompose_string.hangul.LVT)
5133     decompose(decompose_string.hangul.LV)
5134     until prev_s == next_s
5135     return util.canonically_order(next_s)
5136 end
5137 end)()

```

The `util.compose` method performs the canonical composition of a UTF-8-encoded canonically ordered Unicode string `s` based on the low-level parsers in `unicode_data.composition_mapping`, defined in Section 3.1.1.3, and definitions from the Hangul syllable (de)composition algorithm, defined in Section 3.1.1.2. Unlike the low-level parsers, this high-level function is invariant to the number of bytes the Unicode characters occupy after conversion to UTF-8.

```

5138 util.compose = (function()
5139   local fail, any = P(false), P(1)
5140   local eof = -any
5141   local cont = R("\128\191")
5142   local utf8_character
5143     = R("\0\127")
5144     + R("\194\223") * cont
5145     + R("\224\239") * cont * cont
5146     + R("\240\244") * cont * cont * cont

```

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

```

5147   local classify_character = fail
5148   for n = 1, 4 do
5149     classify_character
5150       = classify_character
5151       + unicode_data.ccc[n]
5152   end
5153   classify_character
5154     = classify_character
5155     + utf8_character * Cc(0)

```

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

```

5156   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.

```

5157   local compose_characters = fail
5158   for m = 1, 4 do
5159     local starter = #unicode_data.composition_mapping.starters[m]
5160     local both = fail
5161     for n = 1, 4 do
5162       both = (
5163         both

```

```

5164         + #unicode_data.composition_mapping.both[m] [n]
5165         * unicode_data.composition_mapping.both[m] [n]
5166     )
5167     end
5168     compose_characters = compose_characters + starter * both
5169 end

```

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

```

5170     return function(s)
5171         local s_len = utf8.len(s)
5172         if s == false or s_len <= 1 then
5173             return s
5174         end

```

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

```

5175         local classes = lpeg.match(classify_string, s)
5176         if classes == nil then
5177             return s
5178         end
5179         assert(#classes == s_len)

```

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

```

5180         local buffer = {}
5181         for _, code in utf8.codes(s) do
5182             local char = utf8.char(code)
5183             table.insert(buffer, char)
5184         end
5185         assert(#buffer == s_len)

```

Finally, implement the composition algorithm.

First, find the first starter character in the string.

```

5186         local starter = 1
5187         while starter <= s_len and classes[starter] ~= 0 do
5188             starter = starter + 1
5189         end
5190         local candidate_combining_mark = starter + 1

```

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

```

5191         while candidate_combining_mark <= s_len do
5192             local L = buffer[starter]
5193             local C = buffer[candidate_combining_mark]
5194             local P = lpeg.match(compose_characters, L .. C)
5195             if P ~= nil then
5196                 buffer[starter] = P
5197                 buffer[candidate_combining_mark] = ""
5198             else
5199                 if classes[candidate_combining_mark] == 0 then

```

```

5200         starter = candidate_combining_mark
5201     end
5202     candidate_combining_mark = candidate_combining_mark + 1
5203 end
5204 assert(starter <= s_len)
5205 end

```

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

```

5206     for i = 1, s_len - 1 do
5207         local last, ch = buffer[i], buffer[i + 1]
5208         if last ~= "" and ch ~= "" then
5209             local composite = unicode_data.hangul_compose(last, ch)
5210             if composite ~= nil then
5211                 buffer[i] = ""
5212                 buffer[i + 1] = composite
5213             end
5214         end
5215     end

```

Finally, concatenate the buffer and return an ordered string.

```

5216     return table.concat(buffer, "")
5217 end
5218 end)()

```

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

```

5219 function util.normalize(s, form)
5220     if form == "nfd" then
5221         return util.decompose(s, "canonical")
5222     elseif form == "nfkd" then
5223         return util.decompose(s, "compatibility")
5224     elseif form == "nfc" then
5225         return util.compose(util.decompose(s, "canonical"))
5226     elseif form == "nfkc" then
5227         return util.compose(util.decompose(s, "compatibility"))
5228     else
5229         error(string.format('Unexpected normal form "%s"', form))
5230     end
5231 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.

```

5232 local entities = {}
5233
5234 local character_entities = {

```

5235 ["Tab"] = 9,  
5236 ["NewLine"] = 10,  
5237 ["excl"] = 33,  
5238 ["QUOT"] = 34,  
5239 ["quot"] = 34,  
5240 ["num"] = 35,  
5241 ["dollar"] = 36,  
5242 ["percent"] = 37,  
5243 ["AMP"] = 38,  
5244 ["amp"] = 38,  
5245 ["apos"] = 39,  
5246 ["lpar"] = 40,  
5247 ["rpar"] = 41,  
5248 ["ast"] = 42,  
5249 ["midast"] = 42,  
5250 ["plus"] = 43,  
5251 ["comma"] = 44,  
5252 ["period"] = 46,  
5253 ["sol"] = 47,  
5254 ["colon"] = 58,  
5255 ["semi"] = 59,  
5256 ["LT"] = 60,  
5257 ["lt"] = 60,  
5258 ["nvlT"] = {60, 8402},  
5259 ["bne"] = {61, 8421},  
5260 ["equals"] = 61,  
5261 ["GT"] = 62,  
5262 ["gt"] = 62,  
5263 ["nvgt"] = {62, 8402},  
5264 ["quest"] = 63,  
5265 ["commat"] = 64,  
5266 ["lbrack"] = 91,  
5267 ["lsqb"] = 91,  
5268 ["bsol"] = 92,  
5269 ["rbrack"] = 93,  
5270 ["rsqb"] = 93,  
5271 ["Hat"] = 94,  
5272 ["UnderBar"] = 95,  
5273 ["lowbar"] = 95,  
5274 ["DiacriticalGrave"] = 96,  
5275 ["grave"] = 96,  
5276 ["fjlig"] = {102, 106},  
5277 ["lbrace"] = 123,  
5278 ["lcub"] = 123,  
5279 ["VerticalLine"] = 124,  
5280 ["verbar"] = 124,  
5281 ["vert"] = 124,



5282 ["rbrace"] = 125,  
5283 ["rcub"] = 125,  
5284 ["NonBreakingSpace"] = 160,  
5285 ["nbsp"] = 160,  
5286 ["iexcl"] = 161,  
5287 ["cent"] = 162,  
5288 ["pound"] = 163,  
5289 ["curren"] = 164,  
5290 ["yen"] = 165,  
5291 ["brvbar"] = 166,  
5292 ["sect"] = 167,  
5293 ["Dot"] = 168,  
5294 ["DoubleDot"] = 168,  
5295 ["die"] = 168,  
5296 ["uml"] = 168,  
5297 ["COPY"] = 169,  
5298 ["copy"] = 169,  
5299 ["ordf"] = 170,  
5300 ["laquo"] = 171,  
5301 ["not"] = 172,  
5302 ["shy"] = 173,  
5303 ["REG"] = 174,  
5304 ["circledR"] = 174,  
5305 ["reg"] = 174,  
5306 ["macr"] = 175,  
5307 ["strns"] = 175,  
5308 ["deg"] = 176,  
5309 ["PlusMinus"] = 177,  
5310 ["plusmn"] = 177,  
5311 ["pm"] = 177,  
5312 ["sup2"] = 178,  
5313 ["sup3"] = 179,  
5314 ["DiacriticalAcute"] = 180,  
5315 ["acute"] = 180,  
5316 ["micro"] = 181,  
5317 ["para"] = 182,  
5318 ["CenterDot"] = 183,  
5319 ["centerdot"] = 183,  
5320 ["middot"] = 183,  
5321 ["Cedilla"] = 184,  
5322 ["cedil"] = 184,  
5323 ["sup1"] = 185,  
5324 ["ordm"] = 186,  
5325 ["raquo"] = 187,  
5326 ["frac14"] = 188,  
5327 ["frac12"] = 189,  
5328 ["half"] = 189,

5329 ["frac34"] = 190,  
5330 ["iquest"] = 191,  
5331 ["Agrave"] = 192,  
5332 ["Aacute"] = 193,  
5333 ["Acirc"] = 194,  
5334 ["Atilde"] = 195,  
5335 ["Auml"] = 196,  
5336 ["Aring"] = 197,  
5337 ["angst"] = 197,  
5338 ["AElig"] = 198,  
5339 ["Ccedil"] = 199,  
5340 ["Egrave"] = 200,  
5341 ["Eacute"] = 201,  
5342 ["Ecirc"] = 202,  
5343 ["Euml"] = 203,  
5344 ["Igrave"] = 204,  
5345 ["Iacute"] = 205,  
5346 ["Icirc"] = 206,  
5347 ["Iuml"] = 207,  
5348 ["ETH"] = 208,  
5349 ["Ntilde"] = 209,  
5350 ["Ograve"] = 210,  
5351 ["Oacute"] = 211,  
5352 ["Ocirc"] = 212,  
5353 ["Otilde"] = 213,  
5354 ["Ouml"] = 214,  
5355 ["times"] = 215,  
5356 ["Oslash"] = 216,  
5357 ["Ugrave"] = 217,  
5358 ["Uacute"] = 218,  
5359 ["Ucirc"] = 219,  
5360 ["Uuml"] = 220,  
5361 ["Yacute"] = 221,  
5362 ["THORN"] = 222,  
5363 ["szlig"] = 223,  
5364 ["agrave"] = 224,  
5365 ["aacute"] = 225,  
5366 ["acirc"] = 226,  
5367 ["atilde"] = 227,  
5368 ["auml"] = 228,  
5369 ["aring"] = 229,  
5370 ["aelig"] = 230,  
5371 ["ccedil"] = 231,  
5372 ["egrave"] = 232,  
5373 ["eacute"] = 233,  
5374 ["ecirc"] = 234,  
5375 ["euml"] = 235,

5376 ["igrave"] = 236,  
5377 ["iacute"] = 237,  
5378 ["icirc"] = 238,  
5379 ["iuml"] = 239,  
5380 ["eth"] = 240,  
5381 ["ntilde"] = 241,  
5382 ["ograve"] = 242,  
5383 ["oacute"] = 243,  
5384 ["ocirc"] = 244,  
5385 ["otilde"] = 245,  
5386 ["ouml"] = 246,  
5387 ["div"] = 247,  
5388 ["divide"] = 247,  
5389 ["oslash"] = 248,  
5390 ["ugrave"] = 249,  
5391 ["uacute"] = 250,  
5392 ["ucirc"] = 251,  
5393 ["uuml"] = 252,  
5394 ["yacute"] = 253,  
5395 ["thorn"] = 254,  
5396 ["yuml"] = 255,  
5397 ["Amacr"] = 256,  
5398 ["amacr"] = 257,  
5399 ["Abreve"] = 258,  
5400 ["abreve"] = 259,  
5401 ["Aogon"] = 260,  
5402 ["aogon"] = 261,  
5403 ["Cacute"] = 262,  
5404 ["cacute"] = 263,  
5405 ["Ccirc"] = 264,  
5406 ["ccirc"] = 265,  
5407 ["Cdot"] = 266,  
5408 ["cdot"] = 267,  
5409 ["Ccaron"] = 268,  
5410 ["ccaron"] = 269,  
5411 ["Dcaron"] = 270,  
5412 ["dcaron"] = 271,  
5413 ["Dstrok"] = 272,  
5414 ["dstrok"] = 273,  
5415 ["Emacr"] = 274,  
5416 ["emacr"] = 275,  
5417 ["Edot"] = 278,  
5418 ["edot"] = 279,  
5419 ["Eogon"] = 280,  
5420 ["eogon"] = 281,  
5421 ["Ecaron"] = 282,  
5422 ["ecaron"] = 283,

5423 ["Gcirc"] = 284,  
5424 ["gcirc"] = 285,  
5425 ["Gbreve"] = 286,  
5426 ["gbreve"] = 287,  
5427 ["Gdot"] = 288,  
5428 ["gdot"] = 289,  
5429 ["Gcedil"] = 290,  
5430 ["Hcirc"] = 292,  
5431 ["hcirc"] = 293,  
5432 ["Hstrook"] = 294,  
5433 ["hstrook"] = 295,  
5434 ["Itilde"] = 296,  
5435 ["itilde"] = 297,  
5436 ["Imacr"] = 298,  
5437 ["imacr"] = 299,  
5438 ["Iogon"] = 302,  
5439 ["iogon"] = 303,  
5440 ["Idot"] = 304,  
5441 ["imath"] = 305,  
5442 ["inodot"] = 305,  
5443 ["IJlig"] = 306,  
5444 ["ijlig"] = 307,  
5445 ["Jcirc"] = 308,  
5446 ["jcirc"] = 309,  
5447 ["Kcedil"] = 310,  
5448 ["kcedil"] = 311,  
5449 ["kgreen"] = 312,  
5450 ["Lacute"] = 313,  
5451 ["lacute"] = 314,  
5452 ["Lcedil"] = 315,  
5453 ["lcedil"] = 316,  
5454 ["Lcaron"] = 317,  
5455 ["lcaron"] = 318,  
5456 ["Lmidot"] = 319,  
5457 ["lmidot"] = 320,  
5458 ["Lstrook"] = 321,  
5459 ["lstrook"] = 322,  
5460 ["Nacute"] = 323,  
5461 ["nacute"] = 324,  
5462 ["Ncedil"] = 325,  
5463 ["ncedil"] = 326,  
5464 ["Ncaron"] = 327,  
5465 ["ncaron"] = 328,  
5466 ["napos"] = 329,  
5467 ["ENG"] = 330,  
5468 ["eng"] = 331,  
5469 ["Omacr"] = 332,

5470 ["omacr"] = 333,  
5471 ["Odblac"] = 336,  
5472 ["odblac"] = 337,  
5473 ["OElig"] = 338,  
5474 ["oelig"] = 339,  
5475 ["Racute"] = 340,  
5476 ["racute"] = 341,  
5477 ["Rcedil"] = 342,  
5478 ["rcedil"] = 343,  
5479 ["Rcaron"] = 344,  
5480 ["rcaron"] = 345,  
5481 ["Sacute"] = 346,  
5482 ["sacute"] = 347,  
5483 ["Scirc"] = 348,  
5484 ["scirc"] = 349,  
5485 ["Scedil"] = 350,  
5486 ["scedil"] = 351,  
5487 ["Scaron"] = 352,  
5488 ["scaron"] = 353,  
5489 ["Tcedil"] = 354,  
5490 ["tcedil"] = 355,  
5491 ["Tcaron"] = 356,  
5492 ["tcaron"] = 357,  
5493 ["Tstrok"] = 358,  
5494 ["tstrok"] = 359,  
5495 ["Utilde"] = 360,  
5496 ["utilde"] = 361,  
5497 ["Umacr"] = 362,  
5498 ["umacr"] = 363,  
5499 ["Ubreve"] = 364,  
5500 ["ubreve"] = 365,  
5501 ["Uring"] = 366,  
5502 ["uring"] = 367,  
5503 ["Udblac"] = 368,  
5504 ["udblac"] = 369,  
5505 ["Uogon"] = 370,  
5506 ["uogon"] = 371,  
5507 ["Wcirc"] = 372,  
5508 ["wcirc"] = 373,  
5509 ["Ycirc"] = 374,  
5510 ["ycirc"] = 375,  
5511 ["Yuml"] = 376,  
5512 ["Zacute"] = 377,  
5513 ["zacute"] = 378,  
5514 ["Zdot"] = 379,  
5515 ["zdot"] = 380,  
5516 ["Zcaron"] = 381,

5517 ["zcaron"] = 382,  
5518 ["fnof"] = 402,  
5519 ["imped"] = 437,  
5520 ["gacute"] = 501,  
5521 ["jmath"] = 567,  
5522 ["circ"] = 710,  
5523 ["Hacek"] = 711,  
5524 ["caron"] = 711,  
5525 ["Breve"] = 728,  
5526 ["breve"] = 728,  
5527 ["DiacriticalDot"] = 729,  
5528 ["dot"] = 729,  
5529 ["ring"] = 730,  
5530 ["ogon"] = 731,  
5531 ["DiacriticalTilde"] = 732,  
5532 ["tilde"] = 732,  
5533 ["DiacriticalDoubleAcute"] = 733,  
5534 ["dblac"] = 733,  
5535 ["DownBreve"] = 785,  
5536 ["Alpha"] = 913,  
5537 ["Beta"] = 914,  
5538 ["Gamma"] = 915,  
5539 ["Delta"] = 916,  
5540 ["Epsilon"] = 917,  
5541 ["Zeta"] = 918,  
5542 ["Eta"] = 919,  
5543 ["Theta"] = 920,  
5544 ["Iota"] = 921,  
5545 ["Kappa"] = 922,  
5546 ["Lambda"] = 923,  
5547 ["Mu"] = 924,  
5548 ["Nu"] = 925,  
5549 ["Xi"] = 926,  
5550 ["Omicron"] = 927,  
5551 ["Pi"] = 928,  
5552 ["Rho"] = 929,  
5553 ["Sigma"] = 931,  
5554 ["Tau"] = 932,  
5555 ["Upsilon"] = 933,  
5556 ["Phi"] = 934,  
5557 ["Chi"] = 935,  
5558 ["Psi"] = 936,  
5559 ["Omega"] = 937,  
5560 ["ohm"] = 937,  
5561 ["alpha"] = 945,  
5562 ["beta"] = 946,  
5563 ["gamma"] = 947,

5564 ["delta"] = 948,  
5565 ["epsi"] = 949,  
5566 ["epsilon"] = 949,  
5567 ["zeta"] = 950,  
5568 ["eta"] = 951,  
5569 ["theta"] = 952,  
5570 ["iota"] = 953,  
5571 ["kappa"] = 954,  
5572 ["lambda"] = 955,  
5573 ["mu"] = 956,  
5574 ["nu"] = 957,  
5575 ["xi"] = 958,  
5576 ["omicron"] = 959,  
5577 ["pi"] = 960,  
5578 ["rho"] = 961,  
5579 ["sigmaf"] = 962,  
5580 ["sigmav"] = 962,  
5581 ["varsigma"] = 962,  
5582 ["sigma"] = 963,  
5583 ["tau"] = 964,  
5584 ["upsi"] = 965,  
5585 ["upsilon"] = 965,  
5586 ["phi"] = 966,  
5587 ["chi"] = 967,  
5588 ["psi"] = 968,  
5589 ["omega"] = 969,  
5590 ["thetasym"] = 977,  
5591 ["thetav"] = 977,  
5592 ["vartheta"] = 977,  
5593 ["Upsi"] = 978,  
5594 ["upsih"] = 978,  
5595 ["phiv"] = 981,  
5596 ["straightphi"] = 981,  
5597 ["varphi"] = 981,  
5598 ["piv"] = 982,  
5599 ["varpi"] = 982,  
5600 ["Gammad"] = 988,  
5601 ["digamma"] = 989,  
5602 ["gammad"] = 989,  
5603 ["kappav"] = 1008,  
5604 ["varkappa"] = 1008,  
5605 ["rhov"] = 1009,  
5606 ["varrho"] = 1009,  
5607 ["epsiv"] = 1013,  
5608 ["straightepsilon"] = 1013,  
5609 ["varepsilon"] = 1013,  
5610 ["backepsilon"] = 1014,

5611 ["bepsi"] = 1014,  
5612 ["IOcy"] = 1025,  
5613 ["DJcy"] = 1026,  
5614 ["GJcy"] = 1027,  
5615 ["Jukcy"] = 1028,  
5616 ["DScy"] = 1029,  
5617 ["Iukcy"] = 1030,  
5618 ["YIcy"] = 1031,  
5619 ["Jsercy"] = 1032,  
5620 ["LJcy"] = 1033,  
5621 ["NJcy"] = 1034,  
5622 ["TSHcy"] = 1035,  
5623 ["KJcy"] = 1036,  
5624 ["Ubrcy"] = 1038,  
5625 ["DZcy"] = 1039,  
5626 ["Acy"] = 1040,  
5627 ["Bcy"] = 1041,  
5628 ["Vcy"] = 1042,  
5629 ["Gcy"] = 1043,  
5630 ["Dcy"] = 1044,  
5631 ["IEcy"] = 1045,  
5632 ["ZHcy"] = 1046,  
5633 ["Zcy"] = 1047,  
5634 ["Icy"] = 1048,  
5635 ["Jcy"] = 1049,  
5636 ["Kcy"] = 1050,  
5637 ["Lcy"] = 1051,  
5638 ["Mcy"] = 1052,  
5639 ["Ncy"] = 1053,  
5640 ["Ocy"] = 1054,  
5641 ["Pcy"] = 1055,  
5642 ["Rcy"] = 1056,  
5643 ["Scy"] = 1057,  
5644 ["Tcy"] = 1058,  
5645 ["Ucy"] = 1059,  
5646 ["Fcy"] = 1060,  
5647 ["KHcy"] = 1061,  
5648 ["TScy"] = 1062,  
5649 ["CHcy"] = 1063,  
5650 ["SHcy"] = 1064,  
5651 ["SHCHcy"] = 1065,  
5652 ["HARDcy"] = 1066,  
5653 ["Ycy"] = 1067,  
5654 ["SOFTcy"] = 1068,  
5655 ["Ecy"] = 1069,  
5656 ["YUcy"] = 1070,  
5657 ["YAcy"] = 1071,



5658 ["acy"] = 1072,  
5659 ["bcy"] = 1073,  
5660 ["vcy"] = 1074,  
5661 ["gcy"] = 1075,  
5662 ["dcy"] = 1076,  
5663 ["iecy"] = 1077,  
5664 ["zhcy"] = 1078,  
5665 ["zcy"] = 1079,  
5666 ["icy"] = 1080,  
5667 ["jcy"] = 1081,  
5668 ["kcy"] = 1082,  
5669 ["lcy"] = 1083,  
5670 ["mcy"] = 1084,  
5671 ["ncy"] = 1085,  
5672 ["ocy"] = 1086,  
5673 ["pcy"] = 1087,  
5674 ["rcy"] = 1088,  
5675 ["scy"] = 1089,  
5676 ["tcy"] = 1090,  
5677 ["ucy"] = 1091,  
5678 ["fcy"] = 1092,  
5679 ["khcy"] = 1093,  
5680 ["tscy"] = 1094,  
5681 ["chcy"] = 1095,  
5682 ["shcy"] = 1096,  
5683 ["shchcy"] = 1097,  
5684 ["hardcy"] = 1098,  
5685 ["ycy"] = 1099,  
5686 ["softcy"] = 1100,  
5687 ["ecy"] = 1101,  
5688 ["yucy"] = 1102,  
5689 ["yacy"] = 1103,  
5690 ["iocy"] = 1105,  
5691 ["djcy"] = 1106,  
5692 ["gjcy"] = 1107,  
5693 ["jukcy"] = 1108,  
5694 ["dscy"] = 1109,  
5695 ["iukcy"] = 1110,  
5696 ["yicy"] = 1111,  
5697 ["jsercy"] = 1112,  
5698 ["ljcy"] = 1113,  
5699 ["njcy"] = 1114,  
5700 ["tshcy"] = 1115,  
5701 ["kjcy"] = 1116,  
5702 ["ubrcy"] = 1118,  
5703 ["dzcy"] = 1119,  
5704 ["ensp"] = 8194,

5705 ["emsp"] = 8195,  
5706 ["emsp13"] = 8196,  
5707 ["emsp14"] = 8197,  
5708 ["numsp"] = 8199,  
5709 ["puncsp"] = 8200,  
5710 ["ThinSpace"] = 8201,  
5711 ["thinsp"] = 8201,  
5712 ["VeryThinSpace"] = 8202,  
5713 ["hairsp"] = 8202,  
5714 ["NegativeMediumSpace"] = 8203,  
5715 ["NegativeThickSpace"] = 8203,  
5716 ["NegativeThinSpace"] = 8203,  
5717 ["NegativeVeryThinSpace"] = 8203,  
5718 ["ZeroWidthSpace"] = 8203,  
5719 ["zwnj"] = 8204,  
5720 ["zwj"] = 8205,  
5721 ["lrm"] = 8206,  
5722 ["rlm"] = 8207,  
5723 ["dash"] = 8208,  
5724 ["hyphen"] = 8208,  
5725 ["ndash"] = 8211,  
5726 ["mdash"] = 8212,  
5727 ["horbar"] = 8213,  
5728 ["Verbar"] = 8214,  
5729 ["Vert"] = 8214,  
5730 ["OpenCurlyQuote"] = 8216,  
5731 ["lsquo"] = 8216,  
5732 ["CloseCurlyQuote"] = 8217,  
5733 ["rsquo"] = 8217,  
5734 ["rsquor"] = 8217,  
5735 ["lsquor"] = 8218,  
5736 ["sbquo"] = 8218,  
5737 ["OpenCurlyDoubleQuote"] = 8220,  
5738 ["ldquo"] = 8220,  
5739 ["CloseCurlyDoubleQuote"] = 8221,  
5740 ["rdquo"] = 8221,  
5741 ["rdquor"] = 8221,  
5742 ["bdquo"] = 8222,  
5743 ["ldquor"] = 8222,  
5744 ["dagger"] = 8224,  
5745 ["Dagger"] = 8225,  
5746 ["ddagger"] = 8225,  
5747 ["bull"] = 8226,  
5748 ["bullet"] = 8226,  
5749 ["nldr"] = 8229,  
5750 ["hellip"] = 8230,  
5751 ["mlldr"] = 8230,

5752 ["permil"] = 8240,  
5753 ["pertenk"] = 8241,  
5754 ["prime"] = 8242,  
5755 ["Prime"] = 8243,  
5756 ["tprime"] = 8244,  
5757 ["backprime"] = 8245,  
5758 ["bprime"] = 8245,  
5759 ["lsaquo"] = 8249,  
5760 ["rsaquo"] = 8250,  
5761 ["OverBar"] = 8254,  
5762 ["oline"] = 8254,  
5763 ["caret"] = 8257,  
5764 ["hybull"] = 8259,  
5765 ["frasl"] = 8260,  
5766 ["bsemi"] = 8271,  
5767 ["qprime"] = 8279,  
5768 ["MediumSpace"] = 8287,  
5769 ["ThickSpace"] = {8287, 8202},  
5770 ["NoBreak"] = 8288,  
5771 ["ApplyFunction"] = 8289,  
5772 ["af"] = 8289,  
5773 ["InvisibleTimes"] = 8290,  
5774 ["it"] = 8290,  
5775 ["InvisibleComma"] = 8291,  
5776 ["ic"] = 8291,  
5777 ["euro"] = 8364,  
5778 ["TripleDot"] = 8411,  
5779 ["tdot"] = 8411,  
5780 ["DotDot"] = 8412,  
5781 ["Copf"] = 8450,  
5782 ["complexes"] = 8450,  
5783 ["incare"] = 8453,  
5784 ["gscr"] = 8458,  
5785 ["HilbertSpace"] = 8459,  
5786 ["Hscr"] = 8459,  
5787 ["hamilt"] = 8459,  
5788 ["Hfr"] = 8460,  
5789 ["Poincareplane"] = 8460,  
5790 ["Hopf"] = 8461,  
5791 ["quaternions"] = 8461,  
5792 ["planckh"] = 8462,  
5793 ["hbar"] = 8463,  
5794 ["hslash"] = 8463,  
5795 ["planck"] = 8463,  
5796 ["plankv"] = 8463,  
5797 ["Iscr"] = 8464,  
5798 ["imagline"] = 8464,

5799 ["Ifr"] = 8465,  
5800 ["Im"] = 8465,  
5801 ["image"] = 8465,  
5802 ["imagpart"] = 8465,  
5803 ["Laplacetrif"] = 8466,  
5804 ["Lscr"] = 8466,  
5805 ["lagran"] = 8466,  
5806 ["ell"] = 8467,  
5807 ["Nopf"] = 8469,  
5808 ["naturals"] = 8469,  
5809 ["numero"] = 8470,  
5810 ["copysr"] = 8471,  
5811 ["weierp"] = 8472,  
5812 ["wp"] = 8472,  
5813 ["Popf"] = 8473,  
5814 ["primes"] = 8473,  
5815 ["Qopf"] = 8474,  
5816 ["rationals"] = 8474,  
5817 ["Rscr"] = 8475,  
5818 ["realine"] = 8475,  
5819 ["Re"] = 8476,  
5820 ["Rfr"] = 8476,  
5821 ["real"] = 8476,  
5822 ["realpart"] = 8476,  
5823 ["Ropf"] = 8477,  
5824 ["reals"] = 8477,  
5825 ["rx"] = 8478,  
5826 ["TRADE"] = 8482,  
5827 ["trade"] = 8482,  
5828 ["Zopf"] = 8484,  
5829 ["integers"] = 8484,  
5830 ["mho"] = 8487,  
5831 ["Zfr"] = 8488,  
5832 ["zeetrif"] = 8488,  
5833 ["iiota"] = 8489,  
5834 ["Bernoullis"] = 8492,  
5835 ["Bscr"] = 8492,  
5836 ["bernou"] = 8492,  
5837 ["Cayleys"] = 8493,  
5838 ["Cfr"] = 8493,  
5839 ["escr"] = 8495,  
5840 ["Escr"] = 8496,  
5841 ["expectation"] = 8496,  
5842 ["Fouriertrif"] = 8497,  
5843 ["Fscr"] = 8497,  
5844 ["Mellintrif"] = 8499,  
5845 ["Mscr"] = 8499,

5846 ["phmmat"] = 8499,  
5847 ["order"] = 8500,  
5848 ["orderof"] = 8500,  
5849 ["oscr"] = 8500,  
5850 ["alefsym"] = 8501,  
5851 ["aleph"] = 8501,  
5852 ["beth"] = 8502,  
5853 ["gimel"] = 8503,  
5854 ["daleth"] = 8504,  
5855 ["CapitalDifferentialD"] = 8517,  
5856 ["DD"] = 8517,  
5857 ["DifferentialD"] = 8518,  
5858 ["dd"] = 8518,  
5859 ["ExponentialE"] = 8519,  
5860 ["ee"] = 8519,  
5861 ["exponentiale"] = 8519,  
5862 ["ImaginaryI"] = 8520,  
5863 ["ii"] = 8520,  
5864 ["frac13"] = 8531,  
5865 ["frac23"] = 8532,  
5866 ["frac15"] = 8533,  
5867 ["frac25"] = 8534,  
5868 ["frac35"] = 8535,  
5869 ["frac45"] = 8536,  
5870 ["frac16"] = 8537,  
5871 ["frac56"] = 8538,  
5872 ["frac18"] = 8539,  
5873 ["frac38"] = 8540,  
5874 ["frac58"] = 8541,  
5875 ["frac78"] = 8542,  
5876 ["LeftArrow"] = 8592,  
5877 ["ShortLeftArrow"] = 8592,  
5878 ["larr"] = 8592,  
5879 ["leftarrow"] = 8592,  
5880 ["slarr"] = 8592,  
5881 ["ShortUpArrow"] = 8593,  
5882 ["UpArrow"] = 8593,  
5883 ["uarr"] = 8593,  
5884 ["uparrow"] = 8593,  
5885 ["RightArrow"] = 8594,  
5886 ["ShortRightArrow"] = 8594,  
5887 ["rarr"] = 8594,  
5888 ["rightarrow"] = 8594,  
5889 ["srarr"] = 8594,  
5890 ["DownArrow"] = 8595,  
5891 ["ShortDownArrow"] = 8595,  
5892 ["darr"] = 8595,

5893 ["downarrow"] = 8595,  
5894 ["LeftRightArrow"] = 8596,  
5895 ["harr"] = 8596,  
5896 ["leftrightarrow"] = 8596,  
5897 ["UpDownArrow"] = 8597,  
5898 ["updownarrow"] = 8597,  
5899 ["varr"] = 8597,  
5900 ["UpperLeftArrow"] = 8598,  
5901 ["nwarr"] = 8598,  
5902 ["nwarrow"] = 8598,  
5903 ["UpperRightArrow"] = 8599,  
5904 ["nearr"] = 8599,  
5905 ["nearrow"] = 8599,  
5906 ["LowerRightArrow"] = 8600,  
5907 ["searr"] = 8600,  
5908 ["searrow"] = 8600,  
5909 ["LowerLeftArrow"] = 8601,  
5910 ["swarr"] = 8601,  
5911 ["swarrow"] = 8601,  
5912 ["nlarr"] = 8602,  
5913 ["nleftarrow"] = 8602,  
5914 ["nrarr"] = 8603,  
5915 ["nrightarrow"] = 8603,  
5916 ["nrarrw"] = {8605, 824},  
5917 ["rarrw"] = 8605,  
5918 ["rightsquigarrow"] = 8605,  
5919 ["Larr"] = 8606,  
5920 ["twoheadleftarrow"] = 8606,  
5921 ["Uarr"] = 8607,  
5922 ["Rarr"] = 8608,  
5923 ["twoheadrightarrow"] = 8608,  
5924 ["Darr"] = 8609,  
5925 ["larrtl"] = 8610,  
5926 ["leftarrowtail"] = 8610,  
5927 ["rarrtl"] = 8611,  
5928 ["rightarrowtail"] = 8611,  
5929 ["LeftTeeArrow"] = 8612,  
5930 ["mapstoleft"] = 8612,  
5931 ["UpTeeArrow"] = 8613,  
5932 ["mapstoup"] = 8613,  
5933 ["RightTeeArrow"] = 8614,  
5934 ["map"] = 8614,  
5935 ["mapsto"] = 8614,  
5936 ["DownTeeArrow"] = 8615,  
5937 ["mapstodown"] = 8615,  
5938 ["hookleftarrow"] = 8617,  
5939 ["larrhk"] = 8617,

5940 ["hookrightarrow"] = 8618,  
5941 ["rarrhk"] = 8618,  
5942 ["larrlp"] = 8619,  
5943 ["looparrowleft"] = 8619,  
5944 ["looparrowright"] = 8620,  
5945 ["rarrlp"] = 8620,  
5946 ["harrw"] = 8621,  
5947 ["leftrightsquigarrow"] = 8621,  
5948 ["nharr"] = 8622,  
5949 ["nletrightarrow"] = 8622,  
5950 ["Lsh"] = 8624,  
5951 ["lsh"] = 8624,  
5952 ["Rsh"] = 8625,  
5953 ["rsh"] = 8625,  
5954 ["ldsh"] = 8626,  
5955 ["rdsh"] = 8627,  
5956 ["crarr"] = 8629,  
5957 ["cularr"] = 8630,  
5958 ["curvearrowleft"] = 8630,  
5959 ["curarr"] = 8631,  
5960 ["curvearrowright"] = 8631,  
5961 ["circlearrowleft"] = 8634,  
5962 ["olarr"] = 8634,  
5963 ["circlearrowright"] = 8635,  
5964 ["orarr"] = 8635,  
5965 ["LeftVector"] = 8636,  
5966 ["leftharpoonup"] = 8636,  
5967 ["lharu"] = 8636,  
5968 ["DownLeftVector"] = 8637,  
5969 ["leftharpoondown"] = 8637,  
5970 ["lhard"] = 8637,  
5971 ["RightUpVector"] = 8638,  
5972 ["uharr"] = 8638,  
5973 ["upharpoonright"] = 8638,  
5974 ["LeftUpVector"] = 8639,  
5975 ["uharl"] = 8639,  
5976 ["upharpoonleft"] = 8639,  
5977 ["RightVector"] = 8640,  
5978 ["rharu"] = 8640,  
5979 ["rightharpoonup"] = 8640,  
5980 ["DownRightVector"] = 8641,  
5981 ["rhard"] = 8641,  
5982 ["rightharpoondown"] = 8641,  
5983 ["RightDownVector"] = 8642,  
5984 ["dharr"] = 8642,  
5985 ["downharpoonright"] = 8642,  
5986 ["LeftDownVector"] = 8643,

5987 ["dharl"] = 8643,  
 5988 ["downharpoonleft"] = 8643,  
 5989 ["RightArrowLeftArrow"] = 8644,  
 5990 ["rightleftarrows"] = 8644,  
 5991 ["rlarr"] = 8644,  
 5992 ["UpArrowDownArrow"] = 8645,  
 5993 ["udarr"] = 8645,  
 5994 ["LeftArrowRightArrow"] = 8646,  
 5995 ["leftrightharpoons"] = 8646,  
 5996 ["lrarr"] = 8646,  
 5997 ["leftleftarrows"] = 8647,  
 5998 ["llarr"] = 8647,  
 5999 ["upuparrows"] = 8648,  
 6000 ["uuarr"] = 8648,  
 6001 ["rightrightarrows"] = 8649,  
 6002 ["rrarr"] = 8649,  
 6003 ["ddarr"] = 8650,  
 6004 ["downdownarrows"] = 8650,  
 6005 ["ReverseEquilibrium"] = 8651,  
 6006 ["leftrightharpoons"] = 8651,  
 6007 ["lrhar"] = 8651,  
 6008 ["Equilibrium"] = 8652,  
 6009 ["rightleftharpoons"] = 8652,  
 6010 ["rlhar"] = 8652,  
 6011 ["nLeftarrow"] = 8653,  
 6012 ["nLArr"] = 8653,  
 6013 ["nLeftrightarrow"] = 8654,  
 6014 ["nhArr"] = 8654,  
 6015 ["nRrightarrow"] = 8655,  
 6016 ["nrArr"] = 8655,  
 6017 ["DoubleLeftArrow"] = 8656,  
 6018 ["Leftarrow"] = 8656,  
 6019 ["lArr"] = 8656,  
 6020 ["DoubleUpArrow"] = 8657,  
 6021 ["Uparrow"] = 8657,  
 6022 ["uArr"] = 8657,  
 6023 ["DoubleRightArrow"] = 8658,  
 6024 ["Implies"] = 8658,  
 6025 ["Rrightarrow"] = 8658,  
 6026 ["rArr"] = 8658,  
 6027 ["DoubleDownArrow"] = 8659,  
 6028 ["Downarrow"] = 8659,  
 6029 ["dArr"] = 8659,  
 6030 ["DoubleLeftRightArrow"] = 8660,  
 6031 ["Leftrightarrow"] = 8660,  
 6032 ["hArr"] = 8660,  
 6033 ["iff"] = 8660,



6034 ["DoubleUpDownArrow"] = 8661,  
6035 ["Updownarrow"] = 8661,  
6036 ["vArr"] = 8661,  
6037 ["nwArr"] = 8662,  
6038 ["neArr"] = 8663,  
6039 ["seArr"] = 8664,  
6040 ["swArr"] = 8665,  
6041 ["Lleftarrow"] = 8666,  
6042 ["lAarr"] = 8666,  
6043 ["Rrightarrow"] = 8667,  
6044 ["rAarr"] = 8667,  
6045 ["zigrarr"] = 8669,  
6046 ["LeftArrowBar"] = 8676,  
6047 ["larrb"] = 8676,  
6048 ["RightArrowBar"] = 8677,  
6049 ["rarrb"] = 8677,  
6050 ["DownArrowUpArrow"] = 8693,  
6051 ["duarr"] = 8693,  
6052 ["loarr"] = 8701,  
6053 ["roarr"] = 8702,  
6054 ["hoarr"] = 8703,  
6055 ["ForAll"] = 8704,  
6056 ["forall"] = 8704,  
6057 ["comp"] = 8705,  
6058 ["complement"] = 8705,  
6059 ["PartialD"] = 8706,  
6060 ["npart"] = {8706, 824},  
6061 ["part"] = 8706,  
6062 ["Exists"] = 8707,  
6063 ["exist"] = 8707,  
6064 ["NotExists"] = 8708,  
6065 ["nexist"] = 8708,  
6066 ["nexists"] = 8708,  
6067 ["empty"] = 8709,  
6068 ["emptyset"] = 8709,  
6069 ["emptyv"] = 8709,  
6070 ["varnothing"] = 8709,  
6071 ["Del"] = 8711,  
6072 ["nabla"] = 8711,  
6073 ["Element"] = 8712,  
6074 ["in"] = 8712,  
6075 ["isin"] = 8712,  
6076 ["isinv"] = 8712,  
6077 ["NotElement"] = 8713,  
6078 ["notin"] = 8713,  
6079 ["notinva"] = 8713,  
6080 ["ReverseElement"] = 8715,

6081 ["SuchThat"] = 8715,  
6082 ["ni"] = 8715,  
6083 ["niv"] = 8715,  
6084 ["NotReverseElement"] = 8716,  
6085 ["notni"] = 8716,  
6086 ["notniva"] = 8716,  
6087 ["Product"] = 8719,  
6088 ["prod"] = 8719,  
6089 ["Coproduct"] = 8720,  
6090 ["coprod"] = 8720,  
6091 ["Sum"] = 8721,  
6092 ["sum"] = 8721,  
6093 ["minus"] = 8722,  
6094 ["MinusPlus"] = 8723,  
6095 ["mnplus"] = 8723,  
6096 ["mp"] = 8723,  
6097 ["dotplus"] = 8724,  
6098 ["plusdo"] = 8724,  
6099 ["Backslash"] = 8726,  
6100 ["setminus"] = 8726,  
6101 ["setmn"] = 8726,  
6102 ["smallsetminus"] = 8726,  
6103 ["ssetmn"] = 8726,  
6104 ["lowast"] = 8727,  
6105 ["SmallCircle"] = 8728,  
6106 ["compfn"] = 8728,  
6107 ["Sqrt"] = 8730,  
6108 ["radic"] = 8730,  
6109 ["Proportional"] = 8733,  
6110 ["prop"] = 8733,  
6111 ["propto"] = 8733,  
6112 ["varpropto"] = 8733,  
6113 ["vprop"] = 8733,  
6114 ["infin"] = 8734,  
6115 ["angrt"] = 8735,  
6116 ["ang"] = 8736,  
6117 ["angle"] = 8736,  
6118 ["nang"] = {8736, 8402},  
6119 ["angmsd"] = 8737,  
6120 ["measuredangle"] = 8737,  
6121 ["angsph"] = 8738,  
6122 ["VerticalBar"] = 8739,  
6123 ["mid"] = 8739,  
6124 ["shortmid"] = 8739,  
6125 ["smid"] = 8739,  
6126 ["NotVerticalBar"] = 8740,  
6127 ["nmid"] = 8740,

6128 ["nshortmid"] = 8740,  
6129 ["nsmid"] = 8740,  
6130 ["DoubleVerticalBar"] = 8741,  
6131 ["par"] = 8741,  
6132 ["parallel"] = 8741,  
6133 ["shortparallel"] = 8741,  
6134 ["spar"] = 8741,  
6135 ["NotDoubleVerticalBar"] = 8742,  
6136 ["npar"] = 8742,  
6137 ["nparallel"] = 8742,  
6138 ["nshortparallel"] = 8742,  
6139 ["nspar"] = 8742,  
6140 ["and"] = 8743,  
6141 ["wedge"] = 8743,  
6142 ["or"] = 8744,  
6143 ["vee"] = 8744,  
6144 ["cap"] = 8745,  
6145 ["caps"] = {8745, 65024},  
6146 ["cup"] = 8746,  
6147 ["cups"] = {8746, 65024},  
6148 ["Integral"] = 8747,  
6149 ["int"] = 8747,  
6150 ["Int"] = 8748,  
6151 ["iiint"] = 8749,  
6152 ["tint"] = 8749,  
6153 ["ContourIntegral"] = 8750,  
6154 ["conint"] = 8750,  
6155 ["oint"] = 8750,  
6156 ["Conint"] = 8751,  
6157 ["DoubleContourIntegral"] = 8751,  
6158 ["Cconint"] = 8752,  
6159 ["cwint"] = 8753,  
6160 ["ClockwiseContourIntegral"] = 8754,  
6161 ["cwconint"] = 8754,  
6162 ["CounterClockwiseContourIntegral"] = 8755,  
6163 ["awconint"] = 8755,  
6164 ["Therefore"] = 8756,  
6165 ["there4"] = 8756,  
6166 ["therefore"] = 8756,  
6167 ["Because"] = 8757,  
6168 ["because"] = 8757,  
6169 ["because"] = 8757,  
6170 ["ratio"] = 8758,  
6171 ["Colon"] = 8759,  
6172 ["Proportion"] = 8759,  
6173 ["dotminus"] = 8760,  
6174 ["minusd"] = 8760,

6175 ["mDDot"] = 8762,  
6176 ["homtht"] = 8763,  
6177 ["Tilde"] = 8764,  
6178 ["nvsim"] = {8764, 8402},  
6179 ["sim"] = 8764,  
6180 ["thicksim"] = 8764,  
6181 ["thksim"] = 8764,  
6182 ["backsim"] = 8765,  
6183 ["bsim"] = 8765,  
6184 ["race"] = {8765, 817},  
6185 ["ac"] = 8766,  
6186 ["acE"] = {8766, 819},  
6187 ["mstpos"] = 8766,  
6188 ["acd"] = 8767,  
6189 ["VerticalTilde"] = 8768,  
6190 ["wr"] = 8768,  
6191 ["wreath"] = 8768,  
6192 ["NotTilde"] = 8769,  
6193 ["nsim"] = 8769,  
6194 ["EqualTilde"] = 8770,  
6195 ["NotEqualTilde"] = {8770, 824},  
6196 ["eqsim"] = 8770,  
6197 ["esim"] = 8770,  
6198 ["nesim"] = {8770, 824},  
6199 ["TildeEqual"] = 8771,  
6200 ["sime"] = 8771,  
6201 ["simeq"] = 8771,  
6202 ["NotTildeEqual"] = 8772,  
6203 ["nsime"] = 8772,  
6204 ["nsimeq"] = 8772,  
6205 ["TildeFullEqual"] = 8773,  
6206 ["cong"] = 8773,  
6207 ["simne"] = 8774,  
6208 ["NotTildeFullEqual"] = 8775,  
6209 ["ncong"] = 8775,  
6210 ["TildeTilde"] = 8776,  
6211 ["ap"] = 8776,  
6212 ["approx"] = 8776,  
6213 ["asymp"] = 8776,  
6214 ["thickapprox"] = 8776,  
6215 ["thkap"] = 8776,  
6216 ["NotTildeTilde"] = 8777,  
6217 ["nap"] = 8777,  
6218 ["napprox"] = 8777,  
6219 ["ape"] = 8778,  
6220 ["approxpeq"] = 8778,  
6221 ["apid"] = 8779,

6222 ["napid"] = {8779, 824},  
6223 ["backcong"] = 8780,  
6224 ["bcong"] = 8780,  
6225 ["CupCap"] = 8781,  
6226 ["asympeq"] = 8781,  
6227 ["nvap"] = {8781, 8402},  
6228 ["Bumpeq"] = 8782,  
6229 ["HumpDownHump"] = 8782,  
6230 ["NotHumpDownHump"] = {8782, 824},  
6231 ["bump"] = 8782,  
6232 ["nbump"] = {8782, 824},  
6233 ["HumpEqual"] = 8783,  
6234 ["NotHumpEqual"] = {8783, 824},  
6235 ["bumpe"] = 8783,  
6236 ["bumpeq"] = 8783,  
6237 ["nbumpe"] = {8783, 824},  
6238 ["DotEqual"] = 8784,  
6239 ["doteq"] = 8784,  
6240 ["esdot"] = 8784,  
6241 ["nedot"] = {8784, 824},  
6242 ["doteqdot"] = 8785,  
6243 ["eDot"] = 8785,  
6244 ["efDot"] = 8786,  
6245 ["fallingdotseq"] = 8786,  
6246 ["erDot"] = 8787,  
6247 ["risingdotseq"] = 8787,  
6248 ["Assign"] = 8788,  
6249 ["colone"] = 8788,  
6250 ["coloneq"] = 8788,  
6251 ["ecolon"] = 8789,  
6252 ["eqcolon"] = 8789,  
6253 ["ecir"] = 8790,  
6254 ["eqcirc"] = 8790,  
6255 ["circeq"] = 8791,  
6256 ["cire"] = 8791,  
6257 ["wedgeq"] = 8793,  
6258 ["veeeq"] = 8794,  
6259 ["triangleq"] = 8796,  
6260 ["trie"] = 8796,  
6261 ["equest"] = 8799,  
6262 ["questeq"] = 8799,  
6263 ["NotEqual"] = 8800,  
6264 ["ne"] = 8800,  
6265 ["Congruent"] = 8801,  
6266 ["bnequiv"] = {8801, 8421},  
6267 ["equiv"] = 8801,  
6268 ["NotCongruent"] = 8802,

6269 ["nequiv"] = 8802,  
6270 ["le"] = 8804,  
6271 ["leq"] = 8804,  
6272 ["nvle"] = {8804, 8402},  
6273 ["GreaterEqual"] = 8805,  
6274 ["ge"] = 8805,  
6275 ["geq"] = 8805,  
6276 ["nvge"] = {8805, 8402},  
6277 ["LessFullEqual"] = 8806,  
6278 ["lE"] = 8806,  
6279 ["leqq"] = 8806,  
6280 ["nlE"] = {8806, 824},  
6281 ["nleqq"] = {8806, 824},  
6282 ["GreaterFullEqual"] = 8807,  
6283 ["NotGreaterFullEqual"] = {8807, 824},  
6284 ["gE"] = 8807,  
6285 ["geqq"] = 8807,  
6286 ["ngE"] = {8807, 824},  
6287 ["ngeqq"] = {8807, 824},  
6288 ["lnE"] = 8808,  
6289 ["lneqq"] = 8808,  
6290 ["lvertneqq"] = {8808, 65024},  
6291 ["lvnE"] = {8808, 65024},  
6292 ["gnE"] = 8809,  
6293 ["gneqq"] = 8809,  
6294 ["gvertneqq"] = {8809, 65024},  
6295 ["gvnE"] = {8809, 65024},  
6296 ["Lt"] = 8810,  
6297 ["NestedLessLess"] = 8810,  
6298 ["NotLessLess"] = {8810, 824},  
6299 ["ll"] = 8810,  
6300 ["nLt"] = {8810, 8402},  
6301 ["nLtv"] = {8810, 824},  
6302 ["Gt"] = 8811,  
6303 ["NestedGreaterGreater"] = 8811,  
6304 ["NotGreaterGreater"] = {8811, 824},  
6305 ["gg"] = 8811,  
6306 ["nGt"] = {8811, 8402},  
6307 ["nGtv"] = {8811, 824},  
6308 ["between"] = 8812,  
6309 ["twixt"] = 8812,  
6310 ["NotCupCap"] = 8813,  
6311 ["NotLess"] = 8814,  
6312 ["nless"] = 8814,  
6313 ["nlt"] = 8814,  
6314 ["NotGreater"] = 8815,  
6315 ["ngt"] = 8815,

6316 ["ngtr"] = 8815,  
6317 ["NotLessEqual"] = 8816,  
6318 ["nle"] = 8816,  
6319 ["nleq"] = 8816,  
6320 ["NotGreaterEqual"] = 8817,  
6321 ["nge"] = 8817,  
6322 ["ngeq"] = 8817,  
6323 ["LessTilde"] = 8818,  
6324 ["lesssim"] = 8818,  
6325 ["lsim"] = 8818,  
6326 ["GreaterTilde"] = 8819,  
6327 ["gsim"] = 8819,  
6328 ["gtrsim"] = 8819,  
6329 ["NotLessTilde"] = 8820,  
6330 ["nlsim"] = 8820,  
6331 ["NotGreaterTilde"] = 8821,  
6332 ["ngsim"] = 8821,  
6333 ["LessGreater"] = 8822,  
6334 ["lessgtr"] = 8822,  
6335 ["lg"] = 8822,  
6336 ["GreaterLess"] = 8823,  
6337 ["gl"] = 8823,  
6338 ["gtrless"] = 8823,  
6339 ["NotLessGreater"] = 8824,  
6340 ["ntlg"] = 8824,  
6341 ["NotGreaterLess"] = 8825,  
6342 ["ntgl"] = 8825,  
6343 ["Precedes"] = 8826,  
6344 ["pr"] = 8826,  
6345 ["prec"] = 8826,  
6346 ["Succeeds"] = 8827,  
6347 ["sc"] = 8827,  
6348 ["succ"] = 8827,  
6349 ["PrecedesSlantEqual"] = 8828,  
6350 ["prcue"] = 8828,  
6351 ["preccurlyeq"] = 8828,  
6352 ["SucceedsSlantEqual"] = 8829,  
6353 ["sccue"] = 8829,  
6354 ["succcurlyeq"] = 8829,  
6355 ["PrecedesTilde"] = 8830,  
6356 ["precsim"] = 8830,  
6357 ["prsim"] = 8830,  
6358 ["NotSucceedsTilde"] = {8831, 824},  
6359 ["SucceedsTilde"] = 8831,  
6360 ["scsim"] = 8831,  
6361 ["succsim"] = 8831,  
6362 ["NotPrecedes"] = 8832,

6363 ["npr"] = 8832,  
6364 ["nprec"] = 8832,  
6365 ["NotSucceeds"] = 8833,  
6366 ["nsc"] = 8833,  
6367 ["nsucc"] = 8833,  
6368 ["NotSubset"] = {8834, 8402},  
6369 ["nsubset"] = {8834, 8402},  
6370 ["sub"] = 8834,  
6371 ["subset"] = 8834,  
6372 ["vnsup"] = {8834, 8402},  
6373 ["NotSuperset"] = {8835, 8402},  
6374 ["Superset"] = 8835,  
6375 ["nsupset"] = {8835, 8402},  
6376 ["sup"] = 8835,  
6377 ["supset"] = 8835,  
6378 ["vnsup"] = {8835, 8402},  
6379 ["nsub"] = 8836,  
6380 ["nsup"] = 8837,  
6381 ["SubsetEqual"] = 8838,  
6382 ["sube"] = 8838,  
6383 ["subseteq"] = 8838,  
6384 ["SupersetEqual"] = 8839,  
6385 ["supe"] = 8839,  
6386 ["supseteq"] = 8839,  
6387 ["NotSubsetEqual"] = 8840,  
6388 ["nsube"] = 8840,  
6389 ["nsubseteq"] = 8840,  
6390 ["NotSupersetEqual"] = 8841,  
6391 ["nsupe"] = 8841,  
6392 ["nsupseteq"] = 8841,  
6393 ["subne"] = 8842,  
6394 ["subsetneq"] = 8842,  
6395 ["varsubsetneq"] = {8842, 65024},  
6396 ["vsubne"] = {8842, 65024},  
6397 ["supne"] = 8843,  
6398 ["supsetneq"] = 8843,  
6399 ["varsupsetneq"] = {8843, 65024},  
6400 ["vsupne"] = {8843, 65024},  
6401 ["cupdot"] = 8845,  
6402 ["UnionPlus"] = 8846,  
6403 ["uplus"] = 8846,  
6404 ["NotSquareSubset"] = {8847, 824},  
6405 ["SquareSubset"] = 8847,  
6406 ["sqsub"] = 8847,  
6407 ["sqsubset"] = 8847,  
6408 ["NotSquareSuperset"] = {8848, 824},  
6409 ["SquareSuperset"] = 8848,



6410 ["sqsup"] = 8848,  
6411 ["sqsupset"] = 8848,  
6412 ["SquareSubsetEqual"] = 8849,  
6413 ["sqsube"] = 8849,  
6414 ["sqsubseteq"] = 8849,  
6415 ["SquareSupersetEqual"] = 8850,  
6416 ["sqsupe"] = 8850,  
6417 ["sqsupseteq"] = 8850,  
6418 ["SquareIntersection"] = 8851,  
6419 ["sqcap"] = 8851,  
6420 ["sqcaps"] = {8851, 65024},  
6421 ["SquareUnion"] = 8852,  
6422 ["sqcup"] = 8852,  
6423 ["sqcups"] = {8852, 65024},  
6424 ["CirclePlus"] = 8853,  
6425 ["oplus"] = 8853,  
6426 ["CircleMinus"] = 8854,  
6427 ["ominus"] = 8854,  
6428 ["CircleTimes"] = 8855,  
6429 ["otimes"] = 8855,  
6430 ["osol"] = 8856,  
6431 ["CircleDot"] = 8857,  
6432 ["odot"] = 8857,  
6433 ["circledcirc"] = 8858,  
6434 ["ocir"] = 8858,  
6435 ["circledast"] = 8859,  
6436 ["oast"] = 8859,  
6437 ["circleddash"] = 8861,  
6438 ["odash"] = 8861,  
6439 ["boxplus"] = 8862,  
6440 ["plusb"] = 8862,  
6441 ["boxminus"] = 8863,  
6442 ["minusb"] = 8863,  
6443 ["boxtimes"] = 8864,  
6444 ["timesb"] = 8864,  
6445 ["dotsquare"] = 8865,  
6446 ["sdotb"] = 8865,  
6447 ["RightTee"] = 8866,  
6448 ["vdash"] = 8866,  
6449 ["LeftTee"] = 8867,  
6450 ["dashv"] = 8867,  
6451 ["DownTee"] = 8868,  
6452 ["top"] = 8868,  
6453 ["UpTee"] = 8869,  
6454 ["bot"] = 8869,  
6455 ["bottom"] = 8869,  
6456 ["perp"] = 8869,

6457 ["models"] = 8871,  
6458 ["DoubleRightTee"] = 8872,  
6459 ["vDash"] = 8872,  
6460 ["Vdash"] = 8873,  
6461 ["Vvdash"] = 8874,  
6462 ["VDash"] = 8875,  
6463 ["nvdash"] = 8876,  
6464 ["nvDash"] = 8877,  
6465 ["nVdash"] = 8878,  
6466 ["nVDash"] = 8879,  
6467 ["prurel"] = 8880,  
6468 ["LeftTriangle"] = 8882,  
6469 ["vartriangleleft"] = 8882,  
6470 ["vltri"] = 8882,  
6471 ["RightTriangle"] = 8883,  
6472 ["vartriangleright"] = 8883,  
6473 ["vrtri"] = 8883,  
6474 ["LeftTriangleEqual"] = 8884,  
6475 ["ltrie"] = 8884,  
6476 ["nvltrie"] = {8884, 8402},  
6477 ["trianglelefteq"] = 8884,  
6478 ["RightTriangleEqual"] = 8885,  
6479 ["nvrtrie"] = {8885, 8402},  
6480 ["rtrie"] = 8885,  
6481 ["trianglerighteq"] = 8885,  
6482 ["origof"] = 8886,  
6483 ["imof"] = 8887,  
6484 ["multimap"] = 8888,  
6485 ["mumap"] = 8888,  
6486 ["hercon"] = 8889,  
6487 ["intcal"] = 8890,  
6488 ["intercal"] = 8890,  
6489 ["veebar"] = 8891,  
6490 ["barvee"] = 8893,  
6491 ["angrtvb"] = 8894,  
6492 ["lrtri"] = 8895,  
6493 ["Wedge"] = 8896,  
6494 ["bigwedge"] = 8896,  
6495 ["xwedge"] = 8896,  
6496 ["Vee"] = 8897,  
6497 ["bigvee"] = 8897,  
6498 ["xvee"] = 8897,  
6499 ["Intersection"] = 8898,  
6500 ["bigcap"] = 8898,  
6501 ["xcap"] = 8898,  
6502 ["Union"] = 8899,  
6503 ["bigcup"] = 8899,

6504 ["xcup"] = 8899,  
6505 ["Diamond"] = 8900,  
6506 ["diam"] = 8900,  
6507 ["diamond"] = 8900,  
6508 ["sdot"] = 8901,  
6509 ["Star"] = 8902,  
6510 ["sstarf"] = 8902,  
6511 ["divideontimes"] = 8903,  
6512 ["divonx"] = 8903,  
6513 ["bowtie"] = 8904,  
6514 ["ltimes"] = 8905,  
6515 ["rtimes"] = 8906,  
6516 ["leftthreetimes"] = 8907,  
6517 ["lthree"] = 8907,  
6518 ["rightthreetimes"] = 8908,  
6519 ["rthree"] = 8908,  
6520 ["backsimeq"] = 8909,  
6521 ["bsime"] = 8909,  
6522 ["curlyvee"] = 8910,  
6523 ["cuvee"] = 8910,  
6524 ["curlywedge"] = 8911,  
6525 ["cuwed"] = 8911,  
6526 ["Sub"] = 8912,  
6527 ["Subset"] = 8912,  
6528 ["Sup"] = 8913,  
6529 ["Supset"] = 8913,  
6530 ["Cap"] = 8914,  
6531 ["Cup"] = 8915,  
6532 ["fork"] = 8916,  
6533 ["pitchfork"] = 8916,  
6534 ["epar"] = 8917,  
6535 ["lessdot"] = 8918,  
6536 ["ltdot"] = 8918,  
6537 ["gtdot"] = 8919,  
6538 ["gtrdot"] = 8919,  
6539 ["L1"] = 8920,  
6540 ["nL1"] = {8920, 824},  
6541 ["Gg"] = 8921,  
6542 ["ggg"] = 8921,  
6543 ["nGg"] = {8921, 824},  
6544 ["LessEqualGreater"] = 8922,  
6545 ["leg"] = 8922,  
6546 ["lesg"] = {8922, 65024},  
6547 ["lesseqgtr"] = 8922,  
6548 ["GreaterEqualLess"] = 8923,  
6549 ["gel"] = 8923,  
6550 ["gesl"] = {8923, 65024},

6551 ["gtreqless"] = 8923,  
6552 ["cuepr"] = 8926,  
6553 ["curlyeqprec"] = 8926,  
6554 ["cuesc"] = 8927,  
6555 ["curlyeqsucc"] = 8927,  
6556 ["NotPrecedesSlantEqual"] = 8928,  
6557 ["nprcue"] = 8928,  
6558 ["NotSucceedsSlantEqual"] = 8929,  
6559 ["nsccue"] = 8929,  
6560 ["NotSquareSubsetEqual"] = 8930,  
6561 ["nsqsube"] = 8930,  
6562 ["NotSquareSupersetEqual"] = 8931,  
6563 ["nsqsupe"] = 8931,  
6564 ["lnsim"] = 8934,  
6565 ["gnsim"] = 8935,  
6566 ["precnsim"] = 8936,  
6567 ["prnsim"] = 8936,  
6568 ["scnsim"] = 8937,  
6569 ["succnsim"] = 8937,  
6570 ["NotLeftTriangle"] = 8938,  
6571 ["nltri"] = 8938,  
6572 ["ntriangleleft"] = 8938,  
6573 ["NotRightTriangle"] = 8939,  
6574 ["nrtri"] = 8939,  
6575 ["ntriangleright"] = 8939,  
6576 ["NotLeftTriangleEqual"] = 8940,  
6577 ["nltrie"] = 8940,  
6578 ["ntrianglelefteq"] = 8940,  
6579 ["NotRightTriangleEqual"] = 8941,  
6580 ["nrtrie"] = 8941,  
6581 ["ntrianglerighteq"] = 8941,  
6582 ["vellip"] = 8942,  
6583 ["ctdot"] = 8943,  
6584 ["utdot"] = 8944,  
6585 ["dtdot"] = 8945,  
6586 ["disin"] = 8946,  
6587 ["isinsv"] = 8947,  
6588 ["isins"] = 8948,  
6589 ["isindot"] = 8949,  
6590 ["notinot"] = {8949, 824},  
6591 ["notinvc"] = 8950,  
6592 ["notinvb"] = 8951,  
6593 ["isinE"] = 8953,  
6594 ["notinE"] = {8953, 824},  
6595 ["nisd"] = 8954,  
6596 ["xnis"] = 8955,  
6597 ["nis"] = 8956,

6598 ["notnivc"] = 8957,  
6599 ["notnivb"] = 8958,  
6600 ["barwed"] = 8965,  
6601 ["barwedge"] = 8965,  
6602 ["Barwed"] = 8966,  
6603 ["doublebarwedge"] = 8966,  
6604 ["LeftCeiling"] = 8968,  
6605 ["lceil"] = 8968,  
6606 ["RightCeiling"] = 8969,  
6607 ["rceil"] = 8969,  
6608 ["LeftFloor"] = 8970,  
6609 ["lfloor"] = 8970,  
6610 ["RightFloor"] = 8971,  
6611 ["rfloor"] = 8971,  
6612 ["drcrop"] = 8972,  
6613 ["dlcrop"] = 8973,  
6614 ["urcrop"] = 8974,  
6615 ["ulcrop"] = 8975,  
6616 ["bnot"] = 8976,  
6617 ["proflin"] = 8978,  
6618 ["profsurf"] = 8979,  
6619 ["telrec"] = 8981,  
6620 ["target"] = 8982,  
6621 ["ulcorn"] = 8988,  
6622 ["ulcorner"] = 8988,  
6623 ["urcorn"] = 8989,  
6624 ["urcorner"] = 8989,  
6625 ["dlcorn"] = 8990,  
6626 ["llcorner"] = 8990,  
6627 ["drcorn"] = 8991,  
6628 ["lrcorn"] = 8991,  
6629 ["frown"] = 8994,  
6630 ["sfrown"] = 8994,  
6631 ["smile"] = 8995,  
6632 ["ssmile"] = 8995,  
6633 ["cylcty"] = 9005,  
6634 ["profalar"] = 9006,  
6635 ["topbot"] = 9014,  
6636 ["ovbar"] = 9021,  
6637 ["solbar"] = 9023,  
6638 ["angzarr"] = 9084,  
6639 ["lmoust"] = 9136,  
6640 ["lmoustache"] = 9136,  
6641 ["rmoust"] = 9137,  
6642 ["rmoustache"] = 9137,  
6643 ["OverBracket"] = 9140,  
6644 ["tbrk"] = 9140,

6645 ["UnderBracket"] = 9141,  
6646 ["bbrk"] = 9141,  
6647 ["bbrktbrk"] = 9142,  
6648 ["OverParenthesis"] = 9180,  
6649 ["UnderParenthesis"] = 9181,  
6650 ["OverBrace"] = 9182,  
6651 ["UnderBrace"] = 9183,  
6652 ["trpezium"] = 9186,  
6653 ["elinters"] = 9191,  
6654 ["blank"] = 9251,  
6655 ["circledS"] = 9416,  
6656 ["oS"] = 9416,  
6657 ["HorizontalLine"] = 9472,  
6658 ["boxh"] = 9472,  
6659 ["boxv"] = 9474,  
6660 ["boxdr"] = 9484,  
6661 ["boxdl"] = 9488,  
6662 ["boxur"] = 9492,  
6663 ["boxul"] = 9496,  
6664 ["boxvr"] = 9500,  
6665 ["boxvl"] = 9508,  
6666 ["boxhd"] = 9516,  
6667 ["boxhu"] = 9524,  
6668 ["boxvh"] = 9532,  
6669 ["boxH"] = 9552,  
6670 ["boxV"] = 9553,  
6671 ["boxdR"] = 9554,  
6672 ["boxDr"] = 9555,  
6673 ["boxDR"] = 9556,  
6674 ["boxdL"] = 9557,  
6675 ["boxDL"] = 9558,  
6676 ["boxDL"] = 9559,  
6677 ["boxuR"] = 9560,  
6678 ["boxUr"] = 9561,  
6679 ["boxUR"] = 9562,  
6680 ["boxuL"] = 9563,  
6681 ["boxUL"] = 9564,  
6682 ["boxUL"] = 9565,  
6683 ["boxvR"] = 9566,  
6684 ["boxVr"] = 9567,  
6685 ["boxVR"] = 9568,  
6686 ["boxvL"] = 9569,  
6687 ["boxVl"] = 9570,  
6688 ["boxVL"] = 9571,  
6689 ["boxHd"] = 9572,  
6690 ["boxhD"] = 9573,  
6691 ["boxHD"] = 9574,

6692 ["boxHu"] = 9575,  
6693 ["boxhU"] = 9576,  
6694 ["boxHU"] = 9577,  
6695 ["boxvH"] = 9578,  
6696 ["boxVh"] = 9579,  
6697 ["boxVH"] = 9580,  
6698 ["uhblk"] = 9600,  
6699 ["lhblk"] = 9604,  
6700 ["block"] = 9608,  
6701 ["blk14"] = 9617,  
6702 ["blk12"] = 9618,  
6703 ["blk34"] = 9619,  
6704 ["Square"] = 9633,  
6705 ["squ"] = 9633,  
6706 ["square"] = 9633,  
6707 ["FilledVerySmallSquare"] = 9642,  
6708 ["blacksquare"] = 9642,  
6709 ["squarf"] = 9642,  
6710 ["squf"] = 9642,  
6711 ["EmptyVerySmallSquare"] = 9643,  
6712 ["rect"] = 9645,  
6713 ["marker"] = 9646,  
6714 ["fltns"] = 9649,  
6715 ["bigtriangleup"] = 9651,  
6716 ["xutri"] = 9651,  
6717 ["blacktriangle"] = 9652,  
6718 ["utrif"] = 9652,  
6719 ["triangle"] = 9653,  
6720 ["utri"] = 9653,  
6721 ["blacktriangleright"] = 9656,  
6722 ["rtrif"] = 9656,  
6723 ["rtri"] = 9657,  
6724 ["triangleright"] = 9657,  
6725 ["bigtriangledown"] = 9661,  
6726 ["xdtri"] = 9661,  
6727 ["blacktriangledown"] = 9662,  
6728 ["dtrif"] = 9662,  
6729 ["dtri"] = 9663,  
6730 ["triangledown"] = 9663,  
6731 ["blacktriangleleft"] = 9666,  
6732 ["ltrif"] = 9666,  
6733 ["ltri"] = 9667,  
6734 ["triangleleft"] = 9667,  
6735 ["loz"] = 9674,  
6736 ["lozenge"] = 9674,  
6737 ["cir"] = 9675,  
6738 ["tridot"] = 9708,

6739 ["bigcirc"] = 9711,  
6740 ["xcirc"] = 9711,  
6741 ["ultri"] = 9720,  
6742 ["urtri"] = 9721,  
6743 ["lltri"] = 9722,  
6744 ["EmptySmallSquare"] = 9723,  
6745 ["FilledSmallSquare"] = 9724,  
6746 ["bigstar"] = 9733,  
6747 ["starf"] = 9733,  
6748 ["star"] = 9734,  
6749 ["phone"] = 9742,  
6750 ["female"] = 9792,  
6751 ["male"] = 9794,  
6752 ["spades"] = 9824,  
6753 ["spadesuit"] = 9824,  
6754 ["clubs"] = 9827,  
6755 ["clubsuit"] = 9827,  
6756 ["hearts"] = 9829,  
6757 ["heartsuit"] = 9829,  
6758 ["diamondsuit"] = 9830,  
6759 ["diams"] = 9830,  
6760 ["sung"] = 9834,  
6761 ["flat"] = 9837,  
6762 ["natur"] = 9838,  
6763 ["natural"] = 9838,  
6764 ["sharp"] = 9839,  
6765 ["check"] = 10003,  
6766 ["checkmark"] = 10003,  
6767 ["cross"] = 10007,  
6768 ["malt"] = 10016,  
6769 ["maltese"] = 10016,  
6770 ["sext"] = 10038,  
6771 ["VerticalSeparator"] = 10072,  
6772 ["lbrk"] = 10098,  
6773 ["rbrk"] = 10099,  
6774 ["bsolhsub"] = 10184,  
6775 ["suphsol"] = 10185,  
6776 ["LeftDoubleBracket"] = 10214,  
6777 ["lobrk"] = 10214,  
6778 ["RightDoubleBracket"] = 10215,  
6779 ["robrk"] = 10215,  
6780 ["LeftAngleBracket"] = 10216,  
6781 ["lang"] = 10216,  
6782 ["langle"] = 10216,  
6783 ["RightAngleBracket"] = 10217,  
6784 ["rang"] = 10217,  
6785 ["rangle"] = 10217,



6786 ["Lang"] = 10218,  
6787 ["Rang"] = 10219,  
6788 ["loang"] = 10220,  
6789 ["roang"] = 10221,  
6790 ["LongLeftArrow"] = 10229,  
6791 ["longleftarrow"] = 10229,  
6792 ["xlarr"] = 10229,  
6793 ["LongRightArrow"] = 10230,  
6794 ["longrightarrow"] = 10230,  
6795 ["xrarr"] = 10230,  
6796 ["LongLeftRightArrow"] = 10231,  
6797 ["longleftrightarrow"] = 10231,  
6798 ["xharr"] = 10231,  
6799 ["DoubleLongLeftArrow"] = 10232,  
6800 ["Longleftarrow"] = 10232,  
6801 ["xlArr"] = 10232,  
6802 ["DoubleLongRightArrow"] = 10233,  
6803 ["Longrightarrow"] = 10233,  
6804 ["xrArr"] = 10233,  
6805 ["DoubleLongLeftRightArrow"] = 10234,  
6806 ["Longleftrightarrow"] = 10234,  
6807 ["xhArr"] = 10234,  
6808 ["longmapsto"] = 10236,  
6809 ["xmap"] = 10236,  
6810 ["dzigrarr"] = 10239,  
6811 ["nvlArr"] = 10498,  
6812 ["nvrArr"] = 10499,  
6813 ["nvHarr"] = 10500,  
6814 ["Map"] = 10501,  
6815 ["lbarr"] = 10508,  
6816 ["bkarow"] = 10509,  
6817 ["rbarr"] = 10509,  
6818 ["lBarr"] = 10510,  
6819 ["dbkarow"] = 10511,  
6820 ["rBarr"] = 10511,  
6821 ["RBarr"] = 10512,  
6822 ["drbkarow"] = 10512,  
6823 ["DDottrahd"] = 10513,  
6824 ["UpArrowBar"] = 10514,  
6825 ["DownArrowBar"] = 10515,  
6826 ["Rarrtl"] = 10518,  
6827 ["latail"] = 10521,  
6828 ["ratail"] = 10522,  
6829 ["lAtail"] = 10523,  
6830 ["rAtail"] = 10524,  
6831 ["larrfs"] = 10525,  
6832 ["rarrfs"] = 10526,

6833 ["larrbfs"] = 10527,  
6834 ["rarrbfs"] = 10528,  
6835 ["nwarhk"] = 10531,  
6836 ["nearhk"] = 10532,  
6837 ["hksearow"] = 10533,  
6838 ["searhk"] = 10533,  
6839 ["hkswarow"] = 10534,  
6840 ["swarhk"] = 10534,  
6841 ["nwnear"] = 10535,  
6842 ["nesear"] = 10536,  
6843 ["toea"] = 10536,  
6844 ["seswar"] = 10537,  
6845 ["tosa"] = 10537,  
6846 ["swnwar"] = 10538,  
6847 ["nrarrc"] = {10547, 824},  
6848 ["rarrc"] = 10547,  
6849 ["cudarr"] = 10549,  
6850 ["ldca"] = 10550,  
6851 ["rdca"] = 10551,  
6852 ["cudarrl"] = 10552,  
6853 ["larrpl"] = 10553,  
6854 ["curarrm"] = 10556,  
6855 ["cularrp"] = 10557,  
6856 ["rarrpl"] = 10565,  
6857 ["harrcir"] = 10568,  
6858 ["Uarrocir"] = 10569,  
6859 ["lurdshar"] = 10570,  
6860 ["ldrushar"] = 10571,  
6861 ["LeftRightVector"] = 10574,  
6862 ["RightUpDownVector"] = 10575,  
6863 ["DownLeftRightVector"] = 10576,  
6864 ["LeftUpDownVector"] = 10577,  
6865 ["LeftVectorBar"] = 10578,  
6866 ["RightVectorBar"] = 10579,  
6867 ["RightUpVectorBar"] = 10580,  
6868 ["RightDownVectorBar"] = 10581,  
6869 ["DownLeftVectorBar"] = 10582,  
6870 ["DownRightVectorBar"] = 10583,  
6871 ["LeftUpVectorBar"] = 10584,  
6872 ["LeftDownVectorBar"] = 10585,  
6873 ["LeftTeeVector"] = 10586,  
6874 ["RightTeeVector"] = 10587,  
6875 ["RightUpTeeVector"] = 10588,  
6876 ["RightDownTeeVector"] = 10589,  
6877 ["DownLeftTeeVector"] = 10590,  
6878 ["DownRightTeeVector"] = 10591,  
6879 ["LeftUpTeeVector"] = 10592,

6880 ["LeftDownTeeVector"] = 10593,  
6881 ["lHar"] = 10594,  
6882 ["uHar"] = 10595,  
6883 ["rHar"] = 10596,  
6884 ["dHar"] = 10597,  
6885 ["luruhar"] = 10598,  
6886 ["ldrdhar"] = 10599,  
6887 ["ruluhar"] = 10600,  
6888 ["rdldhar"] = 10601,  
6889 ["lharul"] = 10602,  
6890 ["llhard"] = 10603,  
6891 ["rharul"] = 10604,  
6892 ["lrhard"] = 10605,  
6893 ["UpEquilibrium"] = 10606,  
6894 ["udhar"] = 10606,  
6895 ["ReverseUpEquilibrium"] = 10607,  
6896 ["duhar"] = 10607,  
6897 ["RoundImplies"] = 10608,  
6898 ["erarr"] = 10609,  
6899 ["simrarr"] = 10610,  
6900 ["larrsim"] = 10611,  
6901 ["rarrsim"] = 10612,  
6902 ["rarrap"] = 10613,  
6903 ["ltlarr"] = 10614,  
6904 ["gtrarr"] = 10616,  
6905 ["subrarr"] = 10617,  
6906 ["suplarr"] = 10619,  
6907 ["lfisht"] = 10620,  
6908 ["rfisht"] = 10621,  
6909 ["ufisht"] = 10622,  
6910 ["dfisht"] = 10623,  
6911 ["lopar"] = 10629,  
6912 ["ropar"] = 10630,  
6913 ["lbrke"] = 10635,  
6914 ["rbrke"] = 10636,  
6915 ["lbrkslu"] = 10637,  
6916 ["rbrksld"] = 10638,  
6917 ["lbrksld"] = 10639,  
6918 ["rbrkslu"] = 10640,  
6919 ["langd"] = 10641,  
6920 ["rangd"] = 10642,  
6921 ["lparlt"] = 10643,  
6922 ["rpargt"] = 10644,  
6923 ["gtlPar"] = 10645,  
6924 ["ltrPar"] = 10646,  
6925 ["vzigzag"] = 10650,  
6926 ["vangrt"] = 10652,

6927 ["angrtvbd"] = 10653,  
6928 ["ange"] = 10660,  
6929 ["range"] = 10661,  
6930 ["dwangle"] = 10662,  
6931 ["uwangle"] = 10663,  
6932 ["angmsdaa"] = 10664,  
6933 ["angmsdab"] = 10665,  
6934 ["angmsdac"] = 10666,  
6935 ["angmsdad"] = 10667,  
6936 ["angmsdae"] = 10668,  
6937 ["angmsdaf"] = 10669,  
6938 ["angmsdag"] = 10670,  
6939 ["angmsdah"] = 10671,  
6940 ["bemptyv"] = 10672,  
6941 ["demptyv"] = 10673,  
6942 ["cemptyv"] = 10674,  
6943 ["raemptyv"] = 10675,  
6944 ["laemptyv"] = 10676,  
6945 ["ohbar"] = 10677,  
6946 ["omid"] = 10678,  
6947 ["opar"] = 10679,  
6948 ["operp"] = 10681,  
6949 ["olcross"] = 10683,  
6950 ["odsold"] = 10684,  
6951 ["olcir"] = 10686,  
6952 ["ofcir"] = 10687,  
6953 ["olt"] = 10688,  
6954 ["ogt"] = 10689,  
6955 ["cirscir"] = 10690,  
6956 ["cirE"] = 10691,  
6957 ["solb"] = 10692,  
6958 ["bsolb"] = 10693,  
6959 ["boxbox"] = 10697,  
6960 ["trisb"] = 10701,  
6961 ["rtriltri"] = 10702,  
6962 ["LeftTriangleBar"] = 10703,  
6963 ["NotLeftTriangleBar"] = {10703, 824},  
6964 ["NotRightTriangleBar"] = {10704, 824},  
6965 ["RightTriangleBar"] = 10704,  
6966 ["iinfin"] = 10716,  
6967 ["infintie"] = 10717,  
6968 ["nvinfin"] = 10718,  
6969 ["eparsl"] = 10723,  
6970 ["smeparsl"] = 10724,  
6971 ["eqvparsl"] = 10725,  
6972 ["blacklozenge"] = 10731,  
6973 ["lozf"] = 10731,

6974 ["RuleDelayed"] = 10740,  
6975 ["dsol"] = 10742,  
6976 ["bigodot"] = 10752,  
6977 ["xodot"] = 10752,  
6978 ["bigoplus"] = 10753,  
6979 ["xoplus"] = 10753,  
6980 ["bigotimes"] = 10754,  
6981 ["xotime"] = 10754,  
6982 ["biguplus"] = 10756,  
6983 ["xuplus"] = 10756,  
6984 ["bigsqcup"] = 10758,  
6985 ["xsqlcup"] = 10758,  
6986 ["iiiint"] = 10764,  
6987 ["qint"] = 10764,  
6988 ["fpartint"] = 10765,  
6989 ["cirfnint"] = 10768,  
6990 ["awint"] = 10769,  
6991 ["rppolint"] = 10770,  
6992 ["scpolint"] = 10771,  
6993 ["npolint"] = 10772,  
6994 ["pointint"] = 10773,  
6995 ["quatint"] = 10774,  
6996 ["intlarhk"] = 10775,  
6997 ["pluscir"] = 10786,  
6998 ["plusacir"] = 10787,  
6999 ["simplus"] = 10788,  
7000 ["plusdu"] = 10789,  
7001 ["plussim"] = 10790,  
7002 ["plustwo"] = 10791,  
7003 ["mcomma"] = 10793,  
7004 ["minusdu"] = 10794,  
7005 ["loplus"] = 10797,  
7006 ["roplus"] = 10798,  
7007 ["Cross"] = 10799,  
7008 ["timesd"] = 10800,  
7009 ["timesbar"] = 10801,  
7010 ["smashp"] = 10803,  
7011 ["lotimes"] = 10804,  
7012 ["rotimes"] = 10805,  
7013 ["otimesas"] = 10806,  
7014 ["Otimes"] = 10807,  
7015 ["odiv"] = 10808,  
7016 ["triplus"] = 10809,  
7017 ["triminus"] = 10810,  
7018 ["tritime"] = 10811,  
7019 ["intprod"] = 10812,  
7020 ["iproduct"] = 10812,

7021 ["amalg"] = 10815,  
7022 ["capdot"] = 10816,  
7023 ["ncup"] = 10818,  
7024 ["ncap"] = 10819,  
7025 ["capand"] = 10820,  
7026 ["cupor"] = 10821,  
7027 ["cupcap"] = 10822,  
7028 ["capcup"] = 10823,  
7029 ["cupbrcap"] = 10824,  
7030 ["capbrcup"] = 10825,  
7031 ["cupcup"] = 10826,  
7032 ["capcap"] = 10827,  
7033 ["ccups"] = 10828,  
7034 ["ccaps"] = 10829,  
7035 ["ccupssm"] = 10832,  
7036 ["And"] = 10835,  
7037 ["Or"] = 10836,  
7038 ["andand"] = 10837,  
7039 ["oror"] = 10838,  
7040 ["orslope"] = 10839,  
7041 ["andslope"] = 10840,  
7042 ["andv"] = 10842,  
7043 ["orv"] = 10843,  
7044 ["andd"] = 10844,  
7045 ["ord"] = 10845,  
7046 ["wedbar"] = 10847,  
7047 ["sdote"] = 10854,  
7048 ["simdot"] = 10858,  
7049 ["congdote"] = 10861,  
7050 ["ncongdote"] = {10861, 824},  
7051 ["easter"] = 10862,  
7052 ["apacir"] = 10863,  
7053 ["apE"] = 10864,  
7054 ["napE"] = {10864, 824},  
7055 ["eplus"] = 10865,  
7056 ["pluse"] = 10866,  
7057 ["Esim"] = 10867,  
7058 ["Colone"] = 10868,  
7059 ["Equal"] = 10869,  
7060 ["ddotseq"] = 10871,  
7061 ["eDDot"] = 10871,  
7062 ["equivDD"] = 10872,  
7063 ["ltcir"] = 10873,  
7064 ["gtcir"] = 10874,  
7065 ["ltquest"] = 10875,  
7066 ["gtquest"] = 10876,  
7067 ["LessSlantEqual"] = 10877,

7068 ["NotLessSlantEqual"] = {10877, 824},  
7069 ["leqslant"] = 10877,  
7070 ["les"] = 10877,  
7071 ["nleqslant"] = {10877, 824},  
7072 ["nles"] = {10877, 824},  
7073 ["GreaterSlantEqual"] = 10878,  
7074 ["NotGreaterSlantEqual"] = {10878, 824},  
7075 ["geqslant"] = 10878,  
7076 ["ges"] = 10878,  
7077 ["ngeqslant"] = {10878, 824},  
7078 ["nges"] = {10878, 824},  
7079 ["lesdot"] = 10879,  
7080 ["gesdot"] = 10880,  
7081 ["lesdoto"] = 10881,  
7082 ["gesdoto"] = 10882,  
7083 ["lesdotor"] = 10883,  
7084 ["gesdoto1"] = 10884,  
7085 ["lap"] = 10885,  
7086 ["lessapprox"] = 10885,  
7087 ["gap"] = 10886,  
7088 ["gtrapprox"] = 10886,  
7089 ["lne"] = 10887,  
7090 ["lneq"] = 10887,  
7091 ["gne"] = 10888,  
7092 ["gneq"] = 10888,  
7093 ["lnap"] = 10889,  
7094 ["lnapprox"] = 10889,  
7095 ["gnap"] = 10890,  
7096 ["gnapprox"] = 10890,  
7097 ["lEg"] = 10891,  
7098 ["lesseqqgtr"] = 10891,  
7099 ["gE1"] = 10892,  
7100 ["gtreqqlless"] = 10892,  
7101 ["lsime"] = 10893,  
7102 ["gsime"] = 10894,  
7103 ["lsimg"] = 10895,  
7104 ["gsim1"] = 10896,  
7105 ["lgE"] = 10897,  
7106 ["glE"] = 10898,  
7107 ["lesges"] = 10899,  
7108 ["gesles"] = 10900,  
7109 ["els"] = 10901,  
7110 ["eqslantless"] = 10901,  
7111 ["egs"] = 10902,  
7112 ["eqslantgtr"] = 10902,  
7113 ["elsdot"] = 10903,  
7114 ["egsdot"] = 10904,

7115 ["el"] = 10905,  
7116 ["eg"] = 10906,  
7117 ["siml"] = 10909,  
7118 ["simg"] = 10910,  
7119 ["simlE"] = 10911,  
7120 ["simgE"] = 10912,  
7121 ["LessLess"] = 10913,  
7122 ["NotNestedLessLess"] = {10913, 824},  
7123 ["GreaterGreater"] = 10914,  
7124 ["NotNestedGreaterGreater"] = {10914, 824},  
7125 ["glj"] = 10916,  
7126 ["gla"] = 10917,  
7127 ["ltcc"] = 10918,  
7128 ["gtcc"] = 10919,  
7129 ["lescc"] = 10920,  
7130 ["gescc"] = 10921,  
7131 ["smt"] = 10922,  
7132 ["lat"] = 10923,  
7133 ["smte"] = 10924,  
7134 ["smtes"] = {10924, 65024},  
7135 ["late"] = 10925,  
7136 ["lates"] = {10925, 65024},  
7137 ["bumpE"] = 10926,  
7138 ["NotPrecedesEqual"] = {10927, 824},  
7139 ["PrecedesEqual"] = 10927,  
7140 ["npre"] = {10927, 824},  
7141 ["npreceq"] = {10927, 824},  
7142 ["pre"] = 10927,  
7143 ["preceq"] = 10927,  
7144 ["NotSucceedsEqual"] = {10928, 824},  
7145 ["SucceedsEqual"] = 10928,  
7146 ["nsce"] = {10928, 824},  
7147 ["nsucceq"] = {10928, 824},  
7148 ["sce"] = 10928,  
7149 ["succeq"] = 10928,  
7150 ["prE"] = 10931,  
7151 ["scE"] = 10932,  
7152 ["precneqq"] = 10933,  
7153 ["prnE"] = 10933,  
7154 ["scnE"] = 10934,  
7155 ["succneqq"] = 10934,  
7156 ["prap"] = 10935,  
7157 ["precapprox"] = 10935,  
7158 ["scap"] = 10936,  
7159 ["succapprox"] = 10936,  
7160 ["precnapprox"] = 10937,  
7161 ["prnap"] = 10937,



7162 ["scnap"] = 10938,  
7163 ["succnapprox"] = 10938,  
7164 ["Pr"] = 10939,  
7165 ["Sc"] = 10940,  
7166 ["subdot"] = 10941,  
7167 ["supdot"] = 10942,  
7168 ["subplus"] = 10943,  
7169 ["supplus"] = 10944,  
7170 ["submult"] = 10945,  
7171 ["supmult"] = 10946,  
7172 ["subedot"] = 10947,  
7173 ["supedot"] = 10948,  
7174 ["nsubE"] = {10949, 824},  
7175 ["nsubseteqq"] = {10949, 824},  
7176 ["subE"] = 10949,  
7177 ["subseteqq"] = 10949,  
7178 ["nsupE"] = {10950, 824},  
7179 ["nsupseteqq"] = {10950, 824},  
7180 ["supE"] = 10950,  
7181 ["supseteqq"] = 10950,  
7182 ["subsim"] = 10951,  
7183 ["supsim"] = 10952,  
7184 ["subnE"] = 10955,  
7185 ["subsetneqq"] = 10955,  
7186 ["varsubsetneqq"] = {10955, 65024},  
7187 ["vsubnE"] = {10955, 65024},  
7188 ["supnE"] = 10956,  
7189 ["supsetneqq"] = 10956,  
7190 ["varsupsetneqq"] = {10956, 65024},  
7191 ["vsupnE"] = {10956, 65024},  
7192 ["csub"] = 10959,  
7193 ["csup"] = 10960,  
7194 ["csube"] = 10961,  
7195 ["csupe"] = 10962,  
7196 ["subsup"] = 10963,  
7197 ["supsub"] = 10964,  
7198 ["subsub"] = 10965,  
7199 ["supsup"] = 10966,  
7200 ["suphsub"] = 10967,  
7201 ["supdsub"] = 10968,  
7202 ["forkv"] = 10969,  
7203 ["topfork"] = 10970,  
7204 ["mlcp"] = 10971,  
7205 ["Dashv"] = 10980,  
7206 ["DoubleLeftTee"] = 10980,  
7207 ["Vdashl"] = 10982,  
7208 ["Barv"] = 10983,

7209 ["vBar"] = 10984,  
7210 ["vBarv"] = 10985,  
7211 ["Vbar"] = 10987,  
7212 ["Not"] = 10988,  
7213 ["bNot"] = 10989,  
7214 ["rnmid"] = 10990,  
7215 ["cirmid"] = 10991,  
7216 ["midcir"] = 10992,  
7217 ["topcir"] = 10993,  
7218 ["nhpar"] = 10994,  
7219 ["parsim"] = 10995,  
7220 ["nparsl"] = {11005, 8421},  
7221 ["parsl"] = 11005,  
7222 ["fflig"] = 64256,  
7223 ["filig"] = 64257,  
7224 ["fllig"] = 64258,  
7225 ["ffilig"] = 64259,  
7226 ["ffllig"] = 64260,  
7227 ["Ascr"] = 119964,  
7228 ["Cscr"] = 119966,  
7229 ["Dscr"] = 119967,  
7230 ["Gscr"] = 119970,  
7231 ["Jscr"] = 119973,  
7232 ["Kscr"] = 119974,  
7233 ["Nscr"] = 119977,  
7234 ["Oscr"] = 119978,  
7235 ["Pscr"] = 119979,  
7236 ["Qscr"] = 119980,  
7237 ["Sscr"] = 119982,  
7238 ["Tscr"] = 119983,  
7239 ["Uscr"] = 119984,  
7240 ["Vscr"] = 119985,  
7241 ["Wscr"] = 119986,  
7242 ["Xscr"] = 119987,  
7243 ["Yscr"] = 119988,  
7244 ["Zscr"] = 119989,  
7245 ["ascr"] = 119990,  
7246 ["bscr"] = 119991,  
7247 ["cscr"] = 119992,  
7248 ["dscr"] = 119993,  
7249 ["fscr"] = 119995,  
7250 ["hscr"] = 119997,  
7251 ["iscr"] = 119998,  
7252 ["jscr"] = 119999,  
7253 ["kscr"] = 120000,  
7254 ["lscr"] = 120001,  
7255 ["mscr"] = 120002,

7256 ["nscr"] = 120003,  
7257 ["pscr"] = 120005,  
7258 ["qscr"] = 120006,  
7259 ["rscr"] = 120007,  
7260 ["sscr"] = 120008,  
7261 ["tscr"] = 120009,  
7262 ["uscr"] = 120010,  
7263 ["vscr"] = 120011,  
7264 ["wscr"] = 120012,  
7265 ["xscr"] = 120013,  
7266 ["yscr"] = 120014,  
7267 ["zscr"] = 120015,  
7268 ["Afr"] = 120068,  
7269 ["Bfr"] = 120069,  
7270 ["Dfr"] = 120071,  
7271 ["Efr"] = 120072,  
7272 ["Ffr"] = 120073,  
7273 ["Gfr"] = 120074,  
7274 ["Jfr"] = 120077,  
7275 ["Kfr"] = 120078,  
7276 ["Lfr"] = 120079,  
7277 ["Mfr"] = 120080,  
7278 ["Nfr"] = 120081,  
7279 ["Ofr"] = 120082,  
7280 ["Pfr"] = 120083,  
7281 ["Qfr"] = 120084,  
7282 ["Sfr"] = 120086,  
7283 ["Tfr"] = 120087,  
7284 ["Ufr"] = 120088,  
7285 ["Vfr"] = 120089,  
7286 ["Wfr"] = 120090,  
7287 ["Xfr"] = 120091,  
7288 ["Yfr"] = 120092,  
7289 ["afr"] = 120094,  
7290 ["bfr"] = 120095,  
7291 ["cfr"] = 120096,  
7292 ["dfr"] = 120097,  
7293 ["efr"] = 120098,  
7294 ["ffr"] = 120099,  
7295 ["gfr"] = 120100,  
7296 ["hfr"] = 120101,  
7297 ["ifr"] = 120102,  
7298 ["jfr"] = 120103,  
7299 ["kfr"] = 120104,  
7300 ["lfr"] = 120105,  
7301 ["mfr"] = 120106,  
7302 ["nfr"] = 120107,

7303 ["ofr"] = 120108,  
7304 ["pfr"] = 120109,  
7305 ["qfr"] = 120110,  
7306 ["rfr"] = 120111,  
7307 ["sfr"] = 120112,  
7308 ["tfr"] = 120113,  
7309 ["ufr"] = 120114,  
7310 ["vfr"] = 120115,  
7311 ["wfr"] = 120116,  
7312 ["xfr"] = 120117,  
7313 ["yfr"] = 120118,  
7314 ["zfr"] = 120119,  
7315 ["Aopf"] = 120120,  
7316 ["Bopf"] = 120121,  
7317 ["Dopf"] = 120123,  
7318 ["Eopf"] = 120124,  
7319 ["Fopf"] = 120125,  
7320 ["Gopf"] = 120126,  
7321 ["Iopf"] = 120128,  
7322 ["Jopf"] = 120129,  
7323 ["Kopf"] = 120130,  
7324 ["Lopf"] = 120131,  
7325 ["Mopf"] = 120132,  
7326 ["Oopf"] = 120134,  
7327 ["Sopf"] = 120138,  
7328 ["Topf"] = 120139,  
7329 ["Uopf"] = 120140,  
7330 ["Vopf"] = 120141,  
7331 ["Wopf"] = 120142,  
7332 ["Xopf"] = 120143,  
7333 ["Yopf"] = 120144,  
7334 ["aopf"] = 120146,  
7335 ["bopf"] = 120147,  
7336 ["copf"] = 120148,  
7337 ["dopf"] = 120149,  
7338 ["eopf"] = 120150,  
7339 ["fopf"] = 120151,  
7340 ["gopf"] = 120152,  
7341 ["hopf"] = 120153,  
7342 ["iopf"] = 120154,  
7343 ["jopf"] = 120155,  
7344 ["kopf"] = 120156,  
7345 ["lopf"] = 120157,  
7346 ["mopf"] = 120158,  
7347 ["nopf"] = 120159,  
7348 ["oopf"] = 120160,  
7349 ["popf"] = 120161,

```

7350 ["qopf"] = 120162,
7351 ["ropf"] = 120163,
7352 ["sopf"] = 120164,
7353 ["topf"] = 120165,
7354 ["uopf"] = 120166,
7355 ["vopf"] = 120167,
7356 ["wopf"] = 120168,
7357 ["xopf"] = 120169,
7358 ["yopf"] = 120170,
7359 ["zopf"] = 120171,
7360 }

```

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

```

7361 function entities.dec_entity(s)
7362   local n = tonumber(s)
7363   if n == nil then
7364     return "&#" .. s .. ";" -- fallback for unknown entities
7365   end
7366   return utf8.char(n)
7367 end

```

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

```

7368 function entities.hex_entity(s)
7369   local n = tonumber("0x"..s)
7370   if n == nil then
7371     return "&#x" .. s .. ";" -- fallback for unknown entities
7372   end
7373   return utf8.char(n)
7374 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.

```

7375 function entities.hex_entity_with_x_char(x, s)
7376   local n = tonumber("0x"..s)
7377   if n == nil then
7378     return "&#" .. x .. s .. ";" -- fallback for unknown entities
7379   end
7380   return utf8.char(n)
7381 end

```

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

```

7382 function entities.char_entity(s)
7383   local code_points = character_entities[s]
7384   if code_points == nil then

```

```

7385     return "&" .. s .. ";"
7386 end
7387 if type(code_points) ~= 'table' then
7388     code_points = {code_points}
7389 end
7390 local char_table = {}
7391 for _, code_point in ipairs(code_points) do
7392     table.insert(char_table, utf8.char(code_point))
7393 end
7394 return table.concat(char_table)
7395 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.

```

7396 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.

```

7397 local parsers
7398 function M.writer.new(options)
7399     local self = {}

```

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

```

7400     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.

```

7401     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.

```
7402 local slice_specifiers = {}
7403 for specifier in options.slice:gmatch("[~%s]+") do
7404   table.insert(slice_specifiers, specifier)
7405 end
7406
7407 if #slice_specifiers == 2 then
7408   self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
7409   local slice_begin_type = self.slice_begin:sub(1, 1)
7410   if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
7411     self.slice_begin = "^" .. self.slice_begin
7412   end
7413   local slice_end_type = self.slice_end:sub(1, 1)
7414   if slice_end_type ~= "^" and slice_end_type ~= "$" then
7415     self.slice_end = "$" .. self.slice_end
7416   end
7417 elseif #slice_specifiers == 1 then
7418   self.slice_begin = "^" .. slice_specifiers[1]
7419   self.slice_end = "$" .. slice_specifiers[1]
7420 end
7421
7422 self.slice_begin_type = self.slice_begin:sub(1, 1)
7423 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
7424 self.slice_end_type = self.slice_end:sub(1, 1)
7425 self.slice_end_identifier = self.slice_end:sub(2) or ""
7426
7427 if self.slice_begin == "^" and self.slice_end ~= "^" then
7428   self.is_writing = true
7429 else
7430   self.is_writing = false
7431 end
```

### 3.1.4.2 Basic Formatter Variables and Functions

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

```
7432 self.space = " "
```

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

```
7433 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
7434 function self.plain(s)
7435   return s
7436 end
```

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

```
7437 function self.paragraph(s)
7438   if not self.is_writing then return "" end
7439   return s
7440 end
```

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

```
7441 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
7442 function self.interblocksep()
7443   if not self.is_writing then return "" end
7444   return self.interblocksep_text
7445 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.

```
7446 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
7447 function self.paragraphsep()
7448   if not self.is_writing then return "" end
7449   return self.paragraphsep_text
7450 end
```

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

```
7451 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
7452 function self.undosep()
7453   if not self.is_writing then return "" end
7454   return self.undosep_text
7455 end
```

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

```
7456 self.soft_line_break = function()
7457   if self.flatten_inlines then return "\n" end
7458   return "\\markdownRendererSoftLineBreak\n{"
7459 end
```

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

```
7460 self.hard_line_break = function()
7461   if self.flatten_inlines then return "\n" end
7462   return "\\markdownRendererHardLineBreak\n{"
7463 end
```

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

```
7464 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
7465 function self.thematic_break()
```



```

7466     if not self.is_writing then return "" end
7467     return "\\markdownRendererThematicBreak{}"
7468 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.

```

7469 self.escaped_uri_chars = {
7470   [{""] = "\\markdownRendererLeftBrace{}",
7471   ["}"] = "\\markdownRendererRightBrace{}",
7472   [{"\\"} = "\\markdownRendererBackslash{}",
7473   [{"r"} = " ",
7474   [{"n"} = " ",
7475 }
7476 self.escaped_minimal_strings = {
7477   [{"^"} = "\\markdownRendererCircumflex",
7478   .. "\\markdownRendererCircumflex ",
7479   [{"☒"} = "\\markdownRendererTickedBox{}",
7480   [{"◻"} = "\\markdownRendererHalfTickedBox{}",
7481   [{"□"} = "\\markdownRendererUntickedBox{}",
7482   [entities.hex_entity('FFFD')]
7483     = "\\markdownRendererReplacementCharacter{}",
7484 }

```

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

```

7485 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
7486 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

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

```

7487 self.escaped_chars = {
7488   [{""] = "\\markdownRendererLeftBrace{}",
7489   ["}"] = "\\markdownRendererRightBrace{}",
7490   [{"%"} = "\\markdownRendererPercentSign{}",
7491   [{"\\"} = "\\markdownRendererBackslash{}",
7492   [{"#"} = "\\markdownRendererHash{}",
7493   [{"$"} = "\\markdownRendererDollarSign{}",
7494   [{"&"} = "\\markdownRendererAmpersand{}",
7495   [{"_"} = "\\markdownRendererUnderscore{}",
7496   [{"^"} = "\\markdownRendererCircumflex{}",
7497   [{"~"} = "\\markdownRendererTilde{}",
7498   [{"|"} = "\\markdownRendererPipe{}",
7499   [entities.hex_entity('0000')]

```

```

7500     = "\\markdownRendererReplacementCharacter{ }",
7501   }

```

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

```

7502   local function create_escaper(char_escapes, string_escapes)
7503     local escape = util.escaper(char_escapes, string_escapes)
7504     return function(s)
7505       if self.flatten_inlines then return s end
7506       return escape(s)
7507     end
7508   end
7509   local escape_typographic_text = create_escaper(
7510     self.escaped_chars, self.escaped_strings)
7511   local escape_programmatic_text = create_escaper(
7512     self.escaped_uri_chars, self.escaped_minimal_strings)
7513   local escape_minimal = create_escaper(
7514     {}, 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.

```

7515   self.escape = escape_typographic_text
7516   self.math = escape_minimal
7517   if options.hybrid then
7518     self.identifier = escape_minimal
7519     self.string = escape_minimal
7520     self.uri = escape_minimal
7521     self.infostring = escape_minimal
7522   else
7523     self.identifier = escape_programmatic_text
7524     self.string = escape_typographic_text
7525     self.uri = escape_programmatic_text
7526     self.infostring = escape_programmatic_text
7527   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.

```
7528 function self.warning(t, m)
7529   return {"\markdownRendererWarning{" , self.escape(t), "}{" ,
7530         escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7531         escape_minimal(m or ""), "}"}
7532 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.

```
7533 function self.error(t, m)
7534   return {"\markdownRendererError{" , self.escape(t), "}{" ,
7535         escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
7536         escape_minimal(m or ""), "}"}
7537 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.

```
7538 function self.code(s, attributes)
7539   if self.flatten_inlines then return s end
7540   local buf = {}
7541   if attributes ~= nil then
7542     table.insert(buf,
7543       "\markdownRendererCodeSpanAttributeContextBegin\n")
7544     table.insert(buf, self.attributes(attributes))
7545   end
7546   table.insert(buf,
7547     {"\markdownRendererCodeSpan{" , self.escape(s), "}"}
7548   if attributes ~= nil then
7549     table.insert(buf,
7550       "\markdownRendererCodeSpanAttributeContextEnd{")
7551   end
7552   return buf
7553 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.

```
7554 function self.link(lab, src, tit, attributes)
7555   if self.flatten_inlines then return lab end
7556   local buf = {}
7557   if attributes ~= nil then
7558     table.insert(buf,
```

```

7559             "\markdownRendererLinkAttributeContextBegin\n")
7560         table.insert(buf, self.attributes(attributes))
7561     end
7562     table.insert(buf, {"\markdownRendererLink{" ,lab,"} ",
7563                     "{" ,self.escape(src),"} ",
7564                     "{" ,self.uri(src),"} ",
7565                     "{" ,self.string(tit or ""),"} ")
7566     if attributes ~= nil then
7567         table.insert(buf,
7568                     "\markdownRendererLinkAttributeContextEnd{ }")
7569     end
7570     return buf
7571 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.

```

7572 function self.image(lab, src, tit, attributes)
7573     if self.flatten_inlines then return lab end
7574     local buf = {}
7575     if attributes ~= nil then
7576         table.insert(buf,
7577                     "\markdownRendererImageAttributeContextBegin\n")
7578         table.insert(buf, self.attributes(attributes))
7579     end
7580     table.insert(buf, {"\markdownRendererImage{" ,lab,"} ",
7581                     "{" ,self.string(src),"} ",
7582                     "{" ,self.uri(src),"} ",
7583                     "{" ,self.string(tit or ""),"} ")
7584     if attributes ~= nil then
7585         table.insert(buf,
7586                     "\markdownRendererImageAttributeContextEnd{ }")
7587     end
7588     return buf
7589 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.

```

7590 function self.bulletlist(items,tight)
7591     if not self.is_writing then return "" end
7592     local buffer = {}
7593     for _,item in ipairs(items) do

```

```

7594     if item ~= "" then
7595         buffer[#buffer + 1] = self.bulletitem(item)
7596     end
7597 end
7598 local contents = util.intersperse(buffer,"\n")
7599 if tight and options.tightLists then
7600     return {"\\markdownRendererUlBeginTight\n",contents,
7601           "\n\\markdownRendererUlEndTight "}
7602 else
7603     return {"\\markdownRendererUlBegin\n",contents,
7604           "\n\\markdownRendererUlEnd "}
7605 end
7606 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.

```

7607 function self.bulletitem(s)
7608     return {"\\markdownRendererUlItem ",s,
7609           "\\markdownRendererUlItemEnd "}
7610 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.

```

7611 function self.orderedlist(items,tight,startnum)
7612     if not self.is_writing then return "" end
7613     local buffer = {}
7614     local num = startnum
7615     for _,item in ipairs(items) do
7616         if item ~= "" then
7617             buffer[#buffer + 1] = self.ordereditem(item,num)
7618         end
7619         if num ~= nil and item ~= "" then
7620             num = num + 1
7621         end
7622     end
7623     local contents = util.intersperse(buffer,"\n")
7624     if tight and options.tightLists then
7625         return {"\\markdownRendererOlBeginTight\n",contents,
7626               "\n\\markdownRendererOlEndTight "}
7627     else
7628         return {"\\markdownRendererOlBegin\n",contents,
7629               "\n\\markdownRendererOlEnd "}
7630     end
7631 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.

```
7632 function self.ordereditem(s,num)
7633   if num ~= nil then
7634     return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
7635           "\\markdownRendererOlItemEnd "}
7636   else
7637     return {"\\markdownRendererOlItem ",s,
7638           "\\markdownRendererOlItemEnd "}
7639   end
7640 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.

```
7641 function self.inline_html_comment(contents)
7642   if self.flatten_inlines then return contents end
7643   return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
7644 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.

```
7645 function self.inline_html_tag(contents)
7646   if self.flatten_inlines then return contents end
7647   return {"\\markdownRendererInlineHtmlTag{" ,
7648         self.string(contents),"}"}
7649 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.

```
7650 function self.block_html_element(s)
7651   if not self.is_writing then return "" end
7652   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
7653   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
7654 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.

```
7655 function self.emphasis(s)
7656   if self.flatten_inlines then return s end
```

```

7657     return {"\markdownRendererEmphasis{" ,s,"}"}
7658 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.

```

7659 function self.strong(s)
7660     if self.flatten_inlines then return s end
7661     return {"\markdownRendererStrongEmphasis{" ,s,"}"}
7662 end

```

#### 3.1.4.12 Formatter of Tickboxes

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

```

7663 function self.tickbox(f)
7664     if f == 1.0 then
7665         return "☒ "
7666     elseif f == 0.0 then
7667         return "☐ "
7668     else
7669         return "◻ "
7670     end
7671 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.

```

7672 function self.blockquote(s)
7673     if not self.is_writing then return "" end
7674     return {"\markdownRendererBlockQuoteBegin\n",s,
7675           "\markdownRendererBlockQuoteEnd "}
7676 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.

```

7677 function self.verbatim(s)
7678     if not self.is_writing then return "" end
7679     s = s:gsub("\n$", "")
7680     local name = util.cache_verbatim(options.cacheDir, s)
7681     return {"\markdownRendererInputVerbatim{" ,name,"}"}
7682 end

```

### 3.1.4.15 Formatter of Documents

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

```
7683 function self.document(d)
7684   local buf = {"\\markdownRendererDocumentBegin\n"}
7685
7686   -- warn against the `hybrid` option
7687   if options.hybrid then
7688     local text = "The `hybrid` option has been soft-deprecated."
7689     local more = "Consider using one of the following better options "
7690                 .. "for mixing TeX and markdown: `contentBlocks`, "
7691                 .. "`rawAttribute`, `texComments`, `texMathDollars`, "
7692                 .. "`texMathSingleBackslash`, and "
7693                 .. "`texMathDoubleBackslash`. "
7694                 .. "For more information, see the user manual at "
7695                 .. "<https://witiko.github.io/markdown/>."
7696     table.insert(buf, self.warning(text, more))
7697   end
7698
7699   -- insert the text of the document
7700   table.insert(buf, d)
7701
7702   -- pop all attributes
7703   table.insert(buf, self.pop_attributes())
7704
7705   table.insert(buf, "\\markdownRendererDocumentEnd")
7706
7707   return buf
7708 end
```

### 3.1.4.16 Formatter of Attributes

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

```
7709 local seen_identifiers = {}
7710 local key_value_regex = "([^\s=]+)%s*=%s*(.*)"
7711 local function normalize_attributes(attributes, auto_identifiers)
7712   -- normalize attributes
7713   local normalized_attributes = {}
7714   local has_explicit_identifiers = false
7715   local key, value
7716   for _, attribute in ipairs(attributes or {}) do
7717     if attribute:sub(1, 1) == "#" then
7718       table.insert(normalized_attributes, attribute)
7719       has_explicit_identifiers = true
7720       seen_identifiers[attribute:sub(2)] = true
7721     elseif attribute:sub(1, 1) == "." then
```



```

7722     table.insert(normalized_attributes, attribute)
7723 else
7724     key, value = attribute:match(key_value_regex)
7725     if key:lower() == "id" then
7726         table.insert(normalized_attributes, "#" .. value)
7727     elseif key:lower() == "class" then
7728         local classes = {}
7729         for class in value:gmatch("%S+") do
7730             table.insert(classes, class)
7731         end
7732         table.sort(classes)
7733         for _, class in ipairs(classes) do
7734             table.insert(normalized_attributes, "." .. class)
7735         end
7736     else
7737         table.insert(normalized_attributes, attribute)
7738     end
7739 end
7740 end
7741
7742 -- if no explicit identifiers exist, add auto identifiers
7743 if not has_explicit_identifiers and auto_identifiers ~= nil then
7744     local seen_auto_identifiers = {}
7745     for _, auto_identifier in ipairs(auto_identifiers) do
7746         if seen_auto_identifiers[auto_identifier] == nil then
7747             seen_auto_identifiers[auto_identifier] = true
7748             if seen_identifiers[auto_identifier] == nil then
7749                 seen_identifiers[auto_identifier] = true
7750                 table.insert(normalized_attributes,
7751                     "#" .. auto_identifier)
7752             else
7753                 local auto_identifier_number = 1
7754                 while true do
7755                     local numbered_auto_identifier = auto_identifier .. "-"
7756                                             .. auto_identifier_number
7757                     if seen_identifiers[numbered_auto_identifier] == nil then
7758                         seen_identifiers[numbered_auto_identifier] = true
7759                         table.insert(normalized_attributes,
7760                             "#" .. numbered_auto_identifier)
7761                     break
7762                 end
7763                 auto_identifier_number = auto_identifier_number + 1
7764             end
7765         end
7766     end
7767 end
7768 end

```

```

7769
7770     -- sort and deduplicate normalized attributes
7771     table.sort(normalized_attributes)
7772     local seen_normalized_attributes = {}
7773     local deduplicated_normalized_attributes = {}
7774     for _, attribute in ipairs(normalized_attributes) do
7775         if seen_normalized_attributes[attribute] == nil then
7776             seen_normalized_attributes[attribute] = true
7777             table.insert(deduplicated_normalized_attributes, attribute)
7778         end
7779     end
7780
7781     return deduplicated_normalized_attributes
7782 end
7783
7784 function self.attributes(attributes, should_normalize_attributes)
7785     local normalized_attributes
7786     if should_normalize_attributes == false then
7787         normalized_attributes = attributes
7788     else
7789         normalized_attributes = normalize_attributes(attributes)
7790     end
7791
7792     local buf = {}
7793     local key, value
7794     for _, attribute in ipairs(normalized_attributes) do
7795         if attribute:sub(1, 1) == "#" then
7796             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
7797                 attribute:sub(2), "}"})
7798         elseif attribute:sub(1, 1) == "." then
7799             table.insert(buf, {"\\markdownRendererAttributeName{" ,
7800                 attribute:sub(2), "}"})
7801         else
7802             key, value = attribute:match(key_value_regex)
7803             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
7804                 key, "{" , value, "}"})
7805         end
7806     end
7807
7808     return buf
7809 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.

```

7810     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`.

```
7811 self.attribute_type_levels = {}
7812 setmetatable(self.attribute_type_levels,
7813               { __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`.

```
7814 local function apply_attributes()
7815     local buf = {}
7816     for i = 1, #self.active_attributes do
7817         local start_output = self.active_attributes[i][3]
7818         if start_output ~= nil then
7819             table.insert(buf, start_output)
7820         end
7821     end
7822     return buf
7823 end
7824
7825 local function tear_down_attributes()
7826     local buf = {}
7827     for i = #self.active_attributes, 1, -1 do
7828         local end_output = self.active_attributes[i][4]
7829         if end_output ~= nil then
7830             table.insert(buf, end_output)
7831         end
7832     end
7833     return buf
7834 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.

```
7835 function self.push_attributes(attribute_type, attributes,
7836                               start_output, end_output)
7837     local attribute_type_level
7838     = self.attribute_type_levels[attribute_type]
7839     self.attribute_type_levels[attribute_type]
7840     = attribute_type_level + 1
7841
7842     -- index attributes in a hash table for easy lookup
7843     attributes = attributes or {}
7844     for i = 1, #attributes do
7845         attributes[attributes[i]] = true
```

```

7846     end
7847
7848     local buf = {}
7849     -- handle slicing
7850     if attributes["#" .. self.slice_end_identifrier] ~= nil and
7851        self.slice_end_type == "^" then
7852         if self.is_writing then
7853          table.insert(buf, self.undosep())
7854          table.insert(buf, tear_down_attributes())
7855         end
7856         self.is_writing = false
7857     end
7858     if attributes["#" .. self.slice_begin_identifrier] ~= nil and
7859        self.slice_begin_type == "^" then
7860      table.insert(buf, apply_attributes())
7861      self.is_writing = true
7862    end
7863    if self.is_writing and start_output ~= nil then
7864      table.insert(buf, start_output)
7865    end
7866    table.insert(self.active_attributes,
7867                {attribute_type, attributes,
7868                 start_output, end_output})
7869    return buf
7870  end
7871

```

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`).

```

7872  function self.pop_attributes(attribute_type)
7873    local buf = {}
7874    -- pop attributes until we find attributes of correct type
7875    -- or until no attributes remain
7876    local current_attribute_type = false
7877    while current_attribute_type ~= attribute_type and
7878        #self.active_attributes > 0 do
7879      local attributes, _, end_output
7880      current_attribute_type, attributes, _, end_output = table.unpack(
7881        self.active_attributes[#self.active_attributes])
7882      local attribute_type_level
7883      = self.attribute_type_levels[current_attribute_type]
7884      self.attribute_type_levels[current_attribute_type]
7885      = attribute_type_level - 1

```

```

7886     if self.is_writing and end_output ~= nil then
7887         table.insert(buf, end_output)
7888     end
7889     table.remove(self.active_attributes, #self.active_attributes)
7890     -- handle slicing
7891     if attributes["#" .. self.slice_end_identifer] ~= nil
7892         and self.slice_end_type == "$" then
7893         if self.is_writing then
7894             table.insert(buf, self.undosep())
7895             table.insert(buf, tear_down_attributes())
7896         end
7897         self.is_writing = false
7898     end
7899     if attributes["#" .. self.slice_begin_identifer] ~= nil and
7900         self.slice_begin_type == "$" then
7901         self.is_writing = true
7902         table.insert(buf, apply_attributes())
7903     end
7904 end
7905 return buf
7906 end

```

### 3.1.4.18 Automatically Generated Identifiers for Headings

Create an auto identifier string by stripping and converting characters from string `s`.

```

7907 local function create_auto_identifer(s)
7908     local buffer = {}
7909     local prev_space = false
7910     local letter_found = false
7911     local normalized_s = s
7912     if not options.unicodeNormalization
7913         or options.unicodeNormalizationForm ~= "nfc" then
7914         normalized_s = util.normalize(normalized_s, "nfc")
7915     end
7916
7917     for _, code in utf8.codes(normalized_s) do
7918         local char = utf8.char(code)
7919
7920         -- Remove everything up to the first letter.
7921         if not letter_found then
7922             local is_letter = lpeg.match(
7923                 parsers.unicode.following_alpha,
7924                 char
7925             )
7926             if is_letter then
7927                 letter_found = true

```

```

7928     else
7929         goto continue
7930     end
7931 end
7932
7933 -- Remove all non-alphanumeric characters, except underscores,
7934 -- hyphens, and periods.
7935 if not lpeg.match(
7936     ( parsers.underscore
7937     + parsers.dash
7938     + parsers.period
7939     + parsers.unicode.following_word
7940     + parsers.unicode.following_whitespace ),
7941     char
7942 ) then
7943     goto continue
7944 end
7945
7946 -- Replace all spaces and newlines with hyphens.
7947 if lpeg.match(
7948     ( parsers.newline
7949     + parsers.unicode.following_whitespace ),
7950     char
7951 ) then
7952     char = "-"
7953     if prev_space then
7954         goto continue
7955     else
7956         prev_space = true
7957     end
7958 else
7959     -- Case-fold all alphabetic characters.
7960     local form = nil
7961     if options.unicodeNormalization then
7962         form = options.unicodeNormalizationForm
7963     end
7964     char = util.casefold(char, form)
7965     prev_space = false
7966 end
7967
7968 table.insert(buffer, char)
7969
7970 ::continue::
7971 end
7972
7973 if prev_space then
7974     table.remove(buffer)

```

```

7975     end
7976
7977     local identifier = #buffer == 0 and "section"
7978                   or table.concat(buffer, "")
7979     return identifier
7980 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

7981 local function create_gfm_auto_identifier(s)
7982     local buffer = {}
7983     local prev_space = false
7984     local letter_found = false
7985     local normalized_s = s
7986     if not options.unicodeNormalization
7987       or options.unicodeNormalizationForm ~= "nfc" then
7988       normalized_s = util.normalize(normalized_s, "nfc")
7989     end
7990
7991     for _, code in utf8.codes(normalized_s) do
7992       local char = utf8.char(code)
7993
7994       -- Remove everything up to the first non-space.
7995       if not letter_found then
7996         local is_letter = not lpeg.match(
7997           parsers.unicode.following_whitespace,
7998           char
7999         )
8000         if is_letter then
8001           letter_found = true
8002         else
8003           goto continue
8004         end
8005       end
8006
8007       -- Remove all non-alphanumeric characters, except underscores
8008       -- and hyphens.
8009       if not lpeg.match(
8010         ( parsers.underscore
8011         + parsers.dash
8012         + parsers.unicode.following_word
8013         + parsers.unicode.following_whitespace ),
8014         char
8015       ) then
8016         prev_space = false
8017         goto continue
8018       end

```

```

8019
8020     -- Replace all spaces and newlines with hyphens.
8021     if lpeg.match(
8022         ( parsers.newline
8023         + parsers.unicode.following_whitespace ),
8024         char
8025     ) then
8026         char = "-"
8027         if prev_space then
8028             goto continue
8029         else
8030             prev_space = true
8031         end
8032     else
8033         -- Case-fold all alphabetic characters.
8034         local form = nil
8035         if options.unicodeNormalization then
8036             form = options.unicodeNormalizationForm
8037         end
8038         char = util.casefold(char, form)
8039         prev_space = false
8040     end
8041
8042     table.insert(buffer, char)
8043
8044     ::continue::
8045 end
8046
8047 if prev_space then
8048     table.remove(buffer)
8049 end
8050
8051 local identifier = #buffer == 0 and "section"
8052                 or table.concat(buffer, "")
8053 return identifier
8054 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.

```

8055 self.secbegin_text = "\\markdownRendererSectionBegin\n"
8056 self.secend_text = "\n\\markdownRendererSectionEnd "
8057 function self.heading(s, level, attributes)
8058     local buf = {}
8059     local flat_text, inlines = table.unpack(s)
8060

```



```

8061 -- push empty attributes for implied sections
8062 while self.attribute_type_levels["heading"] < level - 1 do
8063     table.insert(buf,
8064                 self.push_attributes("heading",
8065                                     nil,
8066                                     self.secbegin_text,
8067                                     self.secend_text))
8068 end
8069
8070 -- pop attributes for sections that have ended
8071 while self.attribute_type_levels["heading"] >= level do
8072     table.insert(buf, self.pop_attributes("heading"))
8073 end
8074
8075 -- construct attributes for the new section
8076 local auto_identifiers = {}
8077 if self.options.autoIdentifiers then
8078     table.insert(auto_identifiers, create_auto_identifier(flat_text))
8079 end
8080 if self.options.gfmAutoIdentifiers then
8081     table.insert(auto_identifiers,
8082                 create_gfm_auto_identifier(flat_text))
8083 end
8084 local normalized_attributes = normalize_attributes(attributes,
8085                                                  auto_identifiers)
8086
8087 -- push attributes for the new section
8088 local start_output = {}
8089 local end_output = {}
8090 table.insert(start_output, self.secbegin_text)
8091 table.insert(end_output, self.secend_text)
8092
8093 table.insert(buf, self.push_attributes("heading",
8094                                     normalized_attributes,
8095                                     start_output,
8096                                     end_output))
8097 assert(self.attribute_type_levels["heading"] == level)
8098
8099 -- render the heading and its attributes
8100 if self.is_writing and #normalized_attributes > 0 then
8101     table.insert(buf,
8102                 "\\markdownRendererHeaderAttributeContextBegin\n")
8103     table.insert(buf, self.attributes(normalized_attributes, false))
8104 end
8105
8106 local cmd
8107 level = level + options.shiftHeadings

```

```

8108     if level <= 1 then
8109         cmd = "\\markdownRendererHeadingOne"
8110     elseif level == 2 then
8111         cmd = "\\markdownRendererHeadingTwo"
8112     elseif level == 3 then
8113         cmd = "\\markdownRendererHeadingThree"
8114     elseif level == 4 then
8115         cmd = "\\markdownRendererHeadingFour"
8116     elseif level == 5 then
8117         cmd = "\\markdownRendererHeadingFive"
8118     elseif level >= 6 then
8119         cmd = "\\markdownRendererHeadingSix"
8120     else
8121         cmd = ""
8122     end
8123     if self.is_writing then
8124         table.insert(buf, {cmd, "{", inlines, "}"}
8125     end
8126
8127     if self.is_writing and #normalized_attributes > 0 then
8128         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
8129     end
8130
8131     return buf
8132 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.

```

8133 function self.get_state()
8134     return {
8135         is_writing=self.is_writing,
8136         flatten_inlines=self.flatten_inlines,
8137         active_attributes={table.unpack(self.active_attributes)},
8138     }
8139 end

```

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

```

8140 function self.set_state(s)
8141     local previous_state = self.get_state()
8142     for key, value in pairs(s) do
8143         self[key] = value
8144     end
8145     return previous_state
8146 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`.

```
8147 function self.defer_call(f)
8148     local previous_state = self.get_state()
8149     return function(...)
8150         local state = self.set_state(previous_state)
8151         local return_value = f(...)
8152         self.set_state(state)
8153         return return_value
8154     end
8155 end
8156
8157 return self
8158 end
```

### 3.1.5 Parsers

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

```
8159 parsers = {}
```

#### 3.1.5.1 Basic Parsers

```
8160 parsers.percent = P("%")
8161 parsers.at = P("@")
8162 parsers.comma = P(",")
8163 parsers.asterisk = P("*")
8164 parsers.dash = P("-")
8165 parsers.plus = P("+")
8166 parsers.underscore = P("_")
8167 parsers.period = P(".")
8168 parsers.hash = P("#")
8169 parsers.dollar = P("$")
8170 parsers.ampersand = P("&")
8171 parsers.backtick = P("`")
8172 parsers.less = P("<")
8173 parsers.more = P(">")
8174 parsers.space = P(" ")
8175 parsers.squote = P("'")
8176 parsers.dquote = P('"')
8177 parsers.lparent = P("(")
8178 parsers.rparent = P(")")
8179 parsers.lbracket = P("[")
8180 parsers.rbracket = P("]")
8181 parsers.lbrace = P("{")
```

```

8182 parsers.rbrace           = P("}")
8183 parsers.circumflex       = P("^")
8184 parsers.slash            = P("/")
8185 parsers.equal            = P("=")
8186 parsers.colon           = P(":")
8187 parsers.semicolon       = P(";")
8188 parsers.exclamation     = P("!")
8189 parsers.pipe            = P("|")
8190 parsers.tilde           = P("~")
8191 parsers.backslash       = P("\\")
8192 parsers.tab             = P("\t")
8193 parsers.newline        = P("\n")
8194
8195 parsers.digit           = R("09")
8196 parsers.hexdigit       = R("09","af","AF")
8197 parsers.letter        = R("AZ","az")
8198 parsers.alphanumeric  = R("AZ","az","09")
8199 parsers.keyword       = parsers.letter
8200                        * (parsers.alphanumeric + parsers.dash)^0
8201
8202 parsers.doubleasterisks = P("**")
8203 parsers.doubleunderscores = P("__")
8204 parsers.doubletildes   = P("~~")
8205 parsers.fourspace     = P("    ")
8206
8207 parsers.any           = P(1)
8208 parsers.succeed      = P(true)
8209 parsers.fail         = P(false)
8210
8211 parsers.internal_punctuation = S(":,.;.?")
8212 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
8213
8214 parsers.escapable    = parsers.ascii_punctuation
8215 parsers.anyescaped   = parsers.backslash / ""
8216                    * parsers.escapable
8217                    + parsers.any
8218
8219 parsers.spacechar    = S("\t ")
8220 parsers.spacing     = S(" \n\r\t")
8221 parsers.nonspacechar = parsers.any - parsers.spacing
8222 parsers.optionalspace = parsers.spacechar^0
8223
8224 parsers.normalchar   = parsers.any - (V("SpecialChar")
8225                                + parsers.spacing)
8226 parsers.eof         = -parsers.any
8227 parsers.nonindentpace = parsers.space^-3 * - parsers.spacechar
8228 parsers.indent      = parsers.space^-3 * parsers.tab

```

```

8229                                     + parsers.fourspace / ""
8230 parsers.linechar                     = P(1 - parsers.newline)
8231
8232 parsers.blankline                     = parsers.optionalspace
8233                                     * parsers.newline / "\n"
8234 parsers.blanklines                     = parsers.blankline^0
8235 parsers.skipblanklines                 = ( parsers.optionalspace
8236                                     * parsers.newline)^0
8237 parsers.indentedline                   = parsers.indent / ""
8238                                     * C( parsers.linechar^1
8239                                     * parsers.newline^-1)
8240 parsers.optionallyindentedline         = parsers.indent^-1 / ""
8241                                     * C( parsers.linechar^1
8242                                     * parsers.newline^-1)
8243 parsers.sp                             = parsers.spacing^0
8244 parsers.spnl                           = parsers.optionalspace
8245                                     * ( parsers.newline
8246                                     * parsers.optionalspace)^-1
8247 parsers.line                           = parsers.linechar^0 * parsers.newline
8248 parsers.nonemptyline                   = parsers.line - parsers.blankline
8249

```

### 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.

```

8250 parsers.unicode = {}
8251 parsers.unicode.preceding_punctuation = parsers.fail
8252 parsers.unicode.following_punctuation = parsers.fail
8253 parsers.unicode.following_alpha = parsers.fail
8254 parsers.unicode.following_word = parsers.fail
8255 parsers.unicode.preceding_whitespace = parsers.fail
8256 parsers.unicode.following_whitespace = parsers.fail
8257 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>35</sup>.

```

8258   local punctuation_of_length_n
8259       = unicode_data.categories.P[n]
8260       + unicode_data.categories.S[n]
8261   parsers.unicode.preceding_punctuation
8262       = parsers.unicode.preceding_punctuation
8263       + B(punctuation_of_length_n)
8264   parsers.unicode.following_punctuation

```

<sup>35</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

8265     = parsers.unicode.following_punctuation
8266     + #punctuation_of_length_n

```

For alphabetical characters, accept any characters from Unicode category L (letter), similar to the character class ‘Unicode.

```

8267     local alpha_of_length_n = unicode_data.categories.L[n]
8268     parsers.unicode.following_alpha
8269     = parsers.unicode.following_alpha
8270     + alpha_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 ‘

```

8271     local word_of_length_n
8272     = unicode_data.categories.L[n]
8273     + unicode_data.categories.N[n]
8274     + unicode_data.categories.Pc[n]
8275     parsers.unicode.following_word
8276     = parsers.unicode.following_word
8277     + 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.

```

8278     local whitespace_of_length_n = unicode_data.categories.Z[n]
8279     if n == 1 then
8280         whitespace_of_length_n
8281         = whitespace_of_length_n
8282         + R("\t\r")
8283     end
8284     parsers.unicode.preceding_whitespace
8285     = parsers.unicode.preceding_whitespace
8286     + B(whitespace_of_length_n)
8287     parsers.unicode.following_whitespace
8288     = parsers.unicode.following_whitespace
8289     + #whitespace_of_length_n
8290 end

```

### 3.1.5.3 Parsers Used for Indentation

```

8291
8292 parsers.leader     = parsers.space^-3
8293

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

8294 local function has_trail(indent_table)
8295     return indent_table ~= nil and
8296         indent_table.trail ~= nil and
8297         next(indent_table.trail) ~= nil
8298 end

```

8299

Check if indent table `indent_table` has any indents.

```
8300 local function has_indents(indent_table)
8301   return indent_table ~= nil and
8302     indent_table.indents ~= nil and
8303     next(indent_table.indents) ~= nil
8304 end
8305
```

Add a trail `trail_info` to the indent table `indent_table`.

```
8306 local function add_trail(indent_table, trail_info)
8307   indent_table.trail = trail_info
8308   return indent_table
8309 end
8310
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
8311 local function remove_trail(indent_table)
8312   indent_table.trail = nil
8313   return indent_table
8314 end
8315
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
8316 local function update_indent_table(indent_table, new_indent, add)
8317   indent_table = remove_trail(indent_table)
8318
8319   if not has_indents(indent_table) then
8320     indent_table.indents = {}
8321   end
8322
8323
8324   if add then
8325     indent_table.indents[#indent_table.indents + 1] = new_indent
8326   else
8327     if indent_table.indents[#indent_table.indents].name
8328       == new_indent.name then
8329       indent_table.indents[#indent_table.indents] = nil
8330     end
8331   end
8332
8333   return indent_table
8334 end
8335
```

Remove an indent by its name `name`.

```
8336 local function remove_indent(name)
8337   local remove_indent_level =
```

```

8338     function(s, i, indent_table) -- luacheck: ignore s i
8339         indent_table = update_indent_table(indent_table, {name=name},
8340                                           false)
8341         return true, indent_table
8342     end
8343
8344     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
8345 end
8346

```

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`.

```

8347 local function process_starter_spacing(indent, spacing,
8348                                     minimum, left_strip_length)
8349     left_strip_length = left_strip_length or 0
8350
8351     local count = 0
8352     local tab_value = 4 - (indent) % 4
8353
8354     local code_started, minimum_found = false, false
8355     local code_start, minimum_remainder = "", ""
8356
8357     local left_total_stripped = 0
8358     local full_remainder = ""
8359
8360     if spacing ~= nil then
8361         for i = 1, #spacing do
8362             local character = spacing:sub(i, i)
8363
8364             if character == "\t" then
8365                 count = count + tab_value
8366                 tab_value = 4
8367             elseif character == " " then
8368                 count = count + 1
8369                 tab_value = 4 - (1 - tab_value) % 4
8370             end
8371
8372             if (left_strip_length ~= 0) then
8373                 local possible_to_strip = math.min(count, left_strip_length)
8374                 count = count - possible_to_strip
8375                 left_strip_length = left_strip_length - possible_to_strip
8376                 left_total_stripped = left_total_stripped + possible_to_strip
8377             else
8378                 full_remainder = full_remainder .. character

```



```

8379     end
8380
8381     if (minimum_found) then
8382         minimum_remainder = minimum_remainder .. character
8383     elseif (count >= minimum) then
8384         minimum_found = true
8385         minimum_remainder = minimum_remainder
8386             .. string.rep(" ", count - minimum)
8387     end
8388
8389     if (code_started) then
8390         code_start = code_start .. character
8391     elseif (count >= minimum + 4) then
8392         code_started = true
8393         code_start = code_start
8394             .. string.rep(" ", count - (minimum + 4))
8395     end
8396 end
8397 end
8398
8399 local remainder
8400 if (code_started) then
8401     remainder = code_start
8402 else
8403     remainder = string.rep(" ", count - minimum)
8404 end
8405
8406 local is_minimum = count >= minimum
8407 return {
8408     is_code = code_started,
8409     remainder = remainder,
8410     left_total_stripped = left_total_stripped,
8411     is_minimum = is_minimum,
8412     minimum_remainder = minimum_remainder,
8413     total_length = count,
8414     full_remainder = full_remainder
8415 }
8416 end
8417

```

Count the total width of all indents in the indent table `indent_table`.

```

8418 local function count_indent_tab_level(indent_table)
8419     local count = 0
8420     if not has_indents(indent_table) then
8421         return count
8422     end
8423
8424     for i=1, #indent_table.indents do

```

```

8425     count = count + indent_table.indents[i].length
8426   end
8427   return count
8428 end
8429

```

Count the total width of a delimiter `delimiter`.

```

8430 local function total_delimiter_length(delimiter)
8431   local count = 0
8432   if type(delimiter) == "string" then return #delimiter end
8433   for _, value in pairs(delimiter) do
8434     count = count + total_delimiter_length(value)
8435   end
8436   return count
8437 end
8438

```

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`.

```

8439 local function process_starter_indent(_, _, indent_table, starter,
8440                                     is_blank, indent_type, breakable)
8441   local last_trail = starter[1]
8442   local delimiter = starter[2]
8443   local raw_new_trail = starter[3]
8444
8445   if indent_type == "bq" and not breakable then
8446     indent_table.ignore_blockquote_blank = true
8447   end
8448
8449   if has_trail(indent_table) then
8450     local trail = indent_table.trail
8451     if trail.is_code then
8452       return false
8453     end
8454     last_trail = trail.remainder
8455   else
8456     local sp = process_starter_spacing(0, last_trail, 0, 0)
8457
8458     if sp.is_code then
8459       return false
8460     end
8461     last_trail = sp.remainder
8462   end
8463
8464   local preceding_indentation = count_indent_tab_level(indent_table) % 4
8465   local last_trail_length = #last_trail
8466   local delimiter_length = total_delimiter_length(delimiter)
8467

```

```

8468 local total_indent_level = preceding_indentation + last_trail_length
8469                               + delimiter_length
8470
8471 local sp = {}
8472 if not is_blank then
8473     sp = process_starter_spacing(total_indent_level, raw_new_trail,
8474                               0, 1)
8475 end
8476
8477 local del_trail_length = sp.left_total_stripped
8478 if is_blank then
8479     del_trail_length = 1
8480 elseif not sp.is_code then
8481     del_trail_length = del_trail_length + #sp.remainder
8482 end
8483
8484 local indent_length = last_trail_length + delimiter_length
8485                               + del_trail_length
8486 local new_indent_info = {name=indent_type, length=indent_length}
8487
8488 indent_table = update_indent_table(indent_table, new_indent_info,
8489                                   true)
8490 indent_table = add_trail(indent_table,
8491                         {is_code=sp.is_code,
8492                         remainder=sp.remainder,
8493                         total_length=sp.total_length,
8494                         full_remainder=sp.full_remainder})
8495
8496 return true, indent_table
8497 end
8498

```

Return the pattern corresponding with the indent name `name`.

```

8499 local function decode_pattern(name)
8500     local delimiter = parsers.succeed
8501     if name == "bq" then
8502         delimiter = parsers.more
8503     end
8504
8505     return C(parsers.optionalspace) * C(delimiter)
8506           * C(parsers.optionalspace) * Cp()
8507 end
8508

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

8509 local function left_blank_starter(indent_table)
8510     local blank_starter_index

```

```

8511
8512   if not has_indents(indent_table) then
8513       return
8514   end
8515
8516   for i = #indent_table.indents,1,-1 do
8517       local value = indent_table.indents[i]
8518       if value.name == "li" then
8519           blank_starter_index = i
8520       else
8521           break
8522       end
8523   end
8524
8525   return blank_starter_index
8526 end
8527

```

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.

```

8528 local function traverse_indent(s, i, indent_table, is_optional,
8529                               is_blank, current_line_indents)
8530     local new_index = i
8531
8532     local preceding_indentation = 0
8533     local current_trail = {}
8534
8535     local blank_starter = left_blank_starter(indent_table)
8536
8537     if current_line_indents == nil then
8538         current_line_indents = {}
8539     end
8540
8541     for index = 1,#indent_table.indents do
8542         local value = indent_table.indents[index]
8543         local pattern = decode_pattern(value.name)
8544
8545         -- match decoded pattern
8546         local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
8547         if new_indent_info == nil then
8548             local blankline_end = lpeg.match(
8549                 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
8550             if is_optional or not indent_table.ignore_blockquote_blank
8551                 or not blankline_end then

```

```

8552         return is_optional, new_index, current_trail,
8553                current_line_indents
8554     end
8555
8556     return traverse_indent(s, tonumber(blankline_end.pos),
8557                           indent_table, is_optional, is_blank,
8558                           current_line_indents)
8559 end
8560
8561 local raw_last_trail = new_indent_info[1]
8562 local delimiter = new_indent_info[2]
8563 local raw_new_trail = new_indent_info[3]
8564 local next_index = new_indent_info[4]
8565
8566 local space_only = delimiter == ""
8567
8568 -- check previous trail
8569 if not space_only and next(current_trail) == nil then
8570     local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
8571     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8572                    total_length=sp.total_length,
8573                    full_remainder=sp.full_remainder}
8574 end
8575
8576 if next(current_trail) ~= nil then
8577     if not space_only and current_trail.is_code then
8578         return is_optional, new_index, current_trail,
8579                current_line_indents
8580     end
8581     if current_trail.internal_remainder ~= nil then
8582         raw_last_trail = current_trail.internal_remainder
8583     end
8584 end
8585
8586 local raw_last_trail_length = 0
8587 local delimiter_length = 0
8588
8589 if not space_only then
8590     delimiter_length = #delimiter
8591     raw_last_trail_length = #raw_last_trail
8592 end
8593
8594 local total_indent_level = preceding_indentation
8595                          + raw_last_trail_length + delimiter_length
8596
8597 local spacing_to_process
8598 local minimum = 0

```

```

8599     local left_strip_length = 0
8600
8601     if not space_only then
8602         spacing_to_process = raw_new_trail
8603         left_strip_length = 1
8604     else
8605         spacing_to_process = raw_last_trail
8606         minimum = value.length
8607     end
8608
8609     local sp = process_starter_spacing(total_indent_level,
8610                                     spacing_to_process, minimum,
8611                                     left_strip_length)
8612
8613     if space_only and not sp.is_minimum then
8614         return is_optional or (is_blank and blank_starter <= index),
8615             new_index, current_trail, current_line_indents
8616     end
8617
8618     local indent_length = raw_last_trail_length + delimiter_length
8619                       + sp.left_total_stripped
8620
8621     -- update info for the next pattern
8622     if not space_only then
8623         preceding_indentation = preceding_indentation + indent_length
8624     else
8625         preceding_indentation = preceding_indentation + value.length
8626     end
8627
8628     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
8629                   internal_remainder=sp.minimum_remainder,
8630                   total_length=sp.total_length,
8631                   full_remainder=sp.full_remainder}
8632
8633     current_line_indents[#current_line_indents + 1] = new_indent_info
8634     new_index = next_index
8635 end
8636
8637 return true, new_index, current_trail, current_line_indents
8638 end
8639

```

Check if a code trail is expected.

```

8640 local function check_trail(expect_code, is_code)
8641     return (expect_code and is_code) or (not expect_code and not is_code)
8642 end
8643

```

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`.

```
8644 local check_trail_joined =
8645   function(s, i, indent_table, -- luacheck: ignore s i
8646           spacing, expect_code, omit_remainder)
8647     local is_code
8648     local remainder
8649
8650     if has_trail(indent_table) then
8651       local trail = indent_table.trail
8652       is_code = trail.is_code
8653       if is_code then
8654         remainder = trail.remainder
8655       else
8656         remainder = trail.full_remainder
8657       end
8658     else
8659       local sp = process_starter_spacing(0, spacing, 0, 0)
8660       is_code = sp.is_code
8661       if is_code then
8662         remainder = sp.remainder
8663       else
8664         remainder = sp.full_remainder
8665       end
8666     end
8667
8668     local result = check_trail(expect_code, is_code)
8669     if omit_remainder then
8670       return result
8671     end
8672     return result, remainder
8673   end
8674
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
8675 local check_trail_length =
8676   function(s, i, indent_table, -- luacheck: ignore s i
8677           spacing, min, max)
8678     local trail
8679
8680     if has_trail(indent_table) then
8681       trail = indent_table.trail
8682     else
8683       trail = process_starter_spacing(0, spacing, 0, 0)
8684     end
8685
```

```

8686     local total_length = trail.total_length
8687     if total_length == nil then
8688         return false
8689     end
8690
8691     return min <= total_length and total_length <= max
8692 end
8693

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

8694 local function check_continuation_indentation(s, i, indent_table,
8695                                             is_optional, is_blank)
8696     if not has_indents(indent_table) then
8697         return true
8698     end
8699
8700     local passes, new_index, current_trail, current_line_indents =
8701         traverse_indent(s, i, indent_table, is_optional, is_blank)
8702
8703     if passes then
8704         indent_table.current_line_indents = current_line_indents
8705         indent_table = add_trail(indent_table, current_trail)
8706         return new_index, indent_table
8707     end
8708     return false
8709 end
8710

```

Get name of the last indent from the `indent_table`.

```

8711 local function get_last_indent_name(indent_table)
8712     if has_indents(indent_table) then
8713         return indent_table.indents[#indent_table.indents].name
8714     end
8715 end
8716

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

8717 local function remove_remainder_if_blank(indent_table, remainder)
8718     if get_last_indent_name(indent_table) == "li" then
8719         return ""
8720     end
8721     return remainder
8722 end
8723

```



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.

```
8724 local check_trail_type =
8725   function(s, i, -- luacheck: ignore s i
8726             trail, spacing, trail_type)
8727     if trail == nil then
8728       trail = process_starter_spacing(0, spacing, 0, 0)
8729     end
8730
8731     if trail_type == "non-code" then
8732       return check_trail(false, trail.is_code)
8733     end
8734     if trail_type == "code" then
8735       return check_trail(true, trail.is_code)
8736     end
8737     if trail_type == "full-code" then
8738       if (trail.is_code) then
8739         return i, trail.remainder
8740       end
8741       return i, ""
8742     end
8743     if trail_type == "full-any" then
8744       return i, trail.internal_remainder
8745     end
8746   end
8747
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
8748 local trail_freezing =
8749   function(s, i, -- luacheck: ignore s i
8750             indent_table, is_freezing)
8751     if is_freezing then
8752       if indent_table.is_trail_frozen then
8753         indent_table.trail = indent_table.frozen_trail
8754       else
8755         indent_table.frozen_trail = indent_table.trail
8756         indent_table.is_trail_frozen = true
8757       end
8758     else
8759       indent_table.frozen_trail = nil
8760       indent_table.is_trail_frozen = false
8761     end
8762     return true, indent_table
8763   end
8764
```

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`.

```
8765 local check_continuation_indentation_and_trail =
8766   function (s, i, indent_table, is_optional, is_blank, trail_type,
8767             reset_rem, omit_remainder)
8768     if not has_indents(indent_table) then
8769       local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
8770                                             * Cp(), s, i)
8771       local result, remainder = check_trail_type(s, i,
8772         indent_table.trail, spacing, trail_type)
8773       if remainder == nil then
8774         if result then
8775           return new_index
8776         end
8777         return false
8778       end
8779       if result then
8780         return new_index, remainder
8781       end
8782       return false
8783     end
8784
8785     local passes, new_index, current_trail = traverse_indent(s, i,
8786       indent_table, is_optional, is_blank)
8787
8788     if passes then
8789       local spacing
8790       if current_trail == nil then
8791         local newer_spacing, newer_index = lpeg.match(
8792           C(parsers.spacechar^0) * Cp(), s, i)
8793         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
8794         new_index = newer_index
8795         spacing = newer_spacing
8796       else
8797         spacing = current_trail.remainder
8798       end
8799       local result, remainder = check_trail_type(s, new_index,
8800         current_trail, spacing, trail_type)
8801       if remainder == nil or omit_remainder then
8802         if result then
8803           return new_index
8804         end
8805         return false
8806       end
8807
8808       if is_blank and reset_rem then
8809         remainder = remove_remainder_if_blank(indent_table, remainder)
```

```

8810     end
8811     if result then
8812         return new_index, remainder
8813     end
8814     return false
8815 end
8816 return false
8817 end
8818

```

The following patterns check whitespace indentation at the start of a block.

```

8819 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
8820     * Cc(false), check_trail_joined)
8821
8822 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
8823     * C(parsers.spacechar^0) * Cc(false)
8824     * Cc(true), check_trail_joined)
8825
8826 parsers.check_code_trail = Cmt( Cb("indent_info")
8827     * C(parsers.spacechar^0)
8828     * Cc(true), check_trail_joined)
8829
8830 parsers.check_trail_length_range = function(min, max)
8831     return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
8832         * Cc(max), check_trail_length)
8833 end
8834
8835 parsers.check_trail_length = function(n)
8836     return parsers.check_trail_length_range(n, n)
8837 end
8838

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

8839 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
8840     * Cc(true), trail_freezing), "indent_info")
8841
8842 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
8843     trail_freezing), "indent_info")
8844

```

The following patterns check indentation in continuation lines as defined by the container start.

```

8845 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
8846     check_continuation_indentation)
8847
8848 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
8849     check_continuation_indentation)

```

```

8850
8851 parsers.check_minimal_blank_indent
8852   = Cmt( Cb("indent_info") * Cc(false)
8853         * Cc(true)
8854         , check_continuation_indentation)
8855

```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```

8856
8857 parsers.check_minimal_indent_and_trail =
8858   Cmt( Cb("indent_info")
8859       * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
8860       , check_continuation_indentation_and_trail)
8861
8862 parsers.check_minimal_indent_and_code_trail =
8863   Cmt( Cb("indent_info")
8864       * Cc(false) * Cc(false) * Cc("code") * Cc(false)
8865       , check_continuation_indentation_and_trail)
8866
8867 parsers.check_minimal_blank_indent_and_full_code_trail =
8868   Cmt( Cb("indent_info")
8869       * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
8870       , check_continuation_indentation_and_trail)
8871
8872 parsers.check_minimal_indent_and_any_trail =
8873   Cmt( Cb("indent_info")
8874       * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
8875       , check_continuation_indentation_and_trail)
8876
8877 parsers.check_minimal_blank_indent_and_any_trail =
8878   Cmt( Cb("indent_info")
8879       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
8880       , check_continuation_indentation_and_trail)
8881
8882 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
8883   Cmt( Cb("indent_info")
8884       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
8885       , check_continuation_indentation_and_trail)
8886
8887 parsers.check_optional_indent_and_any_trail =
8888   Cmt( Cb("indent_info")
8889       * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
8890       , check_continuation_indentation_and_trail)
8891
8892 parsers.check_optional_blank_indent_and_any_trail =
8893   Cmt( Cb("indent_info")

```

```

8894     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
8895     , check_continuation_indentation_and_trail)
8896

```

The following patterns specify behaviour around newlines.

```

8897
8898 parsers.spnlc_noexc = parsers.optionalspace
8899                       * ( parsers.newline
8900                       * parsers.check_minimal_indent_and_any_trail)^-1
8901
8902 parsers.spnlc = parsers.optionalspace
8903                 * (V("EndlineNoSub"))^-1
8904
8905 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
8906                  + parsers.spacechar^1
8907
8908 parsers.only_blank = parsers.spacechar^0
8909                  * (parsers.newline + parsers.eof)
8910

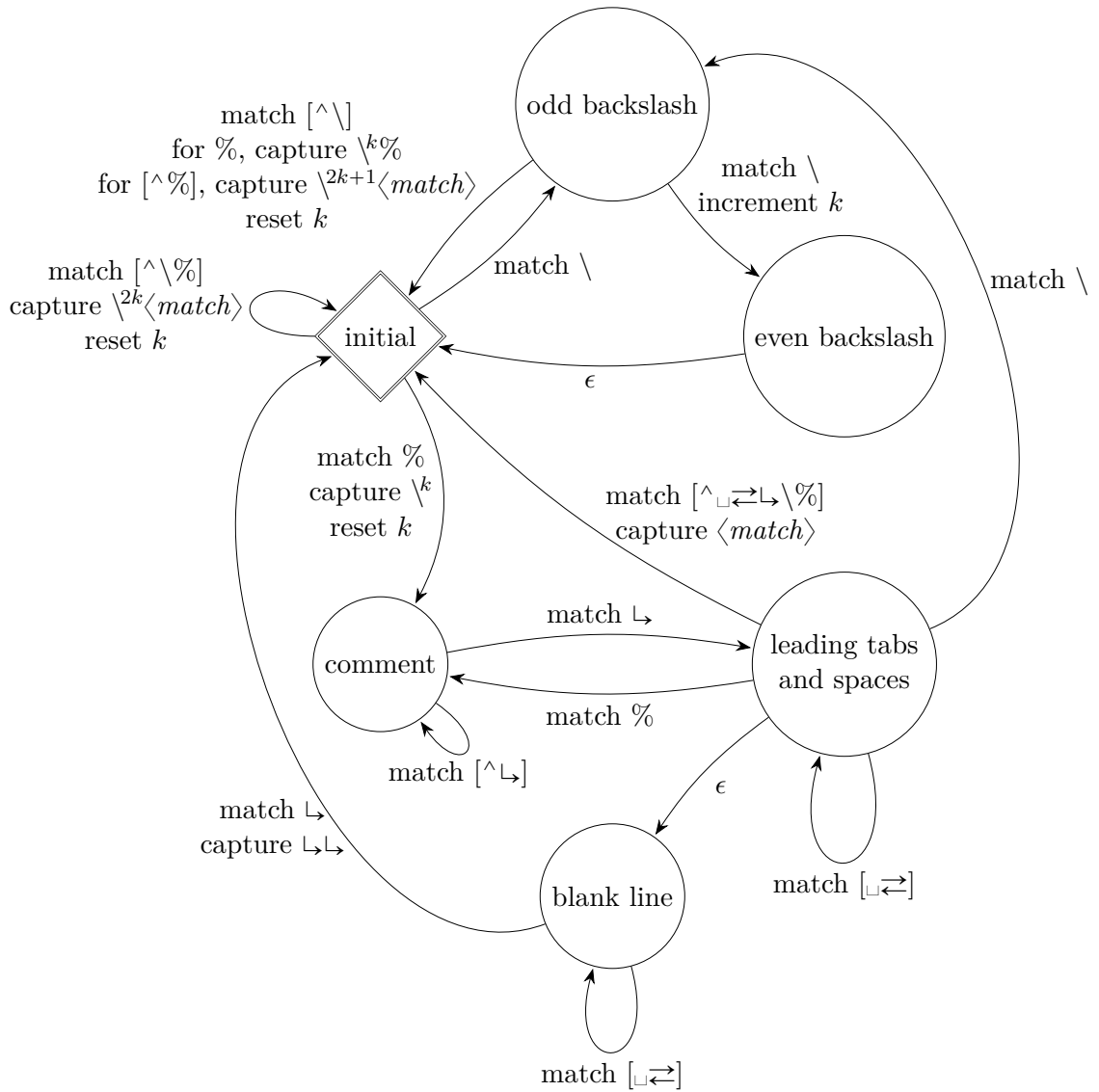
```

The `parsers.commented_line^1` parser recognizes the regular language of  $\text{T}_{\text{E}}\text{X}$  comments, see an equivalent finite automaton in Figure 8.

```

8911 parsers.commented_line_letter = parsers.linechar
8912                               + parsers.newline
8913                               - parsers.backslash
8914                               - parsers.percent
8915 parsers.commented_line = Cg(Cc(""), "backslashes")
8916                       * ((#(parsers.commented_line_letter
8917                       - parsers.newline)
8918                       * Cb("backslashes")
8919                       * Cs(parsers.commented_line_letter
8920                       - parsers.newline)^1 -- initial
8921                       * Cg(Cc(""), "backslashes"))
8922                       + #( parsers.backslash
8923                       * (parsers.backslash + parsers.newline))
8924                       * Cg((parsers.backslash -- even backslash
8925                       * ( parsers.backslash
8926                       + #parsers.newline))^1, "backslashes")
8927                       + (parsers.backslash
8928                       * (#parsers.percent
8929                       * Cb("backslashes")
8930                       / function(backslashes)
8931                       return string.rep("\\", #backslashes / 2)
8932                       end
8933                       * C(parsers.percent)
8934                       + #parsers.commented_line_letter
8935                       * Cb("backslashes")
8936                       * Cc("\\")

```



**Figure 8: A pushdown automaton that recognizes TeX comments**

```

8937         * C(parsers.commented_line_letter))
8938         * Cg(Cc(""), "backslashes"))^0
8939     * (#parsers.percent
8940     * Cb("backslashes")
8941     / function(backslashes)
8942     return string.rep("\\", #backslashes / 2)
8943     end
8944     * ((parsers.percent -- comment
8945     * parsers.line
8946     * #parsers.blankline) -- blank line
8947     / "\n"
8948     + parsers.percent -- comment
8949     * parsers.line
8950     * parsers.optionalspace) -- leading spaces
8951     + #(parsers.newline)
8952     * Cb("backslashes")
8953     * C(parsers.newline))
8954
8955 parsers.chunk = parsers.line * (parsers.optionallyindentedline
8956                             - parsers.blankline)^0
8957
8958 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
8959 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
8960 parsers.attribute_key = (parsers.attribute_key_char
8961                       - parsers.dash - parsers.digit)
8962                       * parsers.attribute_key_char^0
8963 parsers.attribute_value = ( (parsers.dquote / "")
8964                          * (parsers.anyescaped - parsers.dquote)^0
8965                          * (parsers.dquote / ""))
8966 + ( (parsers.squote / "")
8967    * (parsers.anyescaped - parsers.squote)^0
8968    * (parsers.squote / ""))
8969 + ( parsers.anyescaped
8970    - parsers.dquote
8971    - parsers.rbrace
8972    - parsers.space)^0
8973 parsers.attribute_identifier = parsers.attribute_key_char^1
8974 parsers.attribute_classname = parsers.letter
8975                             * parsers.attribute_key_char^0
8976 parsers.attribute_raw = parsers.attribute_raw_char^1
8977
8978 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
8979                   + C( parsers.hash
8980                       * parsers.attribute_identifier)
8981                   + C( parsers.period
8982                       * parsers.attribute_classname)
8983                   + Cs( parsers.attribute_key

```

```

8984         * parsers.optionalspace
8985         * parsers.equal
8986         * parsers.optionalspace
8987         * parsers.attribute_value)
8988 parsers.attributes = parsers.lbrace
8989         * parsers.optionalspace
8990         * parsers.attribute
8991         * (parsers.spacechar^1
8992         * parsers.attribute)^0
8993         * parsers.optionalspace
8994         * parsers.rbrace
8995
8996 parsers.raw_attribute = parsers.lbrace
8997         * parsers.optionalspace
8998         * parsers.equal
8999         * C(parsers.attribute_raw)
9000         * parsers.optionalspace
9001         * parsers.rbrace
9002
9003 -- block followed by 0 or more optionally
9004 -- indented blocks with first line indented.
9005 parsers.indented_blocks = function(bl)
9006   return Cs( bl
9007     * ( parsers.blankline^1
9008     * parsers.indent
9009     * -parsers.blankline
9010     * bl)^0
9011     * (parsers.blankline^1 + parsers.eof) )
9012 end

```

### 3.1.5.4 Parsers Used for HTML Entities

```

9013 local function repeat_between(pattern, min, max)
9014   return -pattern^(max + 1) * pattern^min
9015 end
9016
9017 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
9018         * C(repeat_between(parsers.hexdigit, 1, 6))
9019         * parsers.semicolon
9020 parsers.decentity = parsers.ampersand * parsers.hash
9021         * C(repeat_between(parsers.digit, 1, 7))
9022         * parsers.semicolon
9023 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
9024         * parsers.semicolon
9025
9026 parsers.html_entities
9027   = parsers.hexentity / entities.hex_entity_with_x_char

```



```

9028 + parsers.decentity / entities.dec_entity
9029 + parsers.tagentity / entities.char_entity

```

### 3.1.5.5 Parsers Used for Markdown Lists

```

9030 parsers.bullet = function(bullet_char, interrupting)
9031   local allowed_end
9032   if interrupting then
9033     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9034   else
9035     allowed_end = C(parsers.spacechar^1)
9036                 + #(parsers.newline + parsers.eof)
9037   end
9038   return parsers.check_trail
9039         * Ct(C(bullet_char) * Cc(""))
9040         * allowed_end
9041 end
9042
9043 local function tickbox(interior)
9044   return parsers.optionalspace * parsers.lbracket
9045         * interior * parsers.rbracket * parsers.spacechar^1
9046 end
9047
9048 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
9049 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
9050 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
9051

```

### 3.1.5.6 Parsers Used for Markdown Code Spans

```

9052 parsers.openticks = Cg(parsers.backtick^1, "ticks")
9053
9054 local function captures_equal_length(_,i,a,b)
9055   return #a == #b and i
9056 end
9057
9058 parsers.closeticks = Cmt(C(parsers.backtick^1)
9059                          * Cb("ticks"), captures_equal_length)
9060
9061 parsers.intickschar = (parsers.any - S("\n\r`"))
9062                      + V("NoSoftLineBreakEndline")
9063                      + (parsers.backtick^1 - parsers.closeticks)
9064
9065 local function process_inticks(s)
9066   s = s:gsub("\n", " ")
9067   s = s:gsub("^ (.*) $", "%1")
9068   return s
9069 end

```

```

9070
9071 parsers.inticks = parsers.openticks
9072         * C(parsers.space^0)
9073         * parsers.closeticks
9074         + parsers.openticks
9075         * Cs(Cs(parsers.intickschar^0) / process_inticks)
9076         * parsers.closeticks
9077

```

### 3.1.5.7 Parsers Used for HTML

```

9078 -- case-insensitive match (we assume s is lowercase)
9079 -- must be single byte encoding
9080 parsers.keyword_exact = function(s)
9081     local parser = P(0)
9082     for i=1,#s do
9083         local c = s:sub(i,i)
9084         local m = c .. upper(c)
9085         parser = parser * S(m)
9086     end
9087     return parser
9088 end
9089
9090 parsers.special_block_keyword =
9091     parsers.keyword_exact("pre") +
9092     parsers.keyword_exact("script") +
9093     parsers.keyword_exact("style") +
9094     parsers.keyword_exact("textarea")
9095
9096 parsers.block_keyword =
9097     parsers.keyword_exact("address") +
9098     parsers.keyword_exact("article") +
9099     parsers.keyword_exact("aside") +
9100     parsers.keyword_exact("base") +
9101     parsers.keyword_exact("basefont") +
9102     parsers.keyword_exact("blockquote") +
9103     parsers.keyword_exact("body") +
9104     parsers.keyword_exact("caption") +
9105     parsers.keyword_exact("center") +
9106     parsers.keyword_exact("col") +
9107     parsers.keyword_exact("colgroup") +
9108     parsers.keyword_exact("dd") +
9109     parsers.keyword_exact("details") +
9110     parsers.keyword_exact("dialog") +
9111     parsers.keyword_exact("dir") +
9112     parsers.keyword_exact("div") +
9113     parsers.keyword_exact("dl") +

```

```
9114     parsers.keyword_exact("dt") +
9115     parsers.keyword_exact("fieldset") +
9116     parsers.keyword_exact("figcaption") +
9117     parsers.keyword_exact("figure") +
9118     parsers.keyword_exact("footer") +
9119     parsers.keyword_exact("form") +
9120     parsers.keyword_exact("frame") +
9121     parsers.keyword_exact("frameset") +
9122     parsers.keyword_exact("h1") +
9123     parsers.keyword_exact("h2") +
9124     parsers.keyword_exact("h3") +
9125     parsers.keyword_exact("h4") +
9126     parsers.keyword_exact("h5") +
9127     parsers.keyword_exact("h6") +
9128     parsers.keyword_exact("head") +
9129     parsers.keyword_exact("header") +
9130     parsers.keyword_exact("hr") +
9131     parsers.keyword_exact("html") +
9132     parsers.keyword_exact("iframe") +
9133     parsers.keyword_exact("legend") +
9134     parsers.keyword_exact("li") +
9135     parsers.keyword_exact("link") +
9136     parsers.keyword_exact("main") +
9137     parsers.keyword_exact("menu") +
9138     parsers.keyword_exact("menuitem") +
9139     parsers.keyword_exact("nav") +
9140     parsers.keyword_exact("noframes") +
9141     parsers.keyword_exact("ol") +
9142     parsers.keyword_exact("optgroup") +
9143     parsers.keyword_exact("option") +
9144     parsers.keyword_exact("p") +
9145     parsers.keyword_exact("param") +
9146     parsers.keyword_exact("section") +
9147     parsers.keyword_exact("source") +
9148     parsers.keyword_exact("summary") +
9149     parsers.keyword_exact("table") +
9150     parsers.keyword_exact("tbody") +
9151     parsers.keyword_exact("td") +
9152     parsers.keyword_exact("tfoot") +
9153     parsers.keyword_exact("th") +
9154     parsers.keyword_exact("thead") +
9155     parsers.keyword_exact("title") +
9156     parsers.keyword_exact("tr") +
9157     parsers.keyword_exact("track") +
9158     parsers.keyword_exact("ul")
9159
9160 -- end conditions
```

```

9161 parsers.html_blankline_end_condition
9162   = parsers.linechar^0
9163   * ( parsers.newline
9164     * (parsers.check_minimal_blank_indent_and_any_trail
9165       * #parsers.blankline
9166       + parsers.check_minimal_indent_and_any_trail)
9167     * parsers.linechar^1)^0
9168   * (parsers.newline^-1 / "")
9169
9170 local function remove_trailing_blank_lines(s)
9171   return s:gsub("[\n\r]+%s*$", "")
9172 end
9173
9174 parsers.html_until_end = function(end_marker)
9175   return Cs(Cs((parsers.newline
9176     * (parsers.check_minimal_blank_indent_and_any_trail
9177     * #parsers.blankline
9178     + parsers.check_minimal_indent_and_any_trail)
9179     + parsers.linechar - end_marker)^0
9180     * parsers.linechar^0 * parsers.newline^-1)
9181     / remove_trailing_blank_lines)
9182 end
9183
9184 -- attributes
9185 parsers.html_attribute_spacing = parsers.optionalspace
9186                               * V("NoSoftLineBreakEndline")
9187                               * parsers.optionalspace
9188                               + parsers.spacechar^1
9189
9190 parsers.html_attribute_name = ( parsers.letter
9191                               + parsers.colon
9192                               + parsers.underscore)
9193                               * ( parsers.alphanumeric
9194                               + parsers.colon
9195                               + parsers.underscore
9196                               + parsers.period
9197                               + parsers.dash)^0
9198
9199 parsers.html_attribute_value = parsers.squote
9200                               * (parsers.linechar - parsers.squote)^0
9201                               * parsers.squote
9202                               + parsers.dquote
9203                               * (parsers.linechar - parsers.dquote)^0
9204                               * parsers.dquote
9205                               + ( parsers.any
9206                               - parsers.spacechar
9207                               - parsers.newline

```

```

9208         - parsers.dquote
9209         - parsers.squote
9210         - parsers.backtick
9211         - parsers.equal
9212         - parsers.less
9213         - parsers.more)^1
9214
9215 parsers.html_inline_attribute_value = parsers.squote
9216         * (V("NoSoftLineBreakEndline")
9217         + parsers.any
9218         - parsers.blankline^2
9219         - parsers.squote)^0
9220         * parsers.squote
9221         + parsers.dquote
9222         * (V("NoSoftLineBreakEndline")
9223         + parsers.any
9224         - parsers.blankline^2
9225         - parsers.dquote)^0
9226         * parsers.dquote
9227         + (parsers.any
9228         - parsers.spacechar
9229         - parsers.newline
9230         - parsers.dquote
9231         - parsers.squote
9232         - parsers.backtick
9233         - parsers.equal
9234         - parsers.less
9235         - parsers.more)^1
9236
9237 parsers.html_attribute_value_specification
9238 = parsers.optionalspace
9239 * parsers.equal
9240 * parsers.optionalspace
9241 * parsers.html_attribute_value
9242
9243 parsers.html_spnl = parsers.optionalspace
9244         * (V("NoSoftLineBreakEndline")
9245         * parsers.optionalspace)^-1
9246
9247 parsers.html_inline_attribute_value_specification
9248 = parsers.html_spnl
9249 * parsers.equal
9250 * parsers.html_spnl
9251 * parsers.html_inline_attribute_value
9252
9253 parsers.html_attribute
9254 = parsers.html_attribute_spacing

```

```

9255 * parsers.html_attribute_name
9256 * parsers.html_inline_attribute_value_specification^-1
9257
9258 parsers.html_non_newline_attribute
9259 = parsers.spacechar^1
9260 * parsers.html_attribute_name
9261 * parsers.html_attribute_value_specification^-1
9262
9263 parsers.nested_breaking_blank = parsers.newline
9264                               * parsers.check_minimal_blank_indent
9265                               * parsers.blankline
9266
9267 parsers.html_comment_start = P("<!--")
9268
9269 parsers.html_comment_end = P("-->")
9270
9271 parsers.html_comment
9272 = Cs( parsers.html_comment_start
9273       * parsers.html_until_end(parsers.html_comment_end))
9274
9275 parsers.html_inline_comment = (parsers.html_comment_start / "")
9276                             * -P(">") * -P("-->")
9277                             * Cs(( V("NoSoftLineBreakEndline")
9278                                   + parsers.any
9279                                   - parsers.nested_breaking_blank
9280                                   - parsers.html_comment_end)^0)
9281                             * (parsers.html_comment_end / "")
9282
9283 parsers.html_cdatasection_start = P("<![CDATA[")
9284
9285 parsers.html_cdatasection_end = P("]]>")
9286
9287 parsers.html_cdatasection
9288 = Cs( parsers.html_cdatasection_start
9289       * parsers.html_until_end(parsers.html_cdatasection_end))
9290
9291 parsers.html_inline_cdatasection
9292 = parsers.html_cdatasection_start
9293 * Cs(V("NoSoftLineBreakEndline") + parsers.any
9294     - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
9295 * parsers.html_cdatasection_end
9296
9297 parsers.html_declaration_start = P("<!") * parsers.letter
9298
9299 parsers.html_declaration_end = P(">")
9300
9301 parsers.html_declaration

```

```

9302 = Cs( parsers.html_declaration_start
9303     * parsers.html_until_end(parsers.html_declaration_end))
9304
9305 parsers.html_inline_declaration
9306 = parsers.html_declaration_start
9307 * Cs(V("NoSoftLineBreakEndline") + parsers.any
9308     - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
9309 * parsers.html_declaration_end
9310
9311 parsers.html_instruction_start = P("<?")
9312
9313 parsers.html_instruction_end = P("?>")
9314
9315 parsers.html_instruction
9316 = Cs( parsers.html_instruction_start
9317     * parsers.html_until_end(parsers.html_instruction_end))
9318
9319 parsers.html_inline_instruction = parsers.html_instruction_start
9320     * Cs( V("NoSoftLineBreakEndline")
9321         + parsers.any
9322         - parsers.nested_breaking_blank
9323         - parsers.html_instruction_end)^0
9324     * parsers.html_instruction_end
9325
9326 parsers.html_blankline = parsers.newline
9327     * parsers.optionalspace
9328     * parsers.newline
9329
9330 parsers.html_tag_start = parsers.less
9331
9332 parsers.html_tag_closing_start = parsers.less
9333     * parsers.slash
9334
9335 parsers.html_tag_end = parsers.html_spnl
9336     * parsers.more
9337
9338 parsers.html_empty_tag_end = parsers.html_spnl
9339     * parsers.slash
9340     * parsers.more
9341
9342 -- opening tags
9343 parsers.html_any_open_inline_tag = parsers.html_tag_start
9344     * parsers.keyword
9345     * parsers.html_attribute^0
9346     * parsers.html_tag_end
9347
9348 parsers.html_any_open_tag = parsers.html_tag_start

```

```

9349             * parsers.keyword
9350             * parsers.html_non_newline_attribute^0
9351             * parsers.html_tag_end
9352
9353 parsers.html_open_tag = parsers.html_tag_start
9354             * parsers.block_keyword
9355             * parsers.html_attribute^0
9356             * parsers.html_tag_end
9357
9358 parsers.html_open_special_tag = parsers.html_tag_start
9359             * parsers.special_block_keyword
9360             * parsers.html_attribute^0
9361             * parsers.html_tag_end
9362
9363 -- incomplete tags
9364 parsers.incomplete_tag_following = parsers.spacechar
9365             + parsers.more
9366             + parsers.slash * parsers.more
9367             + #(parsers.newline + parsers.eof)
9368
9369 parsers.incomplete_special_tag_following = parsers.spacechar
9370             + parsers.more
9371             + #( parsers.newline
9372             + parsers.eof)
9373
9374 parsers.html_incomplete_open_tag = parsers.html_tag_start
9375             * parsers.block_keyword
9376             * parsers.incomplete_tag_following
9377
9378 parsers.html_incomplete_open_special_tag
9379 = parsers.html_tag_start
9380 * parsers.special_block_keyword
9381 * parsers.incomplete_special_tag_following
9382
9383 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
9384             * parsers.block_keyword
9385             * parsers.incomplete_tag_following
9386
9387 parsers.html_incomplete_close_special_tag
9388 = parsers.html_tag_closing_start
9389 * parsers.special_block_keyword
9390 * parsers.incomplete_tag_following
9391
9392 -- closing tags
9393 parsers.html_close_tag = parsers.html_tag_closing_start
9394             * parsers.block_keyword
9395             * parsers.html_tag_end

```



```

9396
9397 parsers.html_any_close_tag = parsers.html_tag_closing_start
9398     * parsers.keyword
9399     * parsers.html_tag_end
9400
9401 parsers.html_close_special_tag = parsers.html_tag_closing_start
9402     * parsers.special_block_keyword
9403     * parsers.html_tag_end
9404
9405 -- empty tags
9406 parsers.html_any_empty_inline_tag = parsers.html_tag_start
9407     * parsers.keyword
9408     * parsers.html_attribute^0
9409     * parsers.html_empty_tag_end
9410
9411 parsers.html_any_empty_tag = parsers.html_tag_start
9412     * parsers.keyword
9413     * parsers.html_non_newline_attribute^0
9414     * parsers.optionalspace
9415     * parsers.slash
9416     * parsers.more
9417
9418 parsers.html_empty_tag = parsers.html_tag_start
9419     * parsers.block_keyword
9420     * parsers.html_attribute^0
9421     * parsers.html_empty_tag_end
9422
9423 parsers.html_empty_special_tag = parsers.html_tag_start
9424     * parsers.special_block_keyword
9425     * parsers.html_attribute^0
9426     * parsers.html_empty_tag_end
9427
9428 parsers.html_incomplete_blocks
9429     = parsers.html_incomplete_open_tag
9430     + parsers.html_incomplete_open_special_tag
9431     + parsers.html_incomplete_close_tag
9432
9433 -- parse special html blocks
9434 parsers.html_blankline_ending_special_block_opening
9435     = ( parsers.html_close_special_tag
9436         + parsers.html_empty_special_tag)
9437     * #( parsers.optionalspace
9438         * (parsers.newline + parsers.eof))
9439
9440 parsers.html_blankline_ending_special_block
9441     = parsers.html_blankline_ending_special_block_opening
9442     * parsers.html_blankline_end_condition

```

```

9443
9444 parsers.html_special_block_opening
9445     = parsers.html_incomplete_open_special_tag
9446     - parsers.html_empty_special_tag
9447
9448 parsers.html_closing_special_block
9449     = parsers.html_special_block_opening
9450     * parsers.html_until_end(parsers.html_close_special_tag)
9451
9452 parsers.html_special_block
9453     = parsers.html_blankline_ending_special_block
9454     + parsers.html_closing_special_block
9455
9456 -- parse html blocks
9457 parsers.html_block_opening = parsers.html_incomplete_open_tag
9458                             + parsers.html_incomplete_close_tag
9459
9460 parsers.html_block = parsers.html_block_opening
9461                    * parsers.html_blankline_end_condition
9462
9463 -- parse any html blocks
9464 parsers.html_any_block_opening
9465     = ( parsers.html_any_open_tag
9466         + parsers.html_any_close_tag
9467         + parsers.html_any_empty_tag)
9468     * #(parsers.optionalspace * (parsers.newline + parsers.eof))
9469
9470 parsers.html_any_block = parsers.html_any_block_opening
9471                        * parsers.html_blankline_end_condition
9472
9473 parsers.html_inline_comment_full = parsers.html_comment_start
9474                                  * -P(">") * -P("->")
9475                                  * Cs(( V("NoSoftLineBreakEndline")
9476                                         + parsers.any - P("--")
9477                                         - parsers.nested_breaking_blank
9478                                         - parsers.html_comment_end)^0)
9479                                  * parsers.html_comment_end
9480
9481 parsers.html_inline_tags = parsers.html_inline_comment_full
9482                          + parsers.html_any_empty_inline_tag
9483                          + parsers.html_inline_instruction
9484                          + parsers.html_inline_cdatasection
9485                          + parsers.html_inline_declaration
9486                          + parsers.html_any_open_inline_tag
9487                          + parsers.html_any_close_tag
9488

```

### 3.1.5.8 Parsers Used for Markdown Tags and Links

```
9489 parsers.urlchar = parsers.anyescaped
9490                 - parsers.newline
9491                 - parsers.more
9492
9493 parsers.auto_link_scheme_part = parsers.alphanumeric
9494                             + parsers.plus
9495                             + parsers.period
9496                             + parsers.dash
9497
9498 parsers.auto_link_scheme = parsers.letter
9499                             * parsers.auto_link_scheme_part
9500                             * parsers.auto_link_scheme_part^-30
9501
9502 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
9503                       * ( parsers.any - parsers.spacing
9504                         - parsers.less - parsers.more)^0
9505
9506 parsers.printable_characters = S("!.#%&'*/=?^_`{|}~-")
9507
9508 parsers.email_address_local_part_char = parsers.alphanumeric
9509                                         + parsers.printable_characters
9510
9511 parsers.email_address_local_part
9512   = parsers.email_address_local_part_char^1
9513
9514 parsers.email_address_dns_label = parsers.alphanumeric
9515                                 * ( parsers.alphanumeric
9516                                   + parsers.dash)^-62
9517                                 * B(parsers.alphanumeric)
9518
9519 parsers.email_address_domain = parsers.email_address_dns_label
9520                               * ( parsers.period
9521                                 * parsers.email_address_dns_label)^0
9522
9523 parsers.email_address = parsers.email_address_local_part
9524                       * parsers.at
9525                       * parsers.email_address_domain
9526
9527 parsers.auto_link_url = parsers.less
9528                       * C(parsers.absolute_uri)
9529                       * parsers.more
9530
9531 parsers.auto_link_email = parsers.less
9532                       * C(parsers.email_address)
9533                       * parsers.more
9534
```

```

9535 parsers.auto_link_relative_reference = parsers.less
9536                                     * C(parsers.urlchar^1)
9537                                     * parsers.more
9538
9539 parsers.autolink = parsers.auto_link_url
9540                 + parsers.auto_link_email
9541
9542 -- content in balanced brackets, parentheses, or quotes:
9543 parsers.bracketed = P{ parsers.lbracket
9544                     * (( parsers.backslash / "\"" * parsers.rbracket
9545                       + parsers.any - (parsers.lbracket
9546                                     + parsers.rbracket
9547                                     + parsers.blankline^2)
9548                       ) + V(1))^0
9549                     * parsers.rbracket }
9550
9551 parsers.inparens = P{ parsers.lparent
9552                     * ((parsers.anyescaped - (parsers.lparent
9553                                               + parsers.rparent
9554                                               + parsers.blankline^2)
9555                     ) + V(1))^0
9556                     * parsers.rparent }
9557
9558 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
9559                     * ((parsers.anyescaped - (parsers.squote
9560                                               + parsers.blankline^2)
9561                     ) + V(1))^0
9562                     * parsers.squote }
9563
9564 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
9565                     * ((parsers.anyescaped - (parsers.dquote
9566                                               + parsers.blankline^2)
9567                     ) + V(1))^0
9568                     * parsers.dquote }
9569
9570 parsers.link_text = parsers.lbracket
9571                   * Cs((parsers.alphanumeric^1
9572                       + parsers.bracketed
9573                       + parsers.inticks
9574                       + parsers.autolink
9575                       + V("InlineHtml"))
9576                       + ( parsers.backslash * parsers.backslash)
9577                       + ( parsers.backslash
9578                           * ( parsers.lbracket
9579                               + parsers.rbracket)
9580                       + V("NoSoftLineBreakSpace"))
9581                       + V("NoSoftLineBreakEndline"))

```

```

9582         + (parsers.any
9583           - ( parsers.newline
9584             + parsers.lbracket
9585             + parsers.rbracket
9586             + parsers.blankline^2))))^0)
9587     * parsers.rbracket
9588
9589 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
9590     * #( ( parsers.any
9591         - parsers.rbracket)^-999
9592         * parsers.rbracket)
9593     * Cs((parsers.alphanumeric^1
9594         + parsers.inticks
9595         + parsers.autolink
9596         + V("InlineHtml")
9597         + ( parsers.backslash * parsers.backslash)
9598         + ( parsers.backslash
9599           * ( parsers.lbracket
9600             + parsers.rbracket)
9601           + V("NoSoftLineBreakSpace")
9602           + V("NoSoftLineBreakEndline")
9603           + (parsers.any
9604             - ( parsers.newline
9605               + parsers.lbracket
9606               + parsers.rbracket
9607               + parsers.blankline^2))))^1)
9608
9609 parsers.link_label = parsers.lbracket
9610     * parsers.link_label_body
9611     * parsers.rbracket
9612
9613 parsers.inparens_url = P{ parsers.lparent
9614     * ((parsers.anyescaped - (parsers.lparent
9615                               + parsers.rparent
9616                               + parsers.spacing)
9617       ) + V(1))^0
9618     * parsers.rparent }
9619
9620 -- url for markdown links, allowing nested brackets:
9621 parsers.url = parsers.less * Cs((parsers.anyescaped
9622     - parsers.newline
9623     - parsers.less
9624     - parsers.more)^0)
9625     * parsers.more
9626     + -parsers.less
9627     * Cs((parsers.inparens_url + (parsers.anyescaped
9628     - parsers.spacing

```

```

9629                                     - parsers.lparent
9630                                     - parsers.rparent))^1)
9631
9632 -- quoted text:
9633 parsers.title_s      = parsers.squote
9634                       * Cs((parsers.html_entities
9635                           + V("NoSoftLineBreakSpace")
9636                           + V("NoSoftLineBreakEndline")
9637                           + ( parsers.anyescaped
9638                             - parsers.newline
9639                             - parsers.squote
9640                             - parsers.blankline^2))^0)
9641                       * parsers.squote
9642
9643 parsers.title_d      = parsers.dquote
9644                       * Cs((parsers.html_entities
9645                           + V("NoSoftLineBreakSpace")
9646                           + V("NoSoftLineBreakEndline")
9647                           + ( parsers.anyescaped
9648                             - parsers.newline
9649                             - parsers.dquote
9650                             - parsers.blankline^2))^0)
9651                       * parsers.dquote
9652
9653 parsers.title_p      = parsers.lparent
9654                       * Cs((parsers.html_entities
9655                           + V("NoSoftLineBreakSpace")
9656                           + V("NoSoftLineBreakEndline")
9657                           + ( parsers.anyescaped
9658                             - parsers.newline
9659                             - parsers.lparent
9660                             - parsers.rparent
9661                             - parsers.blankline^2))^0)
9662                       * parsers.rparent
9663
9664 parsers.title
9665     = parsers.title_d + parsers.title_s + parsers.title_p
9666
9667 parsers.optionaltitle
9668     = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
9669

```

### 3.1.5.9 Helpers for Links and Link Reference Definitions

```

9670 -- parse a reference definition: [foo]: /bar "title"
9671 parsers.define_reference_parser = (parsers.check_trail / "")
9672                                 * parsers.link_label * parsers.colon

```

```

9673                                     * parsers.spnlc * parsers.url
9674                                     * ( parsers.spnlc_sep * parsers.title
9675                                     * parsers.only_blank
9676                                     + Cc("") * parsers.only_blank)

```

### 3.1.5.10 Inline Elements

```

9677 parsers.Inline                       = V("Inline")
9678
9679 -- parse many p between starter and ender
9680 parsers.between = function(p, starter, ender)
9681   local ender2 = B(parsers.nonspacechar) * ender
9682   return ( starter
9683           * #parsers.nonspacechar
9684           * Ct(p * (p - ender2)^0)
9685           * ender2)
9686 end
9687

```

### 3.1.5.11 Block Elements

```

9688 parsers.lineof = function(c)
9689   return ( parsers.check_trail_no_rem
9690           * (P(c) * parsers.optionalspace)^3
9691           * (parsers.newline + parsers.eof))
9692 end
9693
9694 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
9695                               + parsers.lineof(parsers.dash)
9696                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.12 Headings

```

9697 -- parse Atx heading start and return level
9698 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
9699                       * -parsers.hash / length
9700
9701 -- parse setext header ending and return level
9702 parsers.heading_level
9703   = parsers.nonindentospace * parsers.equal^1
9704   * parsers.optionalspace * #parsers.newline * Cc(1)
9705   + parsers.nonindentospace * parsers.dash^1
9706   * parsers.optionalspace * #parsers.newline * Cc(2)
9707
9708 local function strip_atx_end(s)
9709   return s:gsub("%s+#+%s*\n$", "")
9710 end
9711

```

```

9712 parsers.atx_heading = parsers.check_trail_no_rem
9713         * Cg(parsers.heading_start, "level")
9714         * (C( parsers.optionalspace
9715             * parsers.hash^0
9716             * parsers.optionalspace
9717             * parsers.newline)
9718         + parsers.spacechar^1
9719         * 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`.

```

9720 M.reader = {}
9721 function M.reader.new(writer, options)
9722     local self = {}

```

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

```

9723     self.writer = writer
9724     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.

```

9725     self.parsers = {}
9726     (function(parsers)
9727         setmetatable(self.parsers, {
9728             __index = function (_, key)
9729                 return parsers[key]
9730             end
9731         })
9732     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.

```

9733     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.

```
9734 function self.normalize_tag(tag)
9735     tag = util.ropetostring(tag)
9736     tag = tag:gsub("[\n\r\t]+", " ")
9737     tag = tag:gsub("^ ", ""):gsub(" $", "")
9738     local form = nil
9739     if options.unicodeNormalization then
9740         form = options.unicodeNormalizationForm
9741     end
9742     tag = util.casefold(tag, form)
9743     return tag
9744 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.

```
9745 local function iterlines(s, f)
9746     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
9747     return util.ropetostring(rope)
9748 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.

```
9749 if options.preserveTabs then
9750     self.expandtabs = function(s) return s end
9751 else
9752     self.expandtabs = function(s)
9753         if s:find("\t") then
9754             return iterlines(s, util.expand_tabs_in_line)
9755         else
9756             return s
9757         end
9758     end
9759 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.

```
9760 self.parser_functions = {}
9761 self.create_parser = function(name, grammar, topLevel)
9762     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.

```

9763     if toplevel and options.stripIndent then
9764         local min_prefix_length, min_prefix = nil, ''
9765         str = iterlines(str, function(line)
9766             if lpeg.match(parsers.nonemptyline, line) == nil then
9767                 return line
9768             end
9769             line = util.expand_tabs_in_line(line)
9770             local prefix = lpeg.match(C(parsers.optionalspace), line)
9771             local prefix_length = #prefix
9772             local is_shorter = min_prefix_length == nil
9773             if not is_shorter then
9774                 is_shorter = prefix_length < min_prefix_length
9775             end
9776             if is_shorter then
9777                 min_prefix_length, min_prefix = prefix_length, prefix
9778             end
9779             return line
9780         end)
9781         str = str:gsub('^' .. min_prefix, '')
9782     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.

```

9783     if toplevel and (options.texComments or options.hybrid) then
9784         str = lpeg.match(Ct(parsers.commented_line^1), str)
9785         str = util.rope_to_string(str)
9786     end
9787     local res = lpeg.match(grammar(), str)
9788     if res == nil then
9789         return writer.error(
9790             format("Parser `%s` failed to process the input text.", name),
9791             format("Here are the first 20 characters of the remaining "
9792                 .. "unprocessed text: `%s`.", str:sub(1,20))
9793         )
9794     else
9795         return res
9796     end
9797 end
9798 end
9799
9800 self.create_parser("parse_blocks",
9801                   function()

```

```

9802         return parsers.blocks
9803     end, true)
9804
9805     self.create_parser("parse_blocks_nested",
9806         function()
9807             return parsers.blocks_nested
9808         end, false)
9809
9810     self.create_parser("parse_inlines",
9811         function()
9812             return parsers.inlines
9813         end, false)
9814
9815     self.create_parser("parse_inlines_no_inline_note",
9816         function()
9817             return parsers.inlines_no_inline_note
9818         end, false)
9819
9820     self.create_parser("parse_inlines_no_html",
9821         function()
9822             return parsers.inlines_no_html
9823         end, false)
9824
9825     self.create_parser("parse_inlines_nbsp",
9826         function()
9827             return parsers.inlines_nbsp
9828         end, false)
9829     self.create_parser("parse_inlines_no_link_or_emphasis",
9830         function()
9831             return parsers.inlines_no_link_or_emphasis
9832         end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

9833     parsers.minimally_indented_blankline
9834         = parsers.check_minimal_indent * (parsers.blankline / "")
9835
9836     parsers.minimally_indented_block
9837         = parsers.check_minimal_indent * V("Block")
9838
9839     parsers.minimally_indented_block_or_paragraph
9840         = parsers.check_minimal_indent * V("BlockOrParagraph")
9841
9842     parsers.minimally_indented_paragraph
9843         = parsers.check_minimal_indent * V("Paragraph")
9844

```

```

9845 parsers.minimally_indented_plain
9846     = parsers.check_minimal_indent * V("Plain")
9847
9848 parsers.minimally_indented_par_or_plain
9849     = parsers.minimally_indented_paragraph
9850     + parsers.minimally_indented_plain
9851
9852 parsers.minimally_indented_par_or_plain_no_blank
9853     = parsers.minimally_indented_par_or_plain
9854     - parsers.minimally_indented_blankline
9855
9856 parsers.minimally_indented_ref
9857     = parsers.check_minimal_indent * V("Reference")
9858
9859 parsers.minimally_indented_blank
9860     = parsers.check_minimal_indent * V("Blank")
9861
9862 parsers.conditionally_indented_blankline
9863     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
9864
9865 parsers.minimally_indented_ref_or_block
9866     = parsers.minimally_indented_ref
9867     + parsers.minimally_indented_block
9868     - parsers.minimally_indented_blankline
9869
9870 parsers.minimally_indented_ref_or_block_or_par
9871     = parsers.minimally_indented_ref
9872     + parsers.minimally_indented_block_or_paragraph
9873     - parsers.minimally_indented_blankline
9874

```

The following pattern parses the properly indented content that follows the initial container start.

```

9875
9876 function parsers.separator_loop(separated_block, paragraph,
9877                                block_separator, paragraph_separator)
9878     return separated_block
9879         + block_separator
9880         * paragraph
9881         * separated_block
9882         + paragraph_separator
9883         * paragraph
9884 end
9885
9886 function parsers.create_loop_body_pair(separated_block, paragraph,
9887                                       block_separator,
9888                                       paragraph_separator)

```

```

9889     return {
9890         block = parsers.separator_loop(separated_block, paragraph,
9891                                     block_separator, block_separator),
9892         par = parsers.separator_loop(separated_block, paragraph,
9893                                    block_separator, paragraph_separator)
9894     }
9895 end
9896
9897 parsers.block_sep_group = function(blank)
9898     return blank^0 * parsers.eof
9899         + ( blank^2 / writer.paragraphsep
9900           + blank^0 / writer.interblocksep
9901         )
9902 end
9903
9904 parsers.par_sep_group = function(blank)
9905     return blank^0 * parsers.eof
9906         + blank^0 / writer.paragraphsep
9907 end
9908
9909 parsers.sep_group_no_output = function(blank)
9910     return blank^0 * parsers.eof
9911         + blank^0
9912 end
9913
9914 parsers.content_blank = parsers.minimally_indented_blankline
9915
9916 parsers.ref_or_block_separated
9917     = parsers.sep_group_no_output(parsers.content_blank)
9918     * ( parsers.minimally_indented_ref
9919       - parsers.content_blank)
9920     + parsers.block_sep_group(parsers.content_blank)
9921     * ( parsers.minimally_indented_block
9922       - parsers.content_blank)
9923
9924 parsers.loop_body_pair =
9925     parsers.create_loop_body_pair(
9926         parsers.ref_or_block_separated,
9927         parsers.minimally_indented_par_or_plain_no_blank,
9928         parsers.block_sep_group(parsers.content_blank),
9929         parsers.par_sep_group(parsers.content_blank))
9930
9931 parsers.content_loop = ( V("Block")
9932                        * parsers.loop_body_pair.block^0
9933                        + (V("Paragraph") + V("Plain")))
9934                        * parsers.ref_or_block_separated
9935                        * parsers.loop_body_pair.block^0

```

```

9936             + (V("Paragraph") + V("Plain"))
9937             * parsers.loop_body_pair.par^0)
9938             * parsers.content_blank^0
9939
9940 parsers.indented_content = function()
9941   return Ct( (V("Reference") + (parsers.blankline / ""))
9942             * parsers.content_blank^0
9943             * parsers.check_minimal_indent
9944             * parsers.content_loop
9945             + (V("Reference") + (parsers.blankline / ""))
9946             * parsers.content_blank^0
9947             + parsers.content_loop)
9948 end
9949
9950 parsers.add_indent = function(pattern, name, breakable)
9951   return Cg(Cmt( Cb("indent_info")
9952               * Ct(pattern)
9953               * ( #parsers.linechar -- check if starter is blank
9954                 * Cc(false) + Cc(true))
9955               * Cc(name)
9956               * Cc(breakable),
9957               process_starter_indent), "indent_info")
9958 end
9959

```

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

```

9960 if options.hashEnumerators then
9961   parsers.dig = parsers.digit + parsers.hash
9962 else
9963   parsers.dig = parsers.digit
9964 end
9965
9966 parsers.enumerator = function(delimiter_type, interrupting)
9967   local delimiter_range
9968   local allowed_end
9969   if interrupting then
9970     delimiter_range = P("1")
9971     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9972   else
9973     delimiter_range = parsers.dig * parsers.dig^-8
9974     allowed_end = C(parsers.spacechar^1)
9975                 + #(parsers.newline + parsers.eof)
9976   end
9977
9978   return parsers.check_trail
9979         * Ct(C(delimiter_range) * C(delimiter_type))

```

```

9980             * allowed_end
9981     end
9982
9983     parsers.starter = parsers.bullet(parsers.dash)
9984                   + parsers.bullet(parsers.asterisk)
9985                   + parsers.bullet(parsers.plus)
9986                   + parsers.enumerator(parsers.period)
9987                   + parsers.enumerator(parsers.rparent)
9988

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

9989     parsers.blockquote_start
9990     = parsers.check_trail
9991     * C(parsers.more)
9992     * C(parsers.spacechar^0)
9993
9994     parsers.blockquote_body
9995     = parsers.add_indent(parsers.blockquote_start, "bq", true)
9996     * parsers.indented_content()
9997     * remove_indent("bq")
9998
9999     if not options.breakableBlockquotes then
10000     parsers.blockquote_body
10001     = parsers.add_indent(parsers.blockquote_start, "bq", false)
10002     * parsers.indented_content()
10003     * remove_indent("bq")
10004     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.

```

10005     local function parse_content_part(content_part)
10006     local rope = util.rope_to_string(content_part)
10007     local parsed
10008     = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
10009     parsed.indent_info = nil
10010     return parsed
10011     end
10012

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10013     local collect_emphasis_content =
10014     function(t, opening_index, closing_index)
10015     local content = {}
10016

```

```

10017     local content_part = {}
10018     for i = opening_index, closing_index do
10019         local value = t[i]
10020
10021         if value.rendered ~= nil then
10022             content[#content + 1] = parse_content_part(content_part)
10023             content_part = {}
10024             content[#content + 1] = value.rendered
10025             value.rendered = nil
10026         else
10027             if value.type == "delimiter"
10028                 and value.element == "emphasis" then
10029                 if value.is_active then
10030                     content_part[#content_part + 1]
10031                         = string.rep(value.character, value.current_count)
10032                 end
10033             else
10034                 content_part[#content_part + 1] = value.content
10035             end
10036             value.content = ''
10037             value.is_active = false
10038         end
10039     end
10040
10041     if next(content_part) ~= nil then
10042         content[#content + 1] = parse_content_part(content_part)
10043     end
10044
10045     return content
10046 end
10047

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

10048 local function fill_emph(t, opening_index, closing_index)
10049     local content
10050         = collect_emphasis_content(t, opening_index + 1,
10051                                   closing_index - 1)
10052     t[opening_index + 1].is_active = true
10053     t[opening_index + 1].rendered = writer.emphasis(content)
10054 end
10055

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

10056 local function fill_strong(t, opening_index, closing_index)
10057     local content
10058         = collect_emphasis_content(t, opening_index + 1,

```



```

10059             closing_index - 1)
10060     t[opening_index + 1].is_active = true
10061     t[opening_index + 1].rendered = writer.strong(content)
10062 end
10063

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

10064 local function breaks_three_rule(opening_delimiter, closing_delimiter)
10065     return ( opening_delimiter.is_closing
10066             or closing_delimiter.is_opening)
10067     and (( opening_delimiter.original_count
10068           + closing_delimiter.original_count) % 3 == 0)
10069     and ( opening_delimiter.original_count % 3 ~= 0
10070         or closing_delimiter.original_count % 3 ~= 0)
10071 end
10072

```

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`.

```

10073 local find_emphasis_opener = function(t, bottom_index, latest_index,
10074                                     character, closing_delimiter)
10075     for i = latest_index, bottom_index, -1 do
10076         local value = t[i]
10077         if value.is_active and
10078            value.is_opening and
10079            value.type == "delimiter" and
10080            value.element == "emphasis" and
10081            (value.character == character) and
10082            (value.current_count > 0) then
10083             if not breaks_three_rule(value, closing_delimiter) then
10084                 return i
10085             end
10086         end
10087     end
10088 end
10089

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

10090 local function process_emphasis(t, opening_index, closing_index)
10091     for i = opening_index, closing_index do
10092         local value = t[i]
10093         if value.type == "delimiter" and value.element == "emphasis" then
10094             local delimiter_length = string.len(value.content)
10095             value.character = string.sub(value.content, 1, 1)
10096             value.current_count = delimiter_length

```

```

10097         value.original_count = delimiter_length
10098     end
10099 end
10100
10101 local openers_bottom = {
10102     ['*'] = {
10103         [true] = {opening_index, opening_index, opening_index},
10104         [false] = {opening_index, opening_index, opening_index}
10105     },
10106     ['_'] = {
10107         [true] = {opening_index, opening_index, opening_index},
10108         [false] = {opening_index, opening_index, opening_index}
10109     }
10110 }
10111
10112 local current_position = opening_index
10113 local max_position = closing_index
10114
10115 while current_position <= max_position do
10116     local value = t[current_position]
10117
10118     if value.type ~= "delimiter" or
10119        value.element ~= "emphasis" or
10120        not value.is_active or
10121        not value.is_closing or
10122        (value.current_count <= 0) then
10123         current_position = current_position + 1
10124         goto continue
10125     end
10126
10127     local character = value.character
10128     local is_opening = value.is_opening
10129     local closing_length_modulo_three = value.original_count % 3
10130
10131     local current_openers_bottom
10132         = openers_bottom[character][is_opening]
10133           [closing_length_modulo_three + 1]
10134
10135     local opener_position
10136         = find_emphasis_opener(t, current_openers_bottom,
10137                               current_position - 1, character, value)
10138
10139     if (opener_position == nil) then
10140         openers_bottom[character][is_opening]
10141             [closing_length_modulo_three + 1]
10142             = current_position
10143         current_position = current_position + 1

```

```

10144         goto continue
10145     end
10146
10147     local opening_delimiter = t[opener_position]
10148
10149     local current_opening_count = opening_delimiter.current_count
10150     local current_closing_count = t[current_position].current_count
10151
10152     if (current_opening_count >= 2)
10153         and (current_closing_count >= 2) then
10154         opening_delimiter.current_count = current_opening_count - 2
10155         t[current_position].current_count = current_closing_count - 2
10156         fill_strong(t, opener_position, current_position)
10157     else
10158         opening_delimiter.current_count = current_opening_count - 1
10159         t[current_position].current_count = current_closing_count - 1
10160         fill_emph(t, opener_position, current_position)
10161     end
10162
10163     ::continue::
10164 end
10165 end
10166
10167 parsers.delimiter_run = function(character)
10168     return (B(parsers.backslash * character) + -B(character))
10169         * character~1
10170         * -#character
10171 end
10172
10173 parsers.left_flanking_delimiter_run = function(character)
10174     return (B( parsers.any)
10175         * ( parsers.unicode.preceding_punctuation
10176           + parsers.unicode.preceding_whitespace)
10177         + -B(parsers.any))
10178     * parsers.delimiter_run(character)
10179     * parsers.unicode.following_punctuation
10180     + parsers.delimiter_run(character)
10181     * -( parsers.unicode.following_punctuation
10182       + parsers.unicode.following_whitespace
10183       + parsers.eof)
10184 end
10185
10186 parsers.right_flanking_delimiter_run = function(character)
10187     return parsers.unicode.preceding_punctuation
10188     * parsers.delimiter_run(character)
10189     * ( parsers.unicode.following_punctuation
10190       + parsers.unicode.following_whitespace

```

```

10191         + parsers.eof)
10192     + (B(parsers.any)
10193       * -( parsers.unicode.preceding_punctuation
10194         + parsers.unicode.preceding_whitespace))
10195     * parsers.delimiter_run(character)
10196 end
10197
10198 if options.underscores then
10199     parsers.emph_start
10200     = parsers.left_flanking_delimiter_run(parsers.asterisk)
10201     + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
10202       + ( parsers.unicode.preceding_punctuation
10203         * #parsers.right_flanking_delimiter_run(parsers.underscore)))
10204     * parsers.left_flanking_delimiter_run(parsers.underscore)
10205
10206     parsers.emph_end
10207     = parsers.right_flanking_delimiter_run(parsers.asterisk)
10208     + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
10209       + #( parsers.left_flanking_delimiter_run(parsers.underscore)
10210         * parsers.unicode.following_punctuation))
10211     * parsers.right_flanking_delimiter_run(parsers.underscore)
10212 else
10213     parsers.emph_start
10214     = parsers.left_flanking_delimiter_run(parsers.asterisk)
10215
10216     parsers.emph_end
10217     = parsers.right_flanking_delimiter_run(parsers.asterisk)
10218 end
10219
10220 parsers.emph_capturing_open_and_close
10221 = #parsers.emph_start * #parsers.emph_end
10222 * Ct( Cg(Cc("delimiter"), "type")
10223     * Cg(Cc("emphasis"), "element")
10224     * Cg(C(parsers.emph_start), "content")
10225     * Cg(Cc(true), "is_opening")
10226     * Cg(Cc(true), "is_closing"))
10227
10228 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
10229     * Cg(Cc("emphasis"), "element")
10230     * Cg(C(parsers.emph_start), "content")
10231     * Cg(Cc(true), "is_opening")
10232     * Cg(Cc(false), "is_closing"))
10233
10234 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
10235     * Cg(Cc("emphasis"), "element")
10236     * Cg(C(parsers.emph_end), "content")
10237     * Cg(Cc(false), "is_opening"))

```

```

10238             * Cg(Cc(true), "is_closing"))
10239
10240 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
10241                             + parsers.emph_capturing_open
10242                             + parsers.emph_capturing_close
10243
10244 parsers.emph_open = parsers.emph_capturing_open_and_close
10245                   + parsers.emph_capturing_open
10246
10247 parsers.emph_close = parsers.emph_capturing_open_and_close
10248                   + parsers.emph_capturing_close
10249

```

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

```

10250 -- List of references defined in the document
10251 local references
10252
10253 -- List of note references defined in the document
10254 parsers.rawnotes = {}
10255

```

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.

```

10256 function self.register_link(_, tag, url, title,
10257                             attributes)
10258     local normalized_tag = self.normalize_tag(tag)
10259     if references[normalized_tag] == nil then
10260         references[normalized_tag] = {
10261             url = url,
10262             title = title,
10263             attributes = attributes
10264         }
10265     end
10266     return ""
10267 end
10268

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

10269 function self.lookup_reference(tag)
10270     return references[self.normalize_tag(tag)]
10271 end
10272

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

10273 function self.lookup_note_reference(tag)

```

```

10274     return parsers.rawnotes[self.normalize_tag(tag)]
10275 end
10276
10277 parsers.title_s_direct_ref = parsers.squote
10278     * Cs((parsers.html_entities
10279         + ( parsers.anyescaped
10280           - parsers.squote
10281           - parsers.blankline^2))^0)
10282     * parsers.squote
10283
10284 parsers.title_d_direct_ref = parsers.dquote
10285     * Cs((parsers.html_entities
10286         + ( parsers.anyescaped
10287           - parsers.dquote
10288           - parsers.blankline^2))^0)
10289     * parsers.dquote
10290
10291 parsers.title_p_direct_ref = parsers.lparent
10292     * Cs((parsers.html_entities
10293         + ( parsers.anyescaped
10294           - parsers.lparent
10295           - parsers.rparent
10296           - parsers.blankline^2))^0)
10297     * parsers.rparent
10298
10299 parsers.title_direct_ref = parsers.title_s_direct_ref
10300     + parsers.title_d_direct_ref
10301     + parsers.title_p_direct_ref
10302
10303 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
10304     * Cg(parsers.url + Cc(""), "url")
10305     * parsers.spnl
10306     * Cg( parsers.title_direct_ref
10307         + Cc(""), "title")
10308     * parsers.spnl * parsers.rparent
10309
10310 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
10311     * Cg(parsers.url + Cc(""), "url")
10312     * parsers.spnlc
10313     * Cg(parsers.title + Cc(""), "title")
10314     * parsers.spnlc * parsers.rparent
10315
10316 parsers.empty_link = parsers.lbracket
10317     * parsers.rbracket
10318
10319 parsers.inline_link = parsers.link_text
10320     * parsers.inline_direct_ref

```

```

10321
10322 parsers.full_link = parsers.link_text
10323                 * parsers.link_label
10324
10325 parsers.shortcut_link = parsers.link_label
10326                 * -(parsers.empty_link + parsers.link_label)
10327
10328 parsers.collapsed_link = parsers.link_label
10329                 * parsers.empty_link
10330
10331 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
10332                 * Cg(Cc("inline"), "link_type")
10333                 + #(parsers.exclamation * parsers.full_link)
10334                 * Cg(Cc("full"), "link_type")
10335                 + #( parsers.exclamation
10336                 * parsers.collapsed_link)
10337                 * Cg(Cc("collapsed"), "link_type")
10338                 + #(parsers.exclamation * parsers.shortcut_link)
10339                 * Cg(Cc("shortcut"), "link_type")
10340                 + #(parsers.exclamation * parsers.empty_link)
10341                 * Cg(Cc("empty"), "link_type")
10342
10343 parsers.link_opening = #parsers.inline_link
10344                 * Cg(Cc("inline"), "link_type")
10345                 + #parsers.full_link
10346                 * Cg(Cc("full"), "link_type")
10347                 + #parsers.collapsed_link
10348                 * Cg(Cc("collapsed"), "link_type")
10349                 + #parsers.shortcut_link
10350                 * Cg(Cc("shortcut"), "link_type")
10351                 + #parsers.empty_link
10352                 * Cg(Cc("empty_link"), "link_type")
10353                 + #parsers.link_text
10354                 * Cg(Cc("link_text"), "link_type")
10355
10356 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
10357                 * Cg(Cc("note_inline"), "link_type")
10358
10359 parsers.raw_note_opening = #( parsers.lbracket
10360                 * parsers.circumflex
10361                 * parsers.link_label_body
10362                 * parsers.rbracket)
10363                 * Cg(Cc("raw_note"), "link_type")
10364
10365 local inline_note_element = Cg(Cc("note"), "element")
10366                 * parsers.note_opening
10367                 * Cg( parsers.circumflex

```

```

10368             * parsers.lbracket, "content")
10369
10370 local image_element = Cg(Cc("image"), "element")
10371     * parsers.image_opening
10372     * Cg( parsers.exclamation
10373         * parsers.lbracket, "content")
10374
10375 local note_element = Cg(Cc("note"), "element")
10376     * parsers.raw_note_opening
10377     * Cg( parsers.lbracket
10378         * parsers.circumflex, "content")
10379
10380 local link_element = Cg(Cc("link"), "element")
10381     * parsers.link_opening
10382     * Cg(parsers.lbracket, "content")
10383
10384 local opening_elements = parsers.fail
10385
10386 if options.inlineNotes then
10387     opening_elements = opening_elements + inline_note_element
10388 end
10389
10390 opening_elements = opening_elements + image_element
10391
10392 if options.notes then
10393     opening_elements = opening_elements + note_element
10394 end
10395
10396 opening_elements = opening_elements + link_element
10397
10398 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
10399     * Cg(Cc(true), "is_opening")
10400     * Cg(Cc(false), "is_closing")
10401     * opening_elements)
10402
10403 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
10404     * Cg(Cc("link"), "element")
10405     * Cg(Cc(false), "is_opening")
10406     * Cg(Cc(true), "is_closing")
10407     * ( Cg(Cc(true), "is_direct")
10408         * Cg( parsers.rbracket
10409             * #parsers.inline_direct_ref,
10410             "content")
10411         + Cg(Cc(false), "is_direct")
10412         * Cg(parsers.rbracket, "content")))
10413
10414 parsers.link_image_open_or_close = parsers.link_image_opening

```



```

10415                                     + parsers.link_image_closing
10416
10417   if options.html then
10418     parsers.link_emph_precedence = parsers.inticks
10419                                   + parsers.autolink
10420                                   + parsers.html_inline_tags
10421   else
10422     parsers.link_emph_precedence = parsers.inticks
10423                                   + parsers.autolink
10424   end
10425
10426   parsers.link_and_emph_endline = parsers.newline
10427                                   * ((parsers.check_minimal_indent
10428                                       * -V("EndlineExceptions")
10429                                       + parsers.check_optional_indent
10430                                       * -V("EndlineExceptions")
10431                                       * -V("ListStarter")) / "")
10432                                   * parsers.spacechar^0 / "\n"
10433
10434   parsers.link_and_emph_content
10435     = Ct( Cg(Cc("content"), "type")
10436           * Cg(Cs(( parsers.link_emph_precedence
10437                     + parsers.backslash * parsers.linechar
10438                     + parsers.link_and_emph_endline
10439                     + (parsers.linechar
10440                        - parsers.blankline^2
10441                        - parsers.link_image_open_or_close
10442                        - parsers.emph_open_or_close))^0), "content"))
10443
10444   parsers.link_and_emph_table
10445     = (parsers.link_image_opening + parsers.emph_open)
10446       * parsers.link_and_emph_content
10447       * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
10448          * parsers.link_and_emph_content)^1
10449
10450   Collect the content between the opening\_index and closing\_index in the delimiter
10451   table t.
10452   local function collect_link_content(t, opening_index, closing_index)
10453     local content = {}
10454     for i = opening_index, closing_index do
10455       content[#content + 1] = t[i].content
10456     end
10457     return util.rope_to_string(content)
10458   end
10459
10460   end
10461
10462   end

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```
10458 local function find_link_opener(t, bottom_index, latest_index)
10459   for i = latest_index, bottom_index, -1 do
10460     local value = t[i]
10461     if value.type == "delimiter" and
10462        value.is_opening and
10463        ( value.element == "link"
10464          or value.element == "image"
10465          or value.element == "note")
10466        and not value.removed then
10467       if value.is_active then
10468         return i
10469       end
10470       value.removed = true
10471       return nil
10472     end
10473   end
10474 end
10475
```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```
10476 local function find_next_link_closing_index(t, latest_index)
10477   for i = latest_index, #t do
10478     local value = t[i]
10479     if value.is_closing and
10480        value.element == "link" and
10481        not value.removed then
10482       return i
10483     end
10484   end
10485 end
10486
```

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.

```
10487 local function disable_previous_link_openers(t, opening_index)
10488   if t[opening_index].element == "image" then
10489     return
10490   end
10491
10492   for i = opening_index, 1, -1 do
10493     local value = t[i]
10494     if value.is_active and
10495        value.type == "delimiter" and
10496        value.is_opening and
```

```

10497         value.element == "link" then
10498         value.is_active = false
10499     end
10500 end
10501 end
10502

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

10503 local function disable_range(t, opening_index, closing_index)
10504     for i = opening_index, closing_index do
10505         local value = t[i]
10506         if value.is_active then
10507             value.is_active = false
10508             if value.type == "delimiter" then
10509                 value.removed = true
10510             end
10511         end
10512     end
10513 end
10514

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10515 local delete_parsed_content_in_range =
10516     function(t, opening_index, closing_index)
10517         for i = opening_index, closing_index do
10518             t[i].rendered = nil
10519         end
10520     end
10521

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

10522 local function empty_content_in_range(t, opening_index, closing_index)
10523     for i = opening_index, closing_index do
10524         t[i].content = ''
10525     end
10526 end
10527

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

10528 local function join_attributes(reference_attributes, own_attributes)
10529     local merged_attributes = {}
10530     for _, attribute in ipairs(reference_attributes or {}) do
10531         table.insert(merged_attributes, attribute)
10532     end

```

```

10533     for _, attribute in ipairs(own_attributes or {}) do
10534         table.insert(merged_attributes, attribute)
10535     end
10536     if next(merged_attributes) == nil then
10537         merged_attributes = nil
10538     end
10539     return merged_attributes
10540 end
10541

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

10542     local render_link_or_image =
10543         function(t, opening_index, closing_index, content_end_index,
10544             reference)
10545             process_emphasis(t, opening_index, content_end_index)
10546             local mapped = collect_emphasis_content(t, opening_index + 1,
10547                 content_end_index - 1)
10548
10549             local rendered = {}
10550             if (t[opening_index].element == "link") then
10551                 rendered = writer.link(mapped, reference.url,
10552                     reference.title, reference.attributes)
10553             end
10554
10555             if (t[opening_index].element == "image") then
10556                 rendered = writer.image(mapped, reference.url, reference.title,
10557                     reference.attributes)
10558             end
10559
10560             if (t[opening_index].element == "note") then
10561                 if (t[opening_index].link_type == "note_inline") then
10562                     rendered = writer.note(mapped)
10563                 end
10564                 if (t[opening_index].link_type == "raw_note") then
10565                     rendered = writer.note(reference)
10566                 end
10567             end
10568
10569             t[opening_index].rendered = rendered
10570             delete_parsed_content_in_range(t, opening_index + 1,
10571                 closing_index)
10572             empty_content_in_range(t, opening_index, closing_index)
10573             disable_previous_link_openers(t, opening_index)
10574             disable_range(t, opening_index, closing_index)
10575         end
10576

```

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.

```
10577 local resolve_inline_following_content =
10578   function(t, closing_index, match_reference, match_link_attributes)
10579     local content = ""
10580     for i = closing_index + 1, #t do
10581       content = content .. t[i].content
10582     end
10583
10584     local matching_content = parsers.succeed
10585
10586     if match_reference then
10587       matching_content = matching_content
10588         * parsers.inline_direct_ref_inside
10589     end
10590
10591     if match_link_attributes then
10592       matching_content = matching_content
10593         * Cg(Ct(parsers.attributes^-1), "attributes")
10594     end
10595
10596     local matched = lpeg.match(Ct( matching_content
10597                                   * Cg(Cp(), "end_position")), content)
10598
10599     local matched_count = matched.end_position - 1
10600     for i = closing_index + 1, #t do
10601       local value = t[i]
10602
10603       local chars_left = matched_count
10604       matched_count = matched_count - #value.content
10605
10606       if matched_count <= 0 then
10607         value.content = value.content:sub(chars_left + 1)
10608         break
10609       end
10610
10611       value.content = ''
10612       value.is_active = false
10613     end
10614
10615     local attributes = matched.attributes
10616     if attributes == nil or next(attributes) == nil then
10617       attributes = nil
10618     end
10619
10620     return {
```

```

10621         url = matched.url or "",
10622         title = matched.title or "",
10623         attributes = attributes
10624     }
10625 end
10626

```

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.

```

10627 local function resolve_inline_link(t, opening_index, closing_index)
10628     local inline_content
10629     = resolve_inline_following_content(t, closing_index, true,
10630                                     t.match_link_attributes)
10631     render_link_or_image(t, opening_index, closing_index,
10632                         closing_index, inline_content)
10633 end
10634

```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```

10635 local resolve_note_inline_link =
10636     function(t, opening_index, closing_index)
10637         local inline_content
10638         = resolve_inline_following_content(t, closing_index,
10639                                         false, false)
10640         render_link_or_image(t, opening_index, closing_index,
10641                             closing_index, inline_content)
10642     end
10643

```

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.

```

10644 local function resolve_shortcut_link(t, opening_index, closing_index)
10645     local content
10646     = collect_link_content(t, opening_index + 1, closing_index - 1)
10647     local r = self.lookup_reference(content)
10648
10649     if r then
10650         local inline_content
10651         = resolve_inline_following_content(t, closing_index, false,
10652                                         t.match_link_attributes)
10653         r.attributes
10654         = join_attributes(r.attributes, inline_content.attributes)
10655         render_link_or_image(t, opening_index, closing_index,
10656                             closing_index, r)
10657     end

```

```
10658 end
10659
```

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.

```
10660 local function resolve_raw_note_link(t, opening_index, closing_index)
10661     local content
10662     = collect_link_content(t, opening_index + 1, closing_index - 1)
10663     local r = self.lookup_note_reference(content)
10664
10665     if r then
10666         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
10667         render_link_or_image(t, opening_index, closing_index,
10668                             closing_index, parsed_ref)
10669     end
10670 end
10671
```

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.

```
10672 local function resolve_full_link(t, opening_index, closing_index)
10673     local next_link_closing_index
10674     = find_next_link_closing_index(t, closing_index + 4)
10675     local next_link_content
10676     = collect_link_content(t, closing_index + 3,
10677                           next_link_closing_index - 1)
10678     local r = self.lookup_reference(next_link_content)
10679
10680     if r then
10681         local inline_content
10682         = resolve_inline_following_content(t, next_link_closing_index,
10683                                           false,
10684                                           t.match_link_attributes)
10685         r.attributes
10686         = join_attributes(r.attributes, inline_content.attributes)
10687         render_link_or_image(t, opening_index, next_link_closing_index,
10688                             closing_index, r)
10689     end
10690 end
10691
```

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.

```
10692 local function resolve_collapsed_link(t, opening_index, closing_index)
10693     local next_link_closing_index
10694     = find_next_link_closing_index(t, closing_index + 4)
10695     local content
```

```

10696     = collect_link_content(t, opening_index + 1, closing_index - 1)
10697 local r = self.lookup_reference(content)
10698
10699 if r then
10700     local inline_content
10701         = resolve_inline_following_content(t, closing_index, false,
10702                                           t.match_link_attributes)
10703     r.attributes
10704         = join_attributes(r.attributes, inline_content.attributes)
10705     render_link_or_image(t, opening_index, next_link_closing_index,
10706                        closing_index, r)
10707 end
10708 end
10709

```

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.

```

10710 local function process_links_and_emphasis(t)
10711     for _,value in ipairs(t) do
10712         value.is_active = true
10713     end
10714
10715     for i,value in ipairs(t) do
10716         if not value.is_closing
10717             or value.type ~= "delimiter"
10718             or not ( value.element == "link"
10719                    or value.element == "image"
10720                    or value.element == "note")
10721             or value.removed then
10722             goto continue
10723         end
10724
10725         local opener_position = find_link_opener(t, 1, i - 1)
10726         if (opener_position == nil) then
10727             goto continue
10728         end
10729
10730         local opening_delimiter = t[opener_position]
10731         opening_delimiter.removed = true
10732
10733         local link_type = opening_delimiter.link_type
10734
10735         if (link_type == "inline") then
10736             resolve_inline_link(t, opener_position, i)
10737         end

```



```

10738     if (link_type == "shortcut") then
10739         resolve_shortcut_link(t, opener_position, i)
10740     end
10741     if (link_type == "full") then
10742         resolve_full_link(t, opener_position, i)
10743     end
10744     if (link_type == "collapsed") then
10745         resolve_collapsed_link(t, opener_position, i)
10746     end
10747     if (link_type == "note_inline") then
10748         resolve_note_inline_link(t, opener_position, i)
10749     end
10750     if (link_type == "raw_note") then
10751         resolve_raw_note_link(t, opener_position, i)
10752     end
10753
10754     ::continue::
10755 end
10756
10757 t[#t].content = t[#t].content:gsub("%s*$", "")
10758
10759 process_emphasis(t, 1, #t)
10760 local final_result = collect_emphasis_content(t, 1, #t)
10761 return final_result
10762 end
10763
10764 function self.defer_link_and_emphasis_processing(delimiter_table)
10765     return writer.defer_call(function()
10766         return process_links_and_emphasis(delimiter_table)
10767     end)
10768 end
10769

```

### 3.1.6.8 Inline Elements (local)

```

10770 parsers.Str      = ( parsers.normalchar
10771                     * (parsers.normalchar + parsers.at)^0)
10772                   / writer.string
10773
10774 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
10775                   / writer.string
10776
10777 parsers.Ellipsis = P("...") / writer.ellipsis
10778
10779 parsers.Smart    = parsers.Ellipsis
10780
10781 parsers.Code     = parsers.inticks / writer.code

```

```

10782
10783   if options.blankBeforeBlockquote then
10784     parsers.bqstart = parsers.fail
10785   else
10786     parsers.bqstart = parsers.blockquote_start
10787   end
10788
10789   if options.blankBeforeHeading then
10790     parsers.headerstart = parsers.fail
10791   else
10792     parsers.headerstart = parsers.atx_heading
10793   end
10794
10795   if options.blankBeforeList then
10796     parsers.interrupting_bullets = parsers.fail
10797     parsers.interrupting_enumerators = parsers.fail
10798   else
10799     parsers.interrupting_bullets
10800       = parsers.bullet(parsers.dash, true)
10801       + parsers.bullet(parsers.asterisk, true)
10802       + parsers.bullet(parsers.plus, true)
10803
10804     parsers.interrupting_enumerators
10805       = parsers.enumerator(parsers.period, true)
10806       + parsers.enumerator(parsers.rparent, true)
10807   end
10808
10809   if options.html then
10810     parsers.html_interrupting
10811       = parsers.check_trail
10812       * ( parsers.html_incomplete_open_tag
10813         + parsers.html_incomplete_close_tag
10814         + parsers.html_incomplete_open_special_tag
10815         + parsers.html_comment_start
10816         + parsers.html_cdatasection_start
10817         + parsers.html_declaration_start
10818         + parsers.html_instruction_start
10819         - parsers.html_close_special_tag
10820         - parsers.html_empty_special_tag)
10821   else
10822     parsers.html_interrupting = parsers.fail
10823   end
10824
10825   parsers.ListStarter = parsers.starter
10826
10827   parsers.EndlineExceptions
10828       = parsers.blankline -- paragraph break

```

```

10829         + parsers.eof          -- end of document
10830         + parsers.bqstart
10831         + parsers.thematic_break_lines
10832         + parsers.interrupting_bullets
10833         + parsers.interrupting_enumerators
10834         + parsers.headerstart
10835         + parsers.html_interrupting
10836
10837 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
10838
10839 parsers.endline = parsers.newline
10840                 * (parsers.check_minimal_indent
10841                   * -V("EndlineExceptions")
10842                   + parsers.check_optional_indent
10843                   * -V("EndlineExceptions")
10844                   * -V("ListStarter")) / function(_) return end
10845                 * parsers.spacechar^0
10846
10847 parsers.Endline = parsers.endline
10848                 / writer.soft_line_break
10849
10850 parsers.EndlineNoSub = parsers.endline
10851
10852 parsers.NoSoftLineBreakEndline
10853                 = parsers.newline
10854                 * (parsers.check_minimal_indent
10855                   * -V("NoSoftLineBreakEndlineExceptions")
10856                   + parsers.check_optional_indent
10857                   * -V("NoSoftLineBreakEndlineExceptions")
10858                   * -V("ListStarter"))
10859                 * parsers.spacechar^0
10860                 / writer.space
10861
10862 parsers.EndlineBreak = parsers.backslash * parsers.endline
10863                       / writer.hard_line_break
10864
10865 parsers.OptionalIndent
10866                 = parsers.spacechar^1 / writer.space
10867
10868 parsers.Space     = parsers.spacechar^2 * parsers.endline
10869                       / writer.hard_line_break
10870                 + parsers.spacechar^1
10871                 * parsers.endline^-1
10872                 * parsers.eof / self.expandtabs
10873                 + parsers.spacechar^1 * parsers.endline
10874                       / writer.soft_line_break
10875                 + parsers.spacechar^1

```

```

10876             * -parsers.newline / self.expandtabs
10877             + parsers.spacechar^1
10878
10879 parsers.NoSoftLineBreakSpace
10880             = parsers.spacechar^2 * parsers.endline
10881               / writer.hard_line_break
10882             + parsers.spacechar^1
10883             * parsers.endline^-1
10884             * parsers.eof / self.expandtabs
10885             + parsers.spacechar^1 * parsers.endline
10886               / writer.soft_line_break
10887             + parsers.spacechar^1
10888             * -parsers.newline / self.expandtabs
10889             + parsers.spacechar^1
10890
10891 parsers.NonbreakingEndline
10892             = parsers.endline
10893             / writer.nbsp
10894
10895 parsers.NonbreakingSpace
10896             = parsers.spacechar^2 * parsers.endline
10897               / writer.nbsp
10898             + parsers.spacechar^1
10899             * parsers.endline^-1 * parsers.eof / ""
10900             + parsers.spacechar^1 * parsers.endline
10901               * parsers.optionalspace
10902               / writer.nbsp
10903             + parsers.spacechar^1 * parsers.optionalspace
10904               / writer.nbsp
10905

```

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.

```

10906 function self.auto_link_url(url, attributes)
10907   return writer.link(writer.escape(url),
10908                     url, nil, attributes)
10909 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.

```

10910 function self.auto_link_email(email, attributes)
10911   return writer.link(writer.escape(email),
10912                     "mailto:".email,
10913                     nil, attributes)
10914 end

```

```

10915
10916 parsers.AutoLinkUrl = parsers.auto_link_url
10917                       / self.auto_link_url
10918
10919 parsers.AutoLinkEmail
10920                       = parsers.auto_link_email
10921                       / self.auto_link_email
10922
10923 parsers.AutoLinkRelativeReference
10924                       = parsers.auto_link_relative_reference
10925                       / self.auto_link_url
10926
10927 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
10928                       / self.defer_link_and_emphasis_processing
10929
10930 parsers.EscapedChar = parsers.backslash
10931                       * C(parsers.escapable) / writer.string
10932
10933 parsers.InlineHtml = Cs(parsers.html_inline_comment)
10934                       / writer.inline_html_comment
10935                       + Cs(parsers.html_any_empty_inline_tag
10936                           + parsers.html_inline_instruction
10937                           + parsers.html_inline_cdatasection
10938                           + parsers.html_inline_declaration
10939                           + parsers.html_any_open_inline_tag
10940                           + parsers.html_any_close_tag)
10941                       / writer.inline_html_tag
10942
10943 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

10944 parsers.DisplayHtml = Cs(parsers.check_trail
10945                          * ( parsers.html_comment
10946                              + parsers.html_special_block
10947                              + parsers.html_block
10948                              + parsers.html_any_block
10949                              + parsers.html_instruction
10950                              + parsers.html_cdatasection
10951                              + parsers.html_declaration))
10952                          / writer.block_html_element
10953
10954 parsers.indented_non_blank_line = parsers.indentedline
10955                                 - parsers.blankline
10956
10957 parsers.Verbatim
10958     = Cs( parsers.check_code_trail

```

```

10959     * (parsers.line - parsers.blankline)
10960     * (( parsers.check_minimal_blank_indent_and_full_code_trail
10961     * parsers.blankline)^0
10962     * ( (parsers.check_minimal_indent / "")
10963     * parsers.check_code_trail
10964     * (parsers.line - parsers.blankline))^1)^0)
10965     / self.expandtabs / writer.verbatim
10966
10967     parsers.Blockquote = parsers.blockquote_body
10968                       / writer.blockquote
10969
10970     parsers.ThematicBreak = parsers.thematic_break_lines
10971                           / writer.thematic_break
10972
10973     parsers.Reference     = parsers.define_reference_parser
10974                           / self.register_link
10975
10976     parsers.Paragraph     = parsers.freeze_trail
10977                           * (Ct((parsers.Inline)^1)
10978                           * (parsers.newline + parsers.eof)
10979                           * parsers.unfreeze_trail
10980                           / writer.paragraph)
10981
10982     parsers.Plain         = parsers.nonindentspace * Ct(parsers.Inline^1)
10983                           / writer.plain

```

### 3.1.6.10 Lists (local)

```

10984
10985     if options.taskLists then
10986         parsers.tickbox = ( parsers.ticked_box
10987                             + parsers.halfticked_box
10988                             + parsers.unticked_box
10989                             ) / writer.tickbox
10990     else
10991         parsers.tickbox = parsers.fail
10992     end
10993
10994     parsers.list_blank = parsers.conditionally_indented_blankline
10995
10996     parsers.ref_or_block_list_separated
10997     = parsers.sep_group_no_output(parsers.list_blank)
10998     * parsers.minimally_indented_ref
10999     + parsers.block_sep_group(parsers.list_blank)
11000     * parsers.minimally_indented_block
11001
11002     parsers.ref_or_block_non_separated

```

```

11003     = parsers.minimally_indented_ref
11004     + (parsers.succeed / writer.interblocksep)
11005     * parsers.minimally_indented_block
11006     - parsers.minimally_indented_blankline
11007
11008     parsers.tight_list_loop_body_pair =
11009     parsers.create_loop_body_pair(
11010     parsers.ref_or_block_non_separated,
11011     parsers.minimally_indented_par_or_plain_no_blank,
11012     (parsers.succeed / writer.interblocksep),
11013     (parsers.succeed / writer.paragraphsep))
11014
11015     parsers.loose_list_loop_body_pair =
11016     parsers.create_loop_body_pair(
11017     parsers.ref_or_block_list_separated,
11018     parsers.minimally_indented_par_or_plain,
11019     parsers.block_sep_group(parsers.list_blank),
11020     parsers.par_sep_group(parsers.list_blank))
11021
11022     parsers.tight_list_content_loop
11023     = V("Block")
11024     * parsers.tight_list_loop_body_pair.block^0
11025     + (V("Paragraph") + V("Plain"))
11026     * parsers.ref_or_block_non_separated
11027     * parsers.tight_list_loop_body_pair.block^0
11028     + (V("Paragraph") + V("Plain"))
11029     * parsers.tight_list_loop_body_pair.par^0
11030
11031     parsers.loose_list_content_loop
11032     = V("Block")
11033     * parsers.loose_list_loop_body_pair.block^0
11034     + (V("Paragraph") + V("Plain"))
11035     * parsers.ref_or_block_list_separated
11036     * parsers.loose_list_loop_body_pair.block^0
11037     + (V("Paragraph") + V("Plain"))
11038     * parsers.loose_list_loop_body_pair.par^0
11039
11040     parsers.list_item_tightness_condition
11041     = -( parsers.list_blank^0
11042     * parsers.minimally_indented_ref_or_block_or_par)
11043     * remove_indent("li")
11044     + remove_indent("li")
11045     * parsers.fail
11046
11047     parsers.indented_content_tight
11048     = Ct( (parsers.blankline / "")
11049     * #parsers.list_blank

```

```

11050     * remove_indent("li")
11051     + ( (V("Reference") + (parsers.blankline / ""))
11052     * parsers.check_minimal_indent
11053     * parsers.tight_list_content_loop
11054     + (V("Reference") + (parsers.blankline / ""))
11055     + (parsers.checkbox^-1 / writer.escape)
11056     * parsers.tight_list_content_loop
11057     )
11058     * parsers.list_item_tightness_condition)
11059
11060 parsers.indented_content_loose
11061 = Ct( (parsers.blankline / "")
11062     * #parsers.list_blank
11063     + ( (V("Reference") + (parsers.blankline / ""))
11064     * parsers.check_minimal_indent
11065     * parsers.loose_list_content_loop
11066     + (V("Reference") + (parsers.blankline / ""))
11067     + (parsers.checkbox^-1 / writer.escape)
11068     * parsers.loose_list_content_loop))
11069
11070 parsers.TightListItem = function(starter)
11071     return -parsers.ThematicBreak
11072     * parsers.add_indent(starter, "li")
11073     * parsers.indented_content_tight
11074 end
11075
11076 parsers.LooseListItem = function(starter)
11077     return -parsers.ThematicBreak
11078     * parsers.add_indent(starter, "li")
11079     * parsers.indented_content_loose
11080     * remove_indent("li")
11081 end
11082
11083 parsers.BulletListOfType = function(bullet_type)
11084     local bullet = parsers.bullet(bullet_type)
11085     return ( Ct( parsers.TightListItem(bullet)
11086         * ( (parsers.check_minimal_indent / "")
11087         * parsers.TightListItem(bullet)
11088         )^0
11089         )
11090     * Cc(true)
11091     * -#( (parsers.list_blank^0 / "")
11092     * parsers.check_minimal_indent
11093     * (bullet - parsers.ThematicBreak)
11094     )
11095     + Ct( parsers.LooseListItem(bullet)
11096     * ( (parsers.list_blank^0 / "")

```



```

11097             * (parsers.check_minimal_indent / "")
11098             * parsers.LooseListItem(bullet)
11099             )^0
11100         )
11101         * Cc(false)
11102     ) / writer.bulletlist
11103 end
11104
11105 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
11106                   + parsers.BulletListOfType(parsers.asterisk)
11107                   + parsers.BulletListOfType(parsers.plus)
11108
11109 local function ordered_list(items,tight,starter)
11110     local startnum = starter[2][1]
11111     if options.startNumber then
11112         startnum = tonumber(startnum) or 1 -- fallback for '#'
11113         if startnum ~= nil then
11114             startnum = math.floor(startnum)
11115         end
11116     else
11117         startnum = nil
11118     end
11119     return writer.orderedlist(items,tight,startnum)
11120 end
11121
11122 parsers.OrderedListOfType = function(delimiter_type)
11123     local enumerator = parsers.enumerator(delimiter_type)
11124     return Cg(enumerator, "listtype")
11125             * (Ct( parsers.TightListItem(Cb("listtype"))
11126                 * ( (parsers.check_minimal_indent / "")
11127                   * parsers.TightListItem(enumerator))^0)
11128             * Cc(true)
11129             * -#((parsers.list_blank^0 / "")
11130               * parsers.check_minimal_indent * enumerator)
11131 + Ct( parsers.LooseListItem(Cb("listtype"))
11132     * ((parsers.list_blank^0 / "")
11133       * (parsers.check_minimal_indent / "")
11134       * parsers.LooseListItem(enumerator))^0)
11135     * Cc(false)
11136     ) * Ct(Cb("listtype")) / ordered_list
11137 end
11138
11139 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
11140                   + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

11141 parsers.Blank          = parsers.blankline / ""
11142                        + V("Reference")

```

### 3.1.6.12 Headings (local)

```

11143 function parsers.parse_heading_text(s)
11144     local inlines = self.parser_functions.parse_inlines(s)
11145     local flatten_inlines = self.writer.flatten_inlines
11146     self.writer.flatten_inlines = true
11147     local flat_text = self.parser_functions.parse_inlines(s)
11148     flat_text = util.ropetostring(flat_text)
11149     self.writer.flatten_inlines = flatten_inlines
11150     return {flat_text, inlines}
11151 end
11152
11153 -- parse atx header
11154 parsers.AtxHeading = parsers.check_trail_no_rem
11155                    * Cg(parsers.heading_start, "level")
11156                    * ((C( parsers.optionalspace
11157                          * parsers.hash^0
11158                          * parsers.optionalspace
11159                          * parsers.newline)
11160                      + parsers.spacechar^1
11161                      * C(parsers.line))
11162                    / strip_atx_end
11163                    / parsers.parse_heading_text)
11164                    * Cb("level")
11165                    / writer.heading
11166
11167 parsers.heading_line = parsers.linechar^1
11168                    - parsers.thematic_break_lines
11169
11170 parsers.heading_text = parsers.heading_line
11171                    * ( (V("Endline") / "\n")
11172                      * ( parsers.heading_line
11173                        - parsers.heading_level))^0
11174                    * parsers.newline^-1
11175
11176 parsers.SetextHeading = parsers.freeze_trail
11177                    * parsers.check_trail_no_rem
11178                    * #( parsers.heading_text
11179                      * parsers.check_minimal_indent
11180                      * parsers.check_trail
11181                      * parsers.heading_level)
11182                    * Cs(parsers.heading_text)
11183                    / parsers.parse_heading_text
11184                    * parsers.check_minimal_indent_and_trail

```

```

11185             * parsers.heading_level
11186             * parsers.newline
11187             * parsers.unfreeze_trail
11188             / writer.heading
11189
11190 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output.

```

11191 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.

```

11192     local walkable_syntax = (function(global_walkable_syntax)
11193         local local_walkable_syntax = {}
11194         for lhs, rule in pairs(global_walkable_syntax) do
11195             local_walkable_syntax[lhs] = util.table_copy(rule)
11196         end
11197         return local_walkable_syntax
11198     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.

```

11199     local current_extension_name = nil
11200     self.insert_pattern = function(selector, pattern, pattern_name)
11201         assert(pattern_name == nil or type(pattern_name) == "string")
11202         local _, _, lhs, pos, rhs
11203             = selector:find("^(%a+)%s+([%a%s]+%a)%s+(%a+)$")
11204         assert(lhs ~= nil,
11205             [[Expected selector in form ]]
11206             .. [[LHS (before|after|instead of) RHS", not "]]
11207             .. selector .. [[]])
11208         assert(walkable_syntax[lhs] ~= nil,
11209             [[Rule ]] .. lhs
11210             .. [[ -> ... does not exist in markdown grammar]])
11211         assert(pos == "before" or pos == "after" or pos == "instead of",
11212             [[Expected positional specifier "before", "after", ]]
11213             .. [[or "instead of", not "]]
11214             .. pos .. [[]])
11215         local rule = walkable_syntax[lhs]

```

```

11216     local index = nil
11217     for current_index, current_rhs in ipairs(rule) do
11218         if type(current_rhs) == "string" and current_rhs == rhs then
11219             index = current_index
11220             if pos == "after" then
11221                 index = index + 1
11222             end
11223             break
11224         end
11225     end
11226     assert(index ~= nil,
11227         [[Rule ]] .. lhs .. [[ -> ]] .. rhs
11228         .. [[ does not exist in markdown grammar]])
11229     local accountable_pattern
11230     if current_extension_name then
11231         accountable_pattern
11232         = {pattern, current_extension_name, pattern_name}
11233     else
11234         assert(type(pattern) == "string",
11235             [[reader->insert_pattern() was called outside ]]
11236             .. [[an extension with ]]
11237             .. [[a PEG pattern instead of a rule name]])
11238         accountable_pattern = pattern
11239     end
11240     if pos == "instead of" then
11241         rule[index] = accountable_pattern
11242     else
11243         table.insert(rule, index, accountable_pattern)
11244     end
11245 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.

```

11246     local syntax =
11247         { "Blocks",
11248
11249           Blocks = V("InitializeState")
11250                 * V("ExpectedJekyllData")
11251                 * 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.

```

11252         * ( V("Block")
11253           * ( V("Blank")^0 * parsers.eof
11254             + ( V("Blank")^2 / writer.paragraphsep
11255               + V("Blank")^0 / writer.interblocksep
11256             )

```

```

11257         )
11258     + ( V("Paragraph") + V("Plain") )
11259     * ( V("Blank")^0 * parsers.eof
11260         + ( V("Blank")^2 / writer.paragraphsep
11261             + V("Blank")^0 / writer.interblocksep
11262         )
11263     )
11264     * V("Block")
11265     * ( V("Blank")^0 * parsers.eof
11266         + ( V("Blank")^2 / writer.paragraphsep
11267             + V("Blank")^0 / writer.interblocksep
11268         )
11269     )
11270     + ( V("Paragraph") + V("Plain") )
11271     * ( V("Blank")^0 * parsers.eof
11272         + V("Blank")^0 / writer.paragraphsep
11273     )
11274     )^0,
11275
11276     ExpectedJekyllData = parsers.succeed,
11277
11278     Blank                = parsers.Blank,
11279     Reference            = parsers.Reference,
11280
11281     Blockquote          = parsers.Blockquote,
11282     Verbatim            = parsers.Verbatim,
11283     ThematicBreak       = parsers.ThematicBreak,
11284     BulletList          = parsers.BulletList,
11285     OrderedList         = parsers.OrderedList,
11286     DisplayHtml         = parsers.DisplayHtml,
11287     Heading             = parsers.Heading,
11288     Paragraph           = parsers.Paragraph,
11289     Plain               = parsers.Plain,
11290
11291     ListStarter         = parsers.ListStarter,
11292     EndlineExceptions   = parsers.EndlineExceptions,
11293     NoSoftLineBreakEndlineExceptions
11294         = parsers.NoSoftLineBreakEndlineExceptions,
11295
11296     Str                 = parsers.Str,
11297     Space               = parsers.Space,
11298     NoSoftLineBreakSpace
11299         = parsers.NoSoftLineBreakSpace,
11300     OptionalIndent     = parsers.OptionalIndent,
11301     Endline             = parsers.Endline,
11302     EndlineNoSub       = parsers.EndlineNoSub,
11303     NoSoftLineBreakEndline

```

```

11304                                     = parsers.NoSoftLineBreakEndline,
11305     EndlineBreak                       = parsers.EndlineBreak,
11306     LinkAndEmph                         = parsers.LinkAndEmph,
11307     Code                                = parsers.Code,
11308     AutoLinkUrl                         = parsers.AutoLinkUrl,
11309     AutoLinkEmail                       = parsers.AutoLinkEmail,
11310     AutoLinkRelativeReference           = parsers.AutoLinkRelativeReference,
11311                                     = parsers.AutoLinkRelativeReference,
11312     InlineHtml                          = parsers.InlineHtml,
11313     HtmlEntity                          = parsers.HtmlEntity,
11314     EscapedChar                         = parsers.EscapedChar,
11315     Smart                               = parsers.Smart,
11316     Symbol                              = parsers.Symbol,
11317     SpecialChar                         = parsers.fail,
11318     InitializeState                     = parsers.succeed,
11319 }

```

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]`.

```

11320     self.update_rule = function(rule_name, get_pattern)
11321         assert(current_extension_name ~= nil)
11322         assert(syntax[rule_name] ~= nil,
11323             [[Rule ]] .. rule_name
11324             .. [[ -> ... does not exist in markdown grammar]])
11325         local previous_pattern
11326         local extension_name
11327         if walkable_syntax[rule_name] then
11328             local previous_accountable_pattern
11329             = walkable_syntax[rule_name][1]
11330             previous_pattern = previous_accountable_pattern[1]
11331             extension_name
11332             = previous_accountable_pattern[2]
11333             .. ", " .. current_extension_name
11334         else
11335             previous_pattern = nil
11336             extension_name = current_extension_name
11337         end
11338         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

```

```

11339     if type(get_pattern) == "function" then
11340         pattern = get_pattern(previous_pattern)
11341     else
11342         assert(previous_pattern == nil,
11343             [[Rule ]] .. rule_name ..
11344             [[ has already been updated by ]] .. extension_name)
11345         pattern = get_pattern
11346     end
11347     local accountable_pattern = { pattern, extension_name, rule_name }
11348     walkable_syntax[rule_name] = { accountable_pattern }
11349 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.

```

11350     local special_characters = {}
11351     self.add_special_character = function(c)
11352         table.insert(special_characters, c)
11353         syntax.SpecialChar = S(table.concat(special_characters, ""))
11354     end
11355
11356     self.add_special_character("*")
11357     self.add_special_character("[")
11358     self.add_special_character("]")
11359     self.add_special_character("<")
11360     self.add_special_character("!")
11361     self.add_special_character("\\")

```

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

```

11362     self.initialize_named_group = function(name, value)
11363         local pattern = Ct("")
11364         if value ~= nil then
11365             pattern = pattern / value
11366         end
11367         syntax.InitializeState = syntax.InitializeState
11368             * Cg(pattern, name)
11369     end

```

Add a named group for indentation.

```

11370     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

11371     for _, extension in ipairs(extensions) do
11372         current_extension_name = extension.name
11373         extension.extend_writer(writer)
11374         extension.extend_reader(self)
11375     end
11376     current_extension_name = nil

```

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

```

11377     if options.debugExtensions then
11378         local sorted_lhs = {}
11379         for lhs, _ in pairs(walkable_syntax) do
11380             table.insert(sorted_lhs, lhs)
11381         end
11382         table.sort(sorted_lhs)
11383
11384         local output_lines = {"{"}
11385         for lhs_index, lhs in ipairs(sorted_lhs) do
11386             local encoded_lhs = util.encode_json_string(lhs)
11387             table.insert(output_lines, [[      ]] .. encoded_lhs .. [[: ]]])
11388             local rule = walkable_syntax[lhs]
11389             for rhs_index, rhs in ipairs(rule) do
11390                 local human_readable_rhs
11391                 if type(rhs) == "string" then
11392                     human_readable_rhs = rhs
11393                 else
11394                     local pattern_name
11395                     if rhs[3] then
11396                         pattern_name = rhs[3]
11397                     else
11398                         pattern_name = "Anonymous Pattern"
11399                     end
11400                     local extension_name = rhs[2]
11401                     human_readable_rhs = pattern_name .. [[ (]]
11402                                     .. extension_name .. [[:]]])
11403                 end
11404                 local encoded_rhs
11405                 = util.encode_json_string(human_readable_rhs)
11406                 local output_line = [[      ]] .. encoded_rhs
11407                 if rhs_index < #rule then
11408                     output_line = output_line .. ", "
11409                 end
11410                 table.insert(output_lines, output_line)
11411             end
11412             local output_line = "    ]"
11413             if lhs_index < #sorted_lhs then
11414                 output_line = output_line .. ", "

```



```

11415         end
11416         table.insert(output_lines, output_line)
11417     end
11418     table.insert(output_lines, "}")
11419
11420     local output = table.concat(output_lines, "\n")
11421     local output_filename = options.debugExtensionsFileName
11422     local output_file = assert(io.open(output_filename, "w"),
11423         [[Could not open file ]] .. output_filename
11424         .. [[ for writing]])
11425     assert(output_file:write(output))
11426     assert(output_file:close())
11427 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`.

```

11428     for lhs, rule in pairs(walkable_syntax) do
11429         syntax[lhs] = parsers.fail
11430         for _, rhs in ipairs(rule) do
11431             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.

```

11432         if type(rhs) == "string" then
11433             pattern = V(rhs)
11434         else
11435             pattern = rhs[1]
11436             if type(pattern) == "string" then
11437                 pattern = V(pattern)
11438             end
11439         end
11440         syntax[lhs] = syntax[lhs] + pattern
11441     end
11442 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.

```

11443     if options.underscores then
11444         self.add_special_character("_")
11445     end
11446
11447     if not options.codeSpans then
11448         syntax.Code = parsers.fail

```

```

11449     else
11450         self.add_special_character("`")
11451     end
11452
11453     if not options.html then
11454         syntax.DisplayHtml = parsers.fail
11455         syntax.InlineHtml = parsers.fail
11456         syntax.HtmlEntity = parsers.fail
11457     else
11458         self.add_special_character("&")
11459     end
11460
11461     if options.preserveTabs then
11462         options.stripIndent = false
11463     end
11464
11465     if not options.smartEllipses then
11466         syntax.Smart = parsers.fail
11467     else
11468         self.add_special_character(".")
11469     end
11470
11471     if not options.relativeReferences then
11472         syntax.AutoLinkRelativeReference = parsers.fail
11473     end
11474
11475     if options.contentLevel == "inline" then
11476         syntax[1] = "Inlines"
11477         syntax.Inlines = V("InitializeState")
11478             * parsers.Inline^0
11479             * ( parsers.spacing^0
11480             * parsers.eof / "" )
11481         syntax.Space = parsers.Space + parsers.blankline / writer.space
11482     end
11483
11484     local blocks_nested_t = util.table_copy(syntax)
11485     blocks_nested_t.ExpectedJekyllData = parsers.succeed
11486     parsers.blocks_nested = Ct(blocks_nested_t)
11487
11488     parsers.blocks = Ct(syntax)
11489
11490     local inlines_t = util.table_copy(syntax)
11491     inlines_t[1] = "Inlines"
11492     inlines_t.Inlines = V("InitializeState")
11493         * parsers.Inline^0
11494         * ( parsers.spacing^0
11495         * parsers.eof / "" )

```

```

11496     parsers.inlines = Ct(inlines_t)
11497
11498     local inlines_no_inline_note_t = util.table_copy(inlines_t)
11499     inlines_no_inline_note_t.InlineNote = parsers.fail
11500     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
11501
11502     local inlines_no_html_t = util.table_copy(inlines_t)
11503     inlines_no_html_t.DisplayHtml = parsers.fail
11504     inlines_no_html_t.InlineHtml = parsers.fail
11505     inlines_no_html_t.HtmlEntity = parsers.fail
11506     parsers.inlines_no_html = Ct(inlines_no_html_t)
11507
11508     local inlines_nbsp_t = util.table_copy(inlines_t)
11509     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
11510     inlines_nbsp_t.Space = parsers.NonbreakingSpace
11511     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
11512
11513     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
11514     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
11515     inlines_no_link_or_emphasis_t.EndlineExceptions
11516     = parsers.EndlineExceptions - parsers.eof
11517     parsers.inlines_no_link_or_emphasis
11518     = Ct(inlines_no_link_or_emphasis_t)

```

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

```

11519     return function(input)
Unicode-normalize the input.
11520         if options.unicodeNormalization then
11521             local form = options.unicodeNormalizationForm
11522             input = util.normalize(input, form)
11523         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.

```

11524         input = input:gsub("\r\n?", "\n")
11525         if input:sub(-1) ~= "\n" then
11526             input = input .. "\n"
11527         end

```

Clear the table of references.

```

11528         references = {}
11529         local document = self.parser_functions.parse_blocks(input)
11530         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`.

```

11531     local undosep_start, undosep_end
11532     local potential_secend_start, secend_start
11533     local potential_sep_start, sep_start
11534     while true do
11535         -- find a `writer->undosep`
11536         undosep_start, undosep_end
11537         = output:find(writer.undosep_text, 1, true)
11538         if undosep_start == nil then break end
11539         -- skip any preceding section ends
11540         secend_start = undosep_start
11541         while true do
11542             potential_secend_start = secend_start - #writer.secend_text
11543             if potential_secend_start < 1
11544                 or output:sub(potential_secend_start,
11545                             secend_start - 1) ~= writer.secend_text
11546             then
11547                 break
11548             end
11549             secend_start = potential_secend_start
11550         end
11551         -- find an immediately preceding
11552         -- block element / paragraph separator
11553         sep_start = secend_start
11554         potential_sep_start = sep_start - #writer.interblocksep_text
11555         if potential_sep_start >= 1
11556             and output:sub(potential_sep_start,
11557                             sep_start - 1) == writer.interblocksep_text
11558         then
11559             sep_start = potential_sep_start
11560         else
11561             potential_sep_start = sep_start - #writer.paragraphsep_text
11562             if potential_sep_start >= 1
11563                 and output:sub(potential_sep_start,
11564                             sep_start - 1) == writer.paragraphsep_text
11565             then
11566                 sep_start = potential_sep_start
11567             end
11568         end
11569         -- remove `writer->undosep` and immediately preceding
11570         -- block element / paragraph separator
11571         output = output:sub(1, sep_start - 1)
11572             .. output:sub(secend_start, undosep_start - 1)
11573             .. output:sub(undosep_end + 1)
11574     end
11575     return output
11576 end
11577 end

```

```
11578 return self
11579 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.

```
11580 M.extensions = {}
```

#### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
11581 M.extensions.bracketed_spans = function()
11582   return {
11583     name = "built-in bracketed_spans syntax extension",
11584     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.

```
11585       function self.span(s, attr)
11586         if self.flatten_inlines then return s end
11587         return {"\\markdownRendererBracketedSpanAttributeContextBegin",
11588             self.attributes(attr),
11589             s,
11590             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
11591     end
11592   end, extend_reader = function(self)
11593     local parsers = self.parsers
11594     local writer = self.writer
11595
11596     local span_label = parsers.lbracket
11597                       * (Cs((parsers.alphanumeric~1
11598                           + parsers.inticks
11599                           + parsers.autolink
11600                           + V("InlineHtml")
11601                           + ( parsers.backslash * parsers.backslash)
11602                           + ( parsers.backslash
11603                             * (parsers.lbracket + parsers.rbracket)
11604                             + V("Space") + V("Endline")
11605                             + (parsers.any
11606                               - ( parsers.newline
11607                                 + parsers.lbracket
11608                                 + parsers.rbracket
```

```

11609             + parsers.blankline^2))))^1)
11610         / self.parser_functions.parse_inlines)
11611         * parsers.rbracket
11612
11613     local Span = span_label
11614             * Ct(parsers.attributes)
11615             / writer.span
11616
11617     self.insert_pattern("Inline before LinkAndEmph",
11618                       Span, "Span")
11619 end
11620 }
11621 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.

```

11622 M.extensions.citations = function(citation_nbsps)
11623   return {
11624     name = "built-in citations syntax extension",
11625     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.

```

11626     function self.citations(text_cites, cites)
11627       local buffer = {}
11628       if self.flatten_inlines then
11629         for _,cite in ipairs(cites) do
11630           if cite.prenote then
11631             table.insert(buffer, {cite.prenote, " "})
11632           end

```

```

11633         table.insert(buffer, cite.name)
11634         if cite.postnote then
11635             table.insert(buffer, {" ", cite.postnote})
11636         end
11637     end
11638 else
11639     table.insert(buffer,
11640         {"\\markdownRenderer",
11641         text_cites and "TextCite" or "Cite",
11642         "{", #cites, "}"}))
11643     for _,cite in ipairs(cites) do
11644         table.insert(buffer,
11645             {cite.suppress_author and "-" or "+", "{",
11646             cite.prenote or "", "}{" ,
11647             cite.postnote or "", "}{" , cite.name, "}"}))
11648     end
11649 end
11650 return buffer
11651 end
11652 end, extend_reader = function(self)
11653     local parsers = self.parsers
11654     local writer = self.writer
11655
11656     local citation_chars
11657         = parsers.alphanumeric
11658         + S("#$%&-+<>~/_")
11659
11660     local citation_name
11661         = Cs(parsers.dash~-1) * parsers.at
11662         * Cs(citation_chars
11663             * ((( citation_chars
11664                 + parsers.internal_punctuation
11665                 - parsers.comma - parsers.semicolon)
11666             * -#(( parsers.internal_punctuation
11667                 - parsers.comma
11668                 - parsers.semicolon)^0
11669             * -( citation_chars
11670                 + parsers.internal_punctuation
11671                 - parsers.comma
11672                 - parsers.semicolon)))^0
11673             * citation_chars)^-1)
11674
11675     local citation_body_prenote
11676         = Cs((parsers.alphanumeric^1
11677             + parsers.bracketed
11678             + parsers.inticks
11679             + parsers.autolink

```

```

11680         + V("InlineHtml")
11681         + V("Space") + V("EndlineNoSub")
11682         + (parsers.anyescaped
11683           - ( parsers.newline
11684             + parsers.rbracket
11685             + parsers.blankline^2))
11686         - ( parsers.spnl
11687           * parsers.dash^-1
11688           * parsers.at))^1)
11689
11690     local citation_body_postnote
11691       = Cs((parsers.alphanumeric^1
11692           + parsers.bracketed
11693           + parsers.inticks
11694           + parsers.autolink
11695           + V("InlineHtml")
11696           + V("Space") + V("EndlineNoSub")
11697           + (parsers.anyescaped
11698             - ( parsers.newline
11699               + parsers.rbracket
11700               + parsers.semicolon
11701               + parsers.blankline^2))
11702           - (parsers.spnl * parsers.rbracket))^1)
11703
11704     local citation_body_chunk
11705       = ( citation_body_prenote
11706         * parsers.spnlc_sep
11707         + Cc("")
11708         * parsers.spnlc
11709         )
11710       * citation_name
11711       * ( parsers.internal_punctuation
11712         - parsers.semicolon)^-1
11713       * ( parsers.spnlc / function(_) return end
11714         * citation_body_postnote
11715         + Cc("")
11716         * parsers.spnlc
11717         )
11718
11719     local citation_body
11720       = citation_body_chunk
11721       * ( parsers.semicolon
11722         * parsers.spnlc
11723         * citation_body_chunk
11724       )^0
11725
11726     local citation_headless_body_postnote

```



```

11727         = Cs((parsers.alphanumeric^1
11728             + parsers.bracketed
11729             + parsers.inticks
11730             + parsers.autolink
11731             + V("InlineHtml")
11732             + V("Space") + V("Endline")
11733             + (parsers.anyescaped
11734               - ( parsers.newline
11735                 + parsers.rbracket
11736                 + parsers.at
11737                 + parsers.semicolon + parsers.blankline^2))
11738             - (parsers.spnl * parsers.rbracket))^0)
11739
11740     local citation_headless_body
11741         = citation_headless_body_postnote
11742         * ( parsers.semicolon
11743           * parsers.spnlc
11744           * citation_body_chunk
11745         )^0
11746
11747     local citations
11748         = function(text_cites, raw_cites)
11749         local function normalize(str)
11750             if str == "" then
11751                 str = nil
11752             else
11753                 str = (citation_nbsps and
11754                       self.parser_functions.parse_inlines_nbsp or
11755                       self.parser_functions.parse_inlines)(str)
11756             end
11757             return str
11758         end
11759
11760         local cites = {}
11761         for i = 1,#raw_cites,4 do
11762             cites[#cites+1] = {
11763                 prenote = normalize(raw_cites[i]),
11764                 suppress_author = raw_cites[i+1] == "-",
11765                 name = writer.identifier(raw_cites[i+2]),
11766                 postnote = normalize(raw_cites[i+3]),
11767             }
11768         end
11769         return writer.citations(text_cites, cites)
11770     end
11771
11772     local TextCitations
11773         = Ct((parsers.spnlc

```

```

11774         * Cc("")
11775         * citation_name
11776         * ((parsers.spnlc
11777           * parsers.lbracket
11778           * citation_headless_body
11779           * parsers.rbracket) + Cc(""))^1)
11780         / function(raw_cites)
11781           return citations(true, raw_cites)
11782         end
11783
11784     local ParenthesizedCitations
11785         = Ct((parsers.spnlc
11786           * parsers.lbracket
11787           * citation_body
11788           * parsers.rbracket)^1)
11789         / function(raw_cites)
11790           return citations(false, raw_cites)
11791         end
11792
11793     local Citations = TextCitations + ParenthesizedCitations
11794
11795     self.insert_pattern("Inline before LinkAndEmph",
11796                       Citations, "Citations")
11797
11798     self.add_special_character("@")
11799     self.add_special_character("-")
11800 end
11801 }
11802 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.

```
11803 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.

```

11804     local languages_json = (function()
11805         local base, prev, curr
11806         for _, pathname in ipairs{kpse.lookup(language_map,
11807                                           {all=true})} do
11808             local file = io.open(pathname, "r")
11809             if not file then goto continue end

```

```

11810     local input = assert(file:read("*a"))
11811     assert(file:close())
11812     local json = input:gsub('("[^\\n]-"):','[%1]=')
11813     curr = load("_ENV = {}; return "..json")()
11814     if type(curr) == "table" then
11815         if base == nil then
11816             base = curr
11817         else
11818             setmetatable(prev, { __index = curr })
11819         end
11820         prev = curr
11821     end
11822     ::continue::
11823 end
11824 return base or {}
11825 end)()
11826
11827 return {
11828     name = "built-in content_blocks syntax extension",
11829     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.

```

11830     function self.contentblock(src,suf,type,tit)
11831         if not self.is_writing then return "" end
11832         src = src..".."..suf
11833         suf = suf:lower()
11834         if type == "onlineimage" then
11835             return {"\\markdownRendererContentBlockOnlineImage{"..suf.."},"",
11836                 {"",self.string(src),"}",
11837                 {"",self.uri(src),"}",
11838                 {"",self.string(tit or ""),"}"}
11839         elseif languages_json[suf] then
11840             return {"\\markdownRendererContentBlockCode{"..suf.."},"",
11841                 {"",self.string(languages_json[suf]),"}",
11842                 {"",self.string(src),"}",
11843                 {"",self.uri(src),"}",
11844                 {"",self.string(tit or ""),"}"}
11845         else
11846             return {"\\markdownRendererContentBlock{"..suf.."},"",
11847                 {"",self.string(src),"}",
11848                 {"",self.uri(src),"}",
11849                 {"",self.string(tit or ""),"}"}
11850         end
11851     end

```

```

11852 end, extend_reader = function(self)
11853     local parsers = self.parsers
11854     local writer = self.writer
11855
11856     local contentblock_tail
11857         = parsers.optionaltitle
11858         * (parsers.newline + parsers.eof)
11859
11860     -- case insensitive online image suffix:
11861     local onlineimagesuffix
11862         = (function(...)
11863             local parser = nil
11864             for _, suffix in ipairs({...}) do
11865                 local pattern=nil
11866                 for i=1,#suffix do
11867                     local char=suffix:sub(i,i)
11868                     char = S(char:lower()..char:upper())
11869                     if pattern == nil then
11870                         pattern = char
11871                     else
11872                         pattern = pattern * char
11873                     end
11874                 end
11875                 if parser == nil then
11876                     parser = pattern
11877                 else
11878                     parser = parser + pattern
11879                 end
11880             end
11881             return parser
11882         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
11883
11884     -- online image url for iA Writer content blocks with
11885     -- mandatory suffix, allowing nested brackets:
11886     local onlineimageurl
11887         = (parsers.less
11888         * Cs((parsers.anyescaped
11889             - parsers.more
11890             - parsers.spacing
11891             - #(parsers.period
11892                 * onlineimagesuffix
11893                 * parsers.more
11894                 * contentblock_tail))^0)
11895         * parsers.period
11896         * Cs(onlineimagesuffix)
11897         * parsers.more
11898         + (Cs((parsers.inparens

```

```

11899         + (parsers.anyescaped
11900         - parsers.spacing
11901         - parsers.rparent
11902         - #(parsers.period
11903           * onlineimagesuffix
11904           * contentblock_tail)))^0)
11905         * parsers.period
11906         * Cs(onlineimagesuffix))
11907     ) * Cc("onlineimage")
11908
11909     -- filename for iA Writer content blocks with mandatory suffix:
11910     local localfilepath
11911         = parsers.slash
11912         * Cs((parsers.anyescaped
11913           - parsers.tab
11914           - parsers.newline
11915           - #(parsers.period
11916             * parsers.alphanumeric^1
11917             * contentblock_tail))^1)
11918         * parsers.period
11919         * Cs(parsers.alphanumeric^1)
11920         * Cc("localfile")
11921
11922     local ContentBlock
11923         = parsers.check_trail_no_rem
11924         * (localfilepath + onlineimageurl)
11925         * contentblock_tail
11926         / writer.contentblock
11927
11928     self.insert_pattern("Block before Blockquote",
11929                       ContentBlock, "ContentBlock")
11930 end
11931 }
11932 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.

```

11933 M.extensions.definition_lists = function(tight_lists)
11934   return {
11935     name = "built-in definition_lists syntax extension",
11936     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.

```
11937     local function dliitem(term, defs)
11938         local retVal = {"\\markdownRendererDlItem{",term,""}
11939         for _, def in ipairs(defs) do
11940             retVal[#retVal+1]
11941                 = {"\\markdownRendererDlDefinitionBegin ",def,
11942                     "\\markdownRendererDlDefinitionEnd "}
11943         end
11944         retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
11945         return retVal
11946     end
11947
11948     function self.definitionlist(items,tight)
11949         if not self.is_writing then return "" end
11950         local buffer = {}
11951         for _,item in ipairs(items) do
11952             buffer[#buffer + 1] = dliitem(item.term, item.definitions)
11953         end
11954         if tight and tight_lists then
11955             return {"\\markdownRendererDlBeginTight\n", buffer,
11956                 "\n\\markdownRendererDlEndTight"}
11957         else
11958             return {"\\markdownRendererDlBegin\n", buffer,
11959                 "\n\\markdownRendererDlEnd"}
11960         end
11961     end
11962     end, extend_reader = function(self)
11963         local parsers = self.parsers
11964         local writer = self.writer
11965
11966         local defstartchar = S("~:")
11967
11968         local defstart
11969             = parsers.check_trail_length(0) * defstartchar
11970             * #parsers.spacing
11971             * (parsers.tab + parsers.space^-3)
11972             + parsers.check_trail_length(1)
11973             * defstartchar * #parsers.spacing
11974             * (parsers.tab + parsers.space^-2)
11975             + parsers.check_trail_length(2)
11976             * defstartchar * #parsers.spacing
11977             * (parsers.tab + parsers.space^-1)
11978             + parsers.check_trail_length(3)
11979             * defstartchar * #parsers.spacing
11980
11981         local indented_line
```

```

11982     = (parsers.check_minimal_indent / "")
11983     * parsers.check_code_trail * parsers.line
11984
11985     local blank
11986     = parsers.check_minimal_blank_indent_and_any_trail
11987     * parsers.optionalspace * parsers.newline
11988
11989     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
11990
11991     local indented_blocks = function(bl)
11992     return Cs( bl
11993     * (blank^1 * (parsers.check_minimal_indent / ""))
11994     * parsers.check_code_trail * -parsers.blankline * bl)^0
11995     * (blank^1 + parsers.eof))
11996     end
11997
11998     local function definition_list_item(term, defs, _)
11999     return { term = self.parser_functions.parse_inlines(term),
12000             definitions = defs }
12001     end
12002
12003     local DefinitionListItemLoose
12004     = C(parsers.line) * blank^0
12005     * Ct((parsers.check_minimal_indent * (defstart
12006     * indented_blocks(dlchunk)
12007     / self.parser_functions.parse_blocks_nested))^1)
12008     * Cc(false) / definition_list_item
12009
12010     local DefinitionListItemTight
12011     = C(parsers.line)
12012     * Ct((parsers.check_minimal_indent * (defstart * dlchunk
12013     / self.parser_functions.parse_blocks_nested))^1)
12014     * Cc(true) / definition_list_item
12015
12016     local DefinitionList
12017     = ( Ct(DefinitionListItemLoose^1) * Cc(false)
12018     + Ct(DefinitionListItemTight^1)
12019     * (blank^0
12020     * -DefinitionListItemLoose * Cc(true))
12021     ) / writer.definitionlist
12022
12023     self.insert_pattern("Block after Heading",
12024     DefinitionList, "DefinitionList")
12025     end
12026 }
12027 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```
12028 M.extensions.fancy_lists = function()
12029   return {
12030     name = "built-in fancy_lists syntax extension",
12031     extend_writer = function(self)
12032       local options = self.options
12033
```

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.

```
12034     function self.fancylist(items,tight,startnum,numstyle,numdelim)
12035       if not self.is_writing then return "" end
12036       local buffer = {}
12037       local num = startnum
12038       for _,item in ipairs(items) do
12039         if item ~= "" then
12040           buffer[#buffer + 1] = self.fancyitem(item,num)
12041         end
12042         if num ~= nil and item ~= "" then
12043           num = num + 1
12044         end
12045       end
12046       local contents = util.intersperse(buffer,"\n")
12047       if tight and options.tightLists then
12048         return {"\\markdownRendererFancyOlBeginTight{",
12049           numstyle,"}{",numdelim,"}",contents,
```



```

12050             "\n\\markdownRendererFancy01EndTight "}
12051     else
12052         return {"\\markdownRendererFancy01Begin{",
12053             numstyle,"}{",numdelim,"}",contents,
12054             "\n\\markdownRendererFancy01End "}
12055     end
12056 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.

```

12057     function self.fancyitem(s,num)
12058         if num ~= nil then
12059             return {"\\markdownRendererFancy01ItemWithNumber{",num,"}",s,
12060                 "\\markdownRendererFancy01ItemEnd "}
12061         else
12062             return {"\\markdownRendererFancy01Item ",s,
12063                 "\\markdownRendererFancy01ItemEnd "}
12064         end
12065     end
12066 end, extend_reader = function(self)
12067     local parsers = self.parsers
12068     local options = self.options
12069     local writer = self.writer
12070
12071     local function combine_markers_and_delims(markers, delims)
12072         local markers_table = {}
12073         for _,marker in ipairs(markers) do
12074             local start_marker
12075             local continuation_marker
12076             if type(marker) == "table" then
12077                 start_marker = marker[1]
12078                 continuation_marker = marker[2]
12079             else
12080                 start_marker = marker
12081                 continuation_marker = marker
12082             end
12083             for _,delim in ipairs(delims) do
12084                 table.insert(markers_table,
12085                     {start_marker, continuation_marker, delim})
12086             end
12087         end
12088         return markers_table
12089     end
12090
12091     local function join_table_with_func(func, markers_table)
12092         local pattern = func(table.unpack(markers_table[1]))

```

```

12093     for i = 2, #markers_table do
12094         pattern = pattern + func(table.unpack(markers_table[i]))
12095     end
12096     return pattern
12097 end
12098
12099 local lowercase_letter_marker = R("az")
12100 local uppercase_letter_marker = R("AZ")
12101
12102 local roman_marker = function(chars)
12103     local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
12104     local l, x, v, i
12105         = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
12106     return m^-3
12107         * (c*m + c*d + d^-1 * c^-3)
12108         * (x*c + x*l + l^-1 * x^-3)
12109         * (i*x + i*v + v^-1 * i^-3)
12110 end
12111
12112 local lowercase_roman_marker
12113     = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
12114 local uppercase_roman_marker
12115     = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
12116
12117 local lowercase_opening_roman_marker = P("i")
12118 local uppercase_opening_roman_marker = P("I")
12119
12120 local digit_marker = parsers.dig * parsers.dig^-8
12121
12122 local markers = {
12123     {lowercase_opening_roman_marker, lowercase_roman_marker},
12124     {uppercase_opening_roman_marker, uppercase_roman_marker},
12125     lowercase_letter_marker,
12126     uppercase_letter_marker,
12127     lowercase_roman_marker,
12128     uppercase_roman_marker,
12129     digit_marker
12130 }
12131
12132 local delims = {
12133     parsers.period,
12134     parsers.rparent
12135 }
12136
12137 local markers_table = combine_markers_and_delims(markers, delims)
12138
12139 local function enumerator(start_marker, _,

```

```

12140                                     delimiter_type, interrupting)
12141     local delimiter_range
12142     local allowed_end
12143     if interrupting then
12144         delimiter_range = P("1")
12145         allowed_end = C(parsers.spacechar~1) * #parsers.linechar
12146     else
12147         delimiter_range = start_marker
12148         allowed_end = C(parsers.spacechar~1)
12149                     + #(parsers.newline + parsers.eof)
12150     end
12151
12152     return parsers.check_trail
12153             * Ct(C(delimiter_range) * C(delimiter_type))
12154             * allowed_end
12155 end
12156
12157 local starter = join_table_with_func(enumerator, markers_table)
12158
12159 local TightListItem = function(starter)
12160     return parsers.add_indent(starter, "li")
12161             * parsers.indented_content_tight
12162 end
12163
12164 local LooseListItem = function(starter)
12165     return parsers.add_indent(starter, "li")
12166             * parsers.indented_content_loose
12167             * remove_indent("li")
12168 end
12169
12170 local function roman2number(roman)
12171     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
12172                     ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
12173     local numeral = 0
12174
12175     local i = 1
12176     local len = string.len(roman)
12177     while i < len do
12178         local z1, z2 = romans[ string.sub(roman, i, i) ],
12179                         romans[ string.sub(roman, i+1, i+1) ]
12180         if z1 < z2 then
12181             numeral = numeral + (z2 - z1)
12182             i = i + 2
12183         else
12184             numeral = numeral + z1
12185             i = i + 1
12186         end

```

```

12187         end
12188         if i <= len then
12189             numeral = numeral + romans[ string.sub(roman,i,i) ]
12190         end
12191         return numeral
12192     end
12193
12194     local function sniffstyle(numstr, delimend)
12195         local numdelim
12196         if delimend == ")" then
12197             numdelim = "OneParen"
12198         elseif delimend == "." then
12199             numdelim = "Period"
12200         else
12201             numdelim = "Default"
12202         end
12203
12204         local num
12205         num = numstr:match("^([I])$")
12206         if num then
12207             return roman2number(num), "UpperRoman", numdelim
12208         end
12209         num = numstr:match("^([i])$")
12210         if num then
12211             return roman2number(string.upper(num)), "LowerRoman", numdelim
12212         end
12213         num = numstr:match("^([A-Z])$")
12214         if num then
12215             return string.byte(num) - string.byte("A") + 1,
12216                 "UpperAlpha", numdelim
12217         end
12218         num = numstr:match("^([a-z])$")
12219         if num then
12220             return string.byte(num) - string.byte("a") + 1,
12221                 "LowerAlpha", numdelim
12222         end
12223         num = numstr:match("^([IVXLCDM]+)")
12224         if num then
12225             return roman2number(num), "UpperRoman", numdelim
12226         end
12227         num = numstr:match("^([ivxlc dm]+)")
12228         if num then
12229             return roman2number(string.upper(num)), "LowerRoman", numdelim
12230         end
12231         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
12232     end
12233

```

```

12234     local function fancylist(items,tight,start)
12235         local startnum, numstyle, numdelim
12236         = sniffstyle(start[2][1], start[2][2])
12237         return writer.fancylist(items,tight,
12238                                 options.startNumber and startnum or 1,
12239                                 numstyle or "Decimal",
12240                                 numdelim or "Default")
12241     end
12242
12243     local FancyListOfType
12244     = function(start_marker, continuation_marker, delimiter_type)
12245         local enumerator_start
12246         = enumerator(start_marker, continuation_marker,
12247                     delimiter_type)
12248         local enumerator_cont
12249         = enumerator(continuation_marker, continuation_marker,
12250                     delimiter_type)
12251         return Cg(enumerator_start, "listtype")
12252             * (Ct( TightListItem(Cb("listtype"))
12253                 * ((parsers.check_minimal_indent / "")
12254                   * TightListItem(enumerator_cont))^0)
12255             * Cc(true)
12256             * -#((parsers.conditionally_indented_blankline^0 / "")
12257                 * parsers.check_minimal_indent * enumerator_cont)
12258             + Ct( LooseListItem(Cb("listtype"))
12259                 * ((parsers.conditionally_indented_blankline^0 / "")
12260                   * (parsers.check_minimal_indent / "")
12261                   * LooseListItem(enumerator_cont))^0)
12262             * Cc(false)
12263             ) * Ct(Cb("listtype")) / fancylist
12264     end
12265
12266     local FancyList
12267     = join_table_with_func(FancyListOfType, markers_table)
12268
12269     local ListStarter = starter
12270
12271     self.update_rule("OrderedList", FancyList)
12272     self.update_rule("ListStarter", ListStarter)
12273 end
12274 }
12275 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.

```
12276 M.extensions.fenced_code = function(blank_before_code_fence,
12277                                     allow_attributes,
12278                                     allow_raw_blocks)
12279   return {
12280     name = "built-in fenced_code syntax extension",
12281     extend_writer = function(self)
12282       local options = self.options
12283
```

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.

```
12284     function self.fencedCode(s, i, attr)
12285       if not self.is_writing then return "" end
12286       s = s:gsub("\n$", "")
12287       local buf = {}
12288       if attr ~= nil then
12289         table.insert(buf,
12290           {"\\markdownRendererFencedCodeAttributeContextBegin",
12291            self.attributes(attr)})
12292       end
12293       local name = util.cache_verbatim(options.cacheDir, s)
12294       table.insert(buf,
12295         {"\\markdownRendererInputFencedCode{" ,
12296          name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}"}})
12297       if attr ~= nil then
12298         table.insert(buf,
12299           "\\markdownRendererFencedCodeAttributeContextEnd{")
12300       end
12301       return buf
12302     end
12303
```

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

```
12304     if allow_raw_blocks then
12305       function self.rawBlock(s, attr)
12306         if not self.is_writing then return "" end
12307         s = s:gsub("\n$", "")
12308         local name = util.cache_verbatim(options.cacheDir, s)
12309         return {"\\markdownRendererInputRawBlock{" ,
12310          name,"}{" , self.string(attr),"}"}

```

```

12311         end
12312     end
12313 end, extend_reader = function(self)
12314     local parsers = self.parsers
12315     local writer = self.writer
12316
12317     local function captures_geq_length(_,i,a,b)
12318         return #a >= #b and i
12319     end
12320
12321     local function strip_enclosing_whitespaces(str)
12322         return str:gsub("^%s*(.)%s*$", "%1")
12323     end
12324
12325     local tilde_infostring = Cs(Cs((V("HtmlEntity")
12326         + parsers.anyescaped
12327         - parsers.newline)^0)
12328         / strip_enclosing_whitespaces)
12329
12330     local backtick_infostring
12331     = Cs( Cs((V("HtmlEntity")
12332         + ( -#(parsers.backslash * parsers.backtick)
12333         * parsers.anyescaped)
12334         - parsers.newline
12335         - parsers.backtick)^0)
12336         / strip_enclosing_whitespaces)
12337
12338     local fenceindent
12339
12340     local function has_trail(indent_table)
12341         return indent_table ~= nil and
12342             indent_table.trail ~= nil and
12343             next(indent_table.trail) ~= nil
12344     end
12345
12346     local function has_indents(indent_table)
12347         return indent_table ~= nil and
12348             indent_table.indents ~= nil and
12349             next(indent_table.indents) ~= nil
12350     end
12351
12352     local function get_last_indent_name(indent_table)
12353         if has_indents(indent_table) then
12354             return indent_table.indents[#indent_table.indents].name
12355         end
12356     end
12357

```

```

12358     local count_fenced_start_indent =
12359         function(_, _, indent_table, trail)
12360             local last_indent_name = get_last_indent_name(indent_table)
12361             fenceindent = 0
12362             if last_indent_name ~= "li" then
12363                 fenceindent = #trail
12364             end
12365             return true
12366         end
12367
12368     local fencehead = function(char, infostring)
12369         return Cmt( Cb("indent_info")
12370             * parsers.check_trail, count_fenced_start_indent)
12371             * Cg(char^3, "fencelength")
12372             * parsers.optionalspace
12373             * infostring
12374             * (parsers.newline + parsers.eof)
12375     end
12376
12377     local fencetail = function(char)
12378         return parsers.check_trail_no_rem
12379             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
12380             * parsers.optionalspace * (parsers.newline + parsers.eof)
12381             + parsers.eof
12382     end
12383
12384     local process_fenced_line =
12385         function(s, i, -- luacheck: ignore s i
12386             indent_table, line_content, is_blank)
12387             local remainder = ""
12388             if has_trail(indent_table) then
12389                 remainder = indent_table.trail.internal_remainder
12390             end
12391
12392             if is_blank
12393                 and get_last_indent_name(indent_table) == "li" then
12394                 remainder = ""
12395             end
12396
12397             local str = remainder .. line_content
12398             local index = 1
12399             local remaining = fenceindent
12400
12401             while true do
12402                 local c = str:sub(index, index)
12403                 if c == " " and remaining > 0 then
12404                     remaining = remaining - 1

```



```

12405         index = index + 1
12406     elseif c == "\t" and remaining > 3 then
12407         remaining = remaining - 4
12408         index = index + 1
12409     else
12410         break
12411     end
12412 end
12413
12414     return true, str:sub(index)
12415 end
12416
12417 local fencedline = function(char)
12418     return Cmt( Cb("indent_info")
12419         * C(parsers.line - fencetail(char))
12420         * Cc(false), process_fenced_line)
12421 end
12422
12423 local blankfencedline
12424     = Cmt( Cb("indent_info")
12425         * C(parsers.blankline)
12426         * Cc(true), process_fenced_line)
12427
12428 local TildeFencedCode
12429     = fencehead(parsers.tilde, tilde_infostring)
12430     * Cs(( (parsers.check_minimal_blank_indent / "")
12431         * blankfencedline
12432         + ( parsers.check_minimal_indent / ""))
12433         * fencedline(parsers.tilde))^0)
12434     * ( (parsers.check_minimal_indent / "")
12435         * fencetail(parsers.tilde) + parsers.succeed)
12436
12437 local BacktickFencedCode
12438     = fencehead(parsers.backtick, backtick_infostring)
12439     * Cs(( (parsers.check_minimal_blank_indent / "")
12440         * blankfencedline
12441         + (parsers.check_minimal_indent / ""))
12442         * fencedline(parsers.backtick))^0)
12443     * ( (parsers.check_minimal_indent / "")
12444         * fencetail(parsers.backtick) + parsers.succeed)
12445
12446 local infostring_with_attributes
12447     = Ct(C((parsers.linechar
12448         - ( parsers.optionalspace
12449         * parsers.attributes))^0)
12450         * parsers.optionalspace
12451         * Ct(parsers.attributes))

```

```

12452
12453     local FencedCode
12454         = ((TildeFencedCode + BacktickFencedCode)
12455           / function(infostring, code)
12456             local expanded_code = self.expandtabs(code)
12457
12458             if allow_raw_blocks then
12459                 local raw_attr = lpeg.match(parsers.raw_attribute,
12460                                             infostring)
12461
12462                 if raw_attr then
12463                     return writer.rawBlock(expanded_code, raw_attr)
12464                 end
12465             end
12466
12467             local attr = nil
12468             if allow_attributes then
12469                 local match = lpeg.match(infostring_with_attributes,
12470                                         infostring)
12471
12472                 if match then
12473                     infostring, attr = table.unpack(match)
12474                 end
12475             end
12476             return writer.fencedCode(expanded_code, infostring, attr)
12477         end)
12478
12479     self.insert_pattern("Block after Verbatim",
12480                       FencedCode, "FencedCode")
12481
12482     local fencestart
12483     if blank_before_code_fence then
12484         fencestart = parsers.fail
12485     else
12486         fencestart = fencehead(parsers.backtick, backtick_infostring)
12487                     + fencehead(parsers.tilde, tilde_infostring)
12488     end
12489
12490     self.update_rule("EndlineExceptions", function(previous_pattern)
12491         if previous_pattern == nil then
12492             previous_pattern = parsers.EndlineExceptions
12493         end
12494         return previous_pattern + fencestart
12495     end)
12496
12497     self.add_special_character("`")
12498     self.add_special_character("~")
12499 end
12500 }

```

12499 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.

```
12500 M.extensions.fenced_divs = function(blank_before_div_fence)
12501   return {
12502     name = "built-in fenced_divs syntax extension",
12503     extend_writer = function(self)
```

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

```
12504     function self.div_begin(attributes)
12505       local start_output
12506       = {"\\markdownRendererFencedDivAttributeContextBegin\n",
12507         self.attributes(attributes)}
12508       local end_output
12509       = {"\\markdownRendererFencedDivAttributeContextEnd{}}"
12510       return self.push_attributes(
12511         "div", attributes, start_output, end_output)
12512     end
```

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

```
12513     function self.div_end()
12514       return self.pop_attributes("div")
12515     end
12516   end, extend_reader = function(self)
12517     local parsers = self.parsers
12518     local writer = self.writer
```

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

```
12519     local fenced_div_infostring
12520       = C((parsers.linechar
12521         - ( parsers.spacechar^1
12522           * parsers.colon^1))^1)
12523
12524     local fenced_div_begin = parsers.nonindentspace
12525       * parsers.colon^3
12526       * parsers.optionalspace
12527       * fenced_div_infostring
12528       * ( parsers.spacechar^1
12529         * parsers.colon^1)^0
12530       * parsers.optionalspace
12531       * (parsers.newline + parsers.eof)
```

```

12532
12533     local fenced_div_end = parsers.nonindentSPACE
12534         * parsers.colon^3
12535         * parsers.optionalspace
12536         * (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.

```

12537     self.initialize_named_group("fenced_div_level", "0")
12538     self.initialize_named_group("fenced_div_num_opening_indents")
12539
12540     local function increment_div_level()
12541         local push_indent_table =
12542             function(s, i, indent_table, -- luacheck: ignore s i
12543                 fenced_div_num_opening_indents, fenced_div_level)
12544                 fenced_div_level = tonumber(fenced_div_level) + 1
12545                 local num_opening_indents = 0
12546                 if indent_table.indents ~= nil then
12547                     num_opening_indents = #indent_table.indents
12548                 end
12549                 fenced_div_num_opening_indents[fenced_div_level]
12550                     = num_opening_indents
12551                 return true, fenced_div_num_opening_indents
12552             end
12553
12554     local increment_level =
12555         function(s, i, fenced_div_level) -- luacheck: ignore s i
12556             fenced_div_level = tonumber(fenced_div_level) + 1
12557             return true, tostring(fenced_div_level)
12558         end
12559
12560     return Cg( Cmt( Cb("indent_info")
12561         * Cb("fenced_div_num_opening_indents")
12562         * Cb("fenced_div_level"), push_indent_table)
12563         , "fenced_div_num_opening_indents")
12564         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
12565         , "fenced_div_level")
12566     end
12567
12568     local function decrement_div_level()
12569         local pop_indent_table =
12570             function(s, i, -- luacheck: ignore s i
12571                 fenced_div_indent_table, fenced_div_level)

```

```

12572         fenced_div_level = tonumber(fenced_div_level)
12573         fenced_div_indent_table[fenced_div_level] = nil
12574         return true, tostring(fenced_div_level - 1)
12575     end
12576
12577     return Cg( Cmt( Cb("fenced_div_num_opening_indents")
12578                 * Cb("fenced_div_level"), pop_indent_table)
12579             , "fenced_div_level")
12580 end
12581
12582
12583 local non_fenced_div_block
12584     = parsers.check_minimal_indent * V("Block")
12585     - parsers.check_minimal_indent_and_trail * fenced_div_end
12586
12587 local non_fenced_div_paragraph
12588     = parsers.check_minimal_indent * V("Paragraph")
12589     - parsers.check_minimal_indent_and_trail * fenced_div_end
12590
12591 local blank = parsers.minimally_indented_blank
12592
12593 local block_separated = parsers.block_sep_group(blank)
12594     * non_fenced_div_block
12595
12596 local loop_body_pair
12597     = parsers.create_loop_body_pair(block_separated,
12598                                     non_fenced_div_paragraph,
12599                                     parsers.block_sep_group(blank),
12600                                     parsers.par_sep_group(blank))
12601
12602 local content_loop = ( non_fenced_div_block
12603                       * loop_body_pair.block^0
12604                       + non_fenced_div_paragraph
12605                       * block_separated
12606                       * loop_body_pair.block^0
12607                       + non_fenced_div_paragraph
12608                       * loop_body_pair.par^0)
12609     * blank^0
12610
12611 local FencedDiv = fenced_div_begin
12612     / function (infostring)
12613         local attr
12614         = lpeg.match(Ct(parsers.attributes),
12615                     infostring)
12616         if attr == nil then
12617             attr = {"." .. infostring}
12618         end

```

```

12619         return attr
12620     end
12621     / writer.div_begin
12622     * increment_div_level()
12623     * parsers.skipblanklines
12624     * Ct(content_loop)
12625     * parsers.minimally_indented_blank~0
12626     * parsers.check_minimal_indent_and_trail
12627     * fenced_div_end
12628     * decrement_div_level()
12629     * (Cc("") / writer.div_end)
12630
12631     self.insert_pattern("Block after Verbatim",
12632                       FencedDiv, "FencedDiv")
12633
12634     self.add_special_character(":")
12635

```

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.

```

12636     local function is_inside_div()
12637         local check_div_level =
12638             function(s, i, fenced_div_level) -- luacheck: ignore s i
12639                 fenced_div_level = tonumber(fenced_div_level)
12640                 return fenced_div_level > 0
12641             end
12642
12643         return Cmt(Cb("fenced_div_level"), check_div_level)
12644     end
12645
12646     local function check_indent()
12647         local compare_indent =
12648             function(s, i, indent_table, -- luacheck: ignore s i
12649                 fenced_div_num_opening_indents, fenced_div_level)
12650                 fenced_div_level = tonumber(fenced_div_level)
12651                 local num_current_indents
12652                 = ( indent_table.current_line_indents ~= nil and
12653                   #indent_table.current_line_indents) or 0
12654                 local num_opening_indents
12655                 = fenced_div_num_opening_indents[fenced_div_level]
12656                 return num_current_indents == num_opening_indents
12657             end
12658
12659         return Cmt( Cb("indent_info")
12660                   * Cb("fenced_div_num_opening_indents")
12661                   * Cb("fenced_div_level"), compare_indent)

```

```

12662     end
12663
12664     local fencestart = is_inside_div()
12665                     * fenced_div_end
12666                     * check_indent()
12667
12668     if not blank_before_div_fence then
12669         self.update_rule("EndlineExceptions", function(previous_pattern)
12670             if previous_pattern == nil then
12671                 previous_pattern = parsers.EndlineExceptions
12672             end
12673             return previous_pattern + fencestart
12674         end)
12675     end
12676 end
12677 }
12678 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

12679 M.extensions.header_attributes = function()
12680     return {
12681         name = "built-in header_attributes syntax extension",
12682         extend_writer = function()
12683             end, extend_reader = function(self)
12684                 local parsers = self.parsers
12685                 local writer = self.writer
12686
12687                 local function strip_atx_end(s)
12688                     return s:gsub("%s+##%s*$", "")
12689                 end
12690
12691                 local AtxHeading = Cg(parsers.heading_start, "level")
12692                                     * parsers.optionalspace
12693                                     * (C(((parsers.linechar
12694                                         - (parsers.attributes
12695                                             * parsers.optionalspace
12696                                             * parsers.newline))
12697                                         * (parsers.linechar
12698                                             - parsers.lbrace)^0)^1)
12699                                       / strip_atx_end
12700                                       / parsers.parse_heading_text)
12701                                     * Cg(Ct(parsers.newline
12702                                             + (parsers.attributes
12703                                                 * parsers.optionalspace

```

```

12704             * parsers.newline)), "attributes")
12705         * Cb("level")
12706         * Cb("attributes")
12707         / writer.heading
12708
12709     local function strip_trailing_spaces(s)
12710         return s:gsub("%s*$", "")
12711     end
12712
12713     local heading_line = (parsers.linechar
12714         - (parsers.attributes
12715             * parsers.optionalspace
12716             * parsers.newline))^1
12717         - parsers.thematic_break_lines
12718
12719     local heading_text
12720         = heading_line
12721         * ( (V("Endline") / "\n")
12722             * (heading_line - parsers.heading_level))^0
12723         * parsers.newline^-1
12724
12725     local SettextHeading
12726         = parsers.freeze_trail * parsers.check_trail_no_rem
12727         * #(heading_text
12728             * (parsers.attributes
12729                 * parsers.optionalspace
12730                 * parsers.newline)^-1
12731             * parsers.check_minimal_indent
12732             * parsers.check_trail
12733             * parsers.heading_level)
12734         * Cs(heading_text) / strip_trailing_spaces
12735         / parsers.parse_heading_text
12736         * Cg(Ct((parsers.attributes
12737             * parsers.optionalspace
12738             * parsers.newline)^-1), "attributes")
12739         * parsers.check_minimal_indent_and_trail * parsers.heading_level
12740         * Cb("attributes")
12741         * parsers.newline
12742         * parsers.unfreeze_trail
12743         / writer.heading
12744
12745     local Heading = AtxHeading + SettextHeading
12746     self.update_rule("Heading", Heading)
12747 end
12748 }
12749 end

```



### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```
12750 M.extensions.inline_code_attributes = function()
12751   return {
12752     name = "built-in inline_code_attributes syntax extension",
12753     extend_writer = function()
12754     end, extend_reader = function(self)
12755       local writer = self.writer
12756
12757       local CodeWithAttributes = parsers.inticks
12758         * Ct(parsers.attributes)
12759         / writer.code
12760
12761       self.insert_pattern("Inline before Code",
12762         CodeWithAttributes,
12763         "CodeWithAttributes")
12764     end
12765   }
12766 end
```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
12767 M.extensions.line_blocks = function()
12768   return {
12769     name = "built-in line_blocks syntax extension",
12770     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.

```
12771     function self.lineblock(lines)
12772       if not self.is_writing then return "" end
12773       local buffer = {}
12774       for i = 1, #lines - 1 do
12775         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
12776       end
12777       buffer[#buffer + 1] = lines[#lines]
12778
12779       return {"\\markdownRendererLineBlockBegin\n"
12780         ,buffer,
12781         "\n\\markdownRendererLineBlockEnd "}
12782     end
12783   end, extend_reader = function(self)
12784     local parsers = self.parsers
12785     local writer = self.writer
```

```

12786
12787     local LineBlock
12788         = Ct((Cs(( (parsers.pipe * parsers.space) / ""
12789                 * ((parsers.space)/entities.char_entity("nbsp"))^0
12790                 * parsers.linechar^0 * (parsers.newline/""))
12791                 * (-parsers.pipe
12792                   * (parsers.space^1/" ")
12793                   * parsers.linechar^1
12794                   * (parsers.newline/""))
12795                 )^0
12796                 * (parsers.blankline/"")^0)
12797         / self.parser_functions.parse_inlines)^1)
12798     / writer.lineblock
12799
12800     self.insert_pattern("Block after Blockquote",
12801                       LineBlock, "LineBlock")
12802 end
12803 }
12804 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

12805 M.extensions.mark = function()
12806     return {
12807         name = "built-in mark syntax extension",
12808         extend_writer = function(self)

```

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

```

12809         function self.mark(s)
12810             if self.flatten_inlines then return s end
12811             return {"\\markdownRendererMark{" , s, "}" }
12812         end
12813     end, extend_reader = function(self)
12814         local parsers = self.parsers
12815         local writer = self.writer
12816
12817         local doubleequals = P("==")
12818
12819         local Mark
12820             = parsers.between(V("Inline"), doubleequals, doubleequals)
12821         / function (inlines) return writer.mark(inlines) end
12822
12823         self.add_special_character("=")
12824         self.insert_pattern("Inline before LinkAndEmph",
12825                             Mark, "Mark")
12826     end

```

```
12827 }
12828 end
```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
12829 M.extensions.link_attributes = function()
12830   return {
12831     name = "built-in link_attributes syntax extension",
12832     extend_writer = function()
12833     end, extend_reader = function(self)
12834       local parsers = self.parsers
12835       local options = self.options
12836
```

The following patterns define link reference definitions with attributes.

```
12837     local define_reference_parser
12838       = (parsers.check_trail / "")
12839       * parsers.link_label
12840       * parsers.colon
12841       * parsers.spnlc * parsers.url
12842       * ( parsers.spnlc_sep * parsers.title
12843         * (parsers.spnlc * Ct(parsers.attributes))
12844         * parsers.only_blank
12845         + parsers.spnlc_sep * parsers.title * parsers.only_blank
12846         + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
12847         * parsers.only_blank
12848         + Cc("") * parsers.only_blank)
12849
12850     local ReferenceWithAttributes = define_reference_parser
12851                                   / self.register_link
12852
12853     self.update_rule("Reference", ReferenceWithAttributes)
12854
```

The following patterns define direct and indirect links with attributes.

```
12855
12856     local LinkWithAttributesAndEmph
12857       = Ct(parsers.link_and_emph_table * Cg(Cc(true),
12858         "match_link_attributes"))
12859       / self.defer_link_and_emphasis_processing
12860
12861     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
12862
```

The following patterns define autolinks with attributes.

```
12863     local AutoLinkUrlWithAttributes
```

```

12864             = parsers.auto_link_url
12865             * Ct(parsers.attributes)
12866             / self.auto_link_url
12867
12868     self.insert_pattern("Inline before AutoLinkUrl",
12869                       AutoLinkUrlWithAttributes,
12870                       "AutoLinkUrlWithAttributes")
12871
12872     local AutoLinkEmailWithAttributes
12873           = parsers.auto_link_email
12874           * Ct(parsers.attributes)
12875           / self.auto_link_email
12876
12877     self.insert_pattern("Inline before AutoLinkEmail",
12878                       AutoLinkEmailWithAttributes,
12879                       "AutoLinkEmailWithAttributes")
12880
12881     if options.relativeReferences then
12882
12883         local AutoLinkRelativeReferenceWithAttributes
12884               = parsers.auto_link_relative_reference
12885               * Ct(parsers.attributes)
12886               / self.auto_link_url
12887
12888         self.insert_pattern(
12889           "Inline before AutoLinkRelativeReference",
12890           AutoLinkRelativeReferenceWithAttributes,
12891           "AutoLinkRelativeReferenceWithAttributes")
12892
12893     end
12894
12895 end
12896 }
12897 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.

```

12898 M.extensions.notes = function(notes, inline_notes)
12899   assert(notes or inline_notes)
12900   return {
12901     name = "built-in notes syntax extension",
12902     extend_writer = function(self)

```

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

```
12903     function self.note(s)
12904         if self.flatten_inlines then return "" end
12905         return {"\\markdownRendererNote{",s,""}
12906     end
12907 end, extend_reader = function(self)
12908     local parsers = self.parsers
12909     local writer = self.writer
12910
12911     local rawnotes = parsers.rawnotes
12912
12913     if inline_notes then
12914         local InlineNote
12915             = parsers.circumflex
12916             * ( parsers.link_label
12917               / self.parser_functions.parse_inlines_no_inline_note)
12918             / writer.note
12919
12920         self.insert_pattern("Inline after LinkAndEmph",
12921                             InlineNote, "InlineNote")
12922     end
12923     if notes then
12924         local function strip_first_char(s)
12925             return s:sub(2)
12926         end
12927
12928         local RawNoteRef
12929             = #(parsers.lbracket * parsers.circumflex)
12930             * parsers.link_label / strip_first_char
12931
12932         -- like indirect_link
12933         local function lookup_note(ref)
12934             return writer.defer_call(function()
12935                 local found = rawnotes[self.normalize_tag(ref)]
12936                 if found then
12937                     return writer.note(
12938                         self.parser_functions.parse_blocks_nested(found))
12939                 else
12940                     return {"[",
12941                             self.parser_functions.parse_inlines("^" .. ref), "]" }
12942                 end
12943             end)
12944         end
12945
12946         local function register_note(ref,rawnote)
12947             local normalized_tag = self.normalize_tag(ref)
```

```

12948         if rawnotes[normalized_tag] == nil then
12949             rawnotes[normalized_tag] = rawnote
12950         end
12951         return ""
12952     end
12953
12954     local NoteRef = RawNoteRef / lookup_note
12955
12956     local optionally_indented_line
12957         = parsers.check_optional_indent_and_any_trail * parsers.line
12958
12959     local blank
12960         = parsers.check_optional_blank_indent_and_any_trail
12961         * parsers.optionalspace * parsers.newline
12962
12963     local chunk
12964         = Cs(parsers.line
12965             * (optionally_indented_line - blank)^0)
12966
12967     local indented_blocks = function(bl)
12968         return Cs( bl
12969             * ( blank^1 * (parsers.check_optional_indent / "")
12970             * parsers.check_code_trail
12971             * -parsers.blankline * bl)^0)
12972     end
12973
12974     local NoteBlock
12975         = parsers.check_trail_no_rem
12976         * RawNoteRef * parsers.colon
12977         * parsers.spnlc * indented_blocks(chunk)
12978         / register_note
12979
12980     self.update_rule("Reference", function(previous_pattern)
12981         if previous_pattern == nil then
12982             previous_pattern = parsers.Reference
12983         end
12984         return NoteBlock + previous_pattern
12985     end)
12986
12987     self.insert_pattern("Inline before LinkAndEmph",
12988         NoteRef, "NoteRef")
12989     end
12990
12991     self.add_special_character("^")
12992 end
12993 }
12994 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.

```
12995 M.extensions.pipe_tables = function(table_captions, table_attributes)
12996
12997   local function make_pipe_table_rectangular(rows)
12998     local num_columns = #rows[2]
12999     local rectangular_rows = {}
13000     for i = 1, #rows do
13001       local row = rows[i]
13002       local rectangular_row = {}
13003       for j = 1, num_columns do
13004         rectangular_row[j] = row[j] or ""
13005       end
13006       table.insert(rectangular_rows, rectangular_row)
13007     end
13008     return rectangular_rows
13009   end
13010
13011   local function pipe_table_row(allow_empty_first_column
13012                                , nonempty_column
13013                                , column_separator
13014                                , column)
13015     local row_beginning
13016     if allow_empty_first_column then
13017       row_beginning = -- empty first column
13018                     #(parsers.spacechar^4
13019                      * column_separator)
13020                     * parsers.optionalspace
13021                     * column
13022                     * parsers.optionalspace
13023                     -- non-empty first column
13024                     + parsers.nonindentSPACE
13025                     * nonempty_column^-1
13026                     * parsers.optionalspace
13027     else
13028       row_beginning = parsers.nonindentSPACE
13029                     * nonempty_column^-1
13030                     * parsers.optionalspace
13031     end
13032
13033   return Ct(row_beginning
```

```

13034         * (-- single column with no leading pipes
13035           #(column_separator
13036             * parsers.optionalspace
13037             * parsers.newline)
13038         * column_separator
13039         * parsers.optionalspace
13040         -- single column with leading pipes or
13041         -- more than a single column
13042         + (column_separator
13043           * parsers.optionalspace
13044           * column
13045           * parsers.optionalspace)^1
13046         * (column_separator
13047           * parsers.optionalspace)^-1))
13048     end
13049
13050     return {
13051       name = "built-in pipe_tables syntax extension",
13052       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.

```

13053     function self.table(rows, caption, attributes)
13054       if not self.is_writing then return "" end
13055       local buffer = {}
13056       if attributes ~= nil then
13057         table.insert(buffer,
13058           "\\markdownRendererTableAttributeContextBegin\n")
13059         table.insert(buffer, self.attributes(attributes))
13060       end
13061       table.insert(buffer,
13062         {"\\markdownRendererTable{",
13063           caption or "", "}{" , #rows - 1, "}{" ,
13064           #rows[1], "}"}))
13065       local temp = rows[2] -- put alignments on the first row
13066       rows[2] = rows[1]
13067       rows[1] = temp
13068       for i, row in ipairs(rows) do
13069         table.insert(buffer, "{")
13070         for _, column in ipairs(row) do
13071           if i > 1 then -- do not use braces for alignments
13072             table.insert(buffer, "{")
13073           end
13074           table.insert(buffer, column)
13075           if i > 1 then
13076             table.insert(buffer, "}")
13077           end

```



```

13078         end
13079         table.insert(buffer, "}")
13080     end
13081     if attributes ~= nil then
13082         table.insert(buffer,
13083             "\\markdownRendererTableAttributeContextEnd{")
13084     end
13085     return buffer
13086 end
13087 end, extend_reader = function(self)
13088     local parsers = self.parsers
13089     local writer = self.writer
13090
13091     local table_hline_separator = parsers.pipe + parsers.plus
13092
13093     local table_hline_column = (parsers.dash
13094         - #(parsers.dash
13095             * (parsers.spacechar
13096                 + table_hline_separator
13097                 + parsers.newline)))^1
13098     * (parsers.colon * Cc("r")
13099         + parsers.dash * Cc("d"))
13100     + parsers.colon
13101     * (parsers.dash
13102         - #(parsers.dash
13103             * (parsers.spacechar
13104                 + table_hline_separator
13105                 + parsers.newline)))^1
13106     * (parsers.colon * Cc("c")
13107         + parsers.dash * Cc("l"))
13108
13109     local table_hline = pipe_table_row(false
13110         , table_hline_column
13111         , table_hline_separator
13112         , table_hline_column)
13113
13114     local table_caption_beginning
13115     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
13116         * parsers.optionalspace * parsers.newline)^0
13117     * parsers.check_minimal_indent_and_trail
13118     * (P("Table")^-1 * parsers.colon)
13119     * parsers.optionalspace
13120
13121     local function strip_trailing_spaces(s)
13122         return s:gsub("%s*$", "")
13123     end
13124

```

```

13125     local table_row
13126         = pipe_table_row(true
13127             , (C((parsers.linechar - parsers.pipe)^1)
13128               / strip_trailing_spaces
13129               / self.parser_functions.parse_inlines)
13130             , parsers.pipe
13131             , (C((parsers.linechar - parsers.pipe)^0)
13132               / strip_trailing_spaces
13133               / self.parser_functions.parse_inlines))
13134
13135     local table_caption
13136     if table_captions then
13137         table_caption = #table_caption_beginning
13138             * table_caption_beginning
13139         if table_attributes then
13140             table_caption = table_caption
13141                 * (C(((( parsers.linechar
13142                       - (parsers.attributes
13143                         * parsers.optionalspace
13144                         * parsers.newline
13145                         * -( parsers.optionalspace
13146                           * parsers.linechar)))
13147                   + ( parsers.newline
13148                       * #( parsers.optionalspace
13149                         * parsers.linechar)
13150                       * C(parsers.optionalspace)
13151                         / writer.space))
13152                   * (parsers.linechar
13153                       - parsers.lbrace)^0)^1)
13154                   / self.parser_functions.parse_inlines)
13155                 * (parsers.newline
13156                   + ( Ct(parsers.attributes)
13157                     * parsers.optionalspace
13158                     * parsers.newline))
13159         else
13160             table_caption = table_caption
13161                 * C(( parsers.linechar
13162                   + ( parsers.newline
13163                     * #( parsers.optionalspace
13164                       * parsers.linechar)
13165                     * C(parsers.optionalspace)
13166                       / writer.space))^1)
13167                   / self.parser_functions.parse_inlines
13168                 * parsers.newline
13169         end
13170     else
13171         table_caption = parsers.fail

```

```

13172     end
13173
13174     local PipeTable
13175         = Ct( table_row * parsers.newline
13176             * (parsers.check_minimal_indent_and_trail / {})
13177             * table_hline * parsers.newline
13178             * ( (parsers.check_minimal_indent / {})
13179               * table_row * parsers.newline)^0)
13180         / make_pipe_table_rectangular
13181         * table_caption^-1
13182         / writer.table
13183
13184     self.insert_pattern("Block after Blockquote",
13185                       PipeTable, "PipeTable")
13186 end
13187 }
13188 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

13189 M.extensions.raw_inline = function()
13190     return {
13191         name = "built-in raw_inline syntax extension",
13192         extend_writer = function(self)
13193             local options = self.options
13194

```

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

```

13195         function self.rawInline(s, attr)
13196             if not self.is_writing then return "" end
13197             if self.flatten_inlines then return s end
13198             local name = util.cache_verbatim(options.cacheDir, s)
13199             return {"\\markdownRendererInputRawInline{" ,
13200                   name,"}{" , self.string(attr),"}"}
13201         end
13202     end, extend_reader = function(self)
13203         local writer = self.writer
13204
13205         local RawInline = parsers.inticks
13206                         * parsers.raw_attribute
13207                         / writer.rawInline
13208
13209         self.insert_pattern("Inline before Code",
13210                           RawInline, "RawInline")
13211     end

```

```
13212 }
13213 end
```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
13214 M.extensions.strike_through = function()
13215   return {
13216     name = "built-in strike_through syntax extension",
13217     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.

```
13218       function self.strike_through(s)
13219         if self.flatten_inlines then return s end
13220         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
13221       end
13222     end, extend_reader = function(self)
13223       local parsers = self.parsers
13224       local writer = self.writer
13225
13226       local StrikeThrough = (
13227         parsers.between(parsers.Inline, parsers.doubletildes,
13228           parsers.doubletildes)
13229       ) / writer.strike_through
13230
13231       self.insert_pattern("Inline after LinkAndEmph",
13232         StrikeThrough, "StrikeThrough")
13233
13234       self.add_special_character("~")
13235     end
13236   }
13237 end
```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
13238 M.extensions.subscripts = function()
13239   return {
13240     name = "built-in subscripts syntax extension",
13241     extend_writer = function(self)
```

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

```
13242       function self.subscript(s)
13243         if self.flatten_inlines then return s end
```

```

13244     return {"\\markdownRendererSubscript{" ,s,"}"}
13245   end
13246 end, extend_reader = function(self)
13247   local parsers = self.parsers
13248   local writer = self.writer
13249
13250   local Subscript = (
13251     parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
13252   ) / writer.subscript
13253
13254   self.insert_pattern("Inline after LinkAndEmph",
13255     Subscript, "Subscript")
13256
13257   self.add_special_character("~")
13258 end
13259 }
13260 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

13261 M.extensions.superscripts = function()
13262   return {
13263     name = "built-in superscripts syntax extension",
13264     extend_writer = function(self)

```

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

```

13265     function self.superscript(s)
13266       if self.flatten_inlines then return s end
13267       return {"\\markdownRendererSuperscript{" ,s,"}"}
13268     end
13269   end, extend_reader = function(self)
13270     local parsers = self.parsers
13271     local writer = self.writer
13272
13273     local Superscript = (
13274       parsers.between(parsers.Str, parsers.circumflex,
13275         parsers.circumflex)
13276     ) / writer.superscript
13277
13278     self.insert_pattern("Inline after LinkAndEmph",
13279       Superscript, "Superscript")
13280
13281     self.add_special_character("^")
13282   end
13283 }

```

13284 end

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
13285 M.extensions.tex_math = function(tex_math_dollars,  
13286                                     tex_math_single_backslash,  
13287                                     tex_math_double_backslash)  
13288   return {  
13289     name = "built-in tex_math syntax extension",  
13290     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.

```
13291     function self.display_math(s)  
13292       if self.flatten_inlines then return s end  
13293       return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}  
13294     end
```

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

```
13295     function self.inline_math(s)  
13296       if self.flatten_inlines then return s end  
13297       return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}  
13298     end  
13299 end, extend_reader = function(self)  
13300   local parsers = self.parsers  
13301   local writer = self.writer  
13302  
13303   local function between(p, starter, ender)  
13304     return (starter * Cs(p * (p - ender)^0) * ender)  
13305   end  
13306  
13307   local function strip_preceding_whitespaces(str)  
13308     return str:gsub("^%s*(.-)$", "%1")  
13309   end  
13310  
13311   local allowed_before_closing  
13312     = B( parsers.backslash * parsers.any  
13313         + parsers.any * (parsers.any - parsers.backslash))  
13314  
13315   local allowed_before_closing_no_space  
13316     = B( parsers.backslash * parsers.any  
13317         + parsers.any * (parsers.nonspacechar - parsers.backslash))  
13318
```

The following patterns implement the Pandoc dollar math syntax extension.

```

13319     local dollar_math_content
13320         = (parsers.newline * (parsers.check_optional_indent / ""))
13321           + parsers.backslash^-1
13322           * parsers.linechar)
13323         - parsers.blankline^2
13324         - parsers.dollar
13325
13326     local inline_math_opening_dollars = parsers.dollar
13327                                         * #(parsers.nonspacechar)
13328
13329     local inline_math_closing_dollars
13330         = allowed_before_closing_no_space
13331         * parsers.dollar
13332         * -#(parsers.digit)
13333
13334     local inline_math_dollars = between(Cs( dollar_math_content),
13335                                         inline_math_opening_dollars,
13336                                         inline_math_closing_dollars)
13337
13338     local display_math_opening_dollars = parsers.dollar
13339                                         * parsers.dollar
13340
13341     local display_math_closing_dollars = parsers.dollar
13342                                         * parsers.dollar
13343
13344     local display_math_dollars = between(Cs( dollar_math_content),
13345                                         display_math_opening_dollars,
13346                                         display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

13347     local backslash_math_content
13348         = (parsers.newline * (parsers.check_optional_indent / ""))
13349           + parsers.linechar)
13350         - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

13351     local inline_math_opening_double = parsers.backslash
13352                                         * parsers.backslash
13353                                         * parsers.lparent
13354
13355     local inline_math_closing_double = allowed_before_closing
13356                                         * parsers.spacechar^0
13357                                         * parsers.backslash
13358                                         * parsers.backslash
13359                                         * parsers.rparent
13360

```

```

13361     local inline_math_double = between(Cs( backslash_math_content),
13362                                     inline_math_opening_double,
13363                                     inline_math_closing_double)
13364                                     / strip_preceding_whitespaces
13365
13366     local display_math_opening_double = parsers.backslash
13367                                     * parsers.backslash
13368                                     * parsers.lbracket
13369
13370     local display_math_closing_double = allowed_before_closing
13371                                     * parsers.spacechar^0
13372                                     * parsers.backslash
13373                                     * parsers.backslash
13374                                     * parsers.rbracket
13375
13376     local display_math_double = between(Cs( backslash_math_content),
13377                                     display_math_opening_double,
13378                                     display_math_closing_double)
13379                                     / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

13380     local inline_math_opening_single = parsers.backslash
13381                                     * parsers.lparent
13382
13383     local inline_math_closing_single = allowed_before_closing
13384                                     * parsers.spacechar^0
13385                                     * parsers.backslash
13386                                     * parsers.rparent
13387
13388     local inline_math_single = between(Cs( backslash_math_content),
13389                                     inline_math_opening_single,
13390                                     inline_math_closing_single)
13391                                     / strip_preceding_whitespaces
13392
13393     local display_math_opening_single = parsers.backslash
13394                                     * parsers.lbracket
13395
13396     local display_math_closing_single = allowed_before_closing
13397                                     * parsers.spacechar^0
13398                                     * parsers.backslash
13399                                     * parsers.rbracket
13400
13401     local display_math_single = between(Cs( backslash_math_content),
13402                                     display_math_opening_single,
13403                                     display_math_closing_single)
13404                                     / strip_preceding_whitespaces
13405
13406     local display_math = parsers.fail

```



```

13407
13408     local inline_math = parsers.fail
13409
13410     if tex_math_dollars then
13411         display_math = display_math + display_math_dollars
13412         inline_math = inline_math + inline_math_dollars
13413     end
13414
13415     if tex_math_double_backslash then
13416         display_math = display_math + display_math_double
13417         inline_math = inline_math + inline_math_double
13418     end
13419
13420     if tex_math_single_backslash then
13421         display_math = display_math + display_math_single
13422         inline_math = inline_math + inline_math_single
13423     end
13424
13425     local TexMath = display_math / writer.display_math
13426                   + inline_math / writer.inline_math
13427
13428     self.insert_pattern("Inline after LinkAndEmph",
13429                       TexMath, "TexMath")
13430
13431     if tex_math_dollars then
13432         self.add_special_character("$")
13433     end
13434
13435     if tex_math_single_backslash or tex_math_double_backslash then
13436         self.add_special_character("\\")
13437         self.add_special_character("[")
13438         self.add_special_character("]")
13439         self.add_special_character("(")
13440         self.add_special_character("(")
13441     end
13442 end
13443 }
13444 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.

```
13445 M.extensions.jekyll_data = function(expect_jekyll_data,
13446                                     ensure_jekyll_data)
13447   return {
13448     name = "built-in jekyll_data syntax extension",
13449     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.

```
13450     function self.jekyllData(d, t, p)
13451       if not self.is_writing then return "" end
13452
13453       local buf = {}
13454
13455       local keys = {}
13456       for k, _ in pairs(d) do
13457         table.insert(keys, k)
13458       end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
13459       table.sort(keys, function(first, second)
13460         if type(first) ~= type(second) then
13461           return type(first) < type(second)
13462         else
13463           return first < second
13464         end
13465       end)
13466
13467       if not p then
13468         table.insert(buf, "\\markdownRendererJekyllDataBegin")
13469       end
13470
13471       local is_sequence = false
13472       if #d > 0 and #d == #keys then
13473         for i=1, #d do
13474           if d[i] == nil then
13475             goto not_a_sequence
13476           end
13477         end
13478         is_sequence = true
13479       end
```

```

13480     ::not_a_sequence::
13481
13482     if is_sequence then
13483         table.insert(buf,
13484             "\\markdownRendererJekyllDataSequenceBegin{")
13485         table.insert(buf, self.identifier(p or "null"))
13486         table.insert(buf, "}{")
13487         table.insert(buf, #keys)
13488         table.insert(buf, "}")
13489     else
13490         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
13491         table.insert(buf, self.identifier(p or "null"))
13492         table.insert(buf, "}{")
13493         table.insert(buf, #keys)
13494         table.insert(buf, "}")
13495     end
13496
13497     for _, k in ipairs(keys) do
13498         local v = d[k]
13499         local typ = type(v)
13500         k = tostring(k or "null")
13501         if typ == "table" and next(v) ~= nil then
13502             table.insert(
13503                 buf,
13504                 self.jekyllData(v, t, k)
13505             )
13506         else
13507             k = self.identifier(k)
13508             v = tostring(v)
13509             if typ == "boolean" then
13510                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
13511                 table.insert(buf, k)
13512                 table.insert(buf, "}{")
13513                 table.insert(buf, v)
13514                 table.insert(buf, "}")
13515             elseif typ == "number" then
13516                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
13517                 table.insert(buf, k)
13518                 table.insert(buf, "}{")
13519                 table.insert(buf, v)
13520                 table.insert(buf, "}")
13521             elseif typ == "string" then
13522                 table.insert(buf,
13523                     "\\markdownRendererJekyllDataProgrammaticString{")
13524                 table.insert(buf, k)
13525                 table.insert(buf, "}{")
13526                 table.insert(buf, self.identifier(v))

```

```

13527         table.insert(buf, "}")
13528         table.insert(buf,
13529             "\\markdownRendererJekyllDataTypographicString{")
13530         table.insert(buf, k)
13531         table.insert(buf, "}{")
13532         table.insert(buf, t(v))
13533         table.insert(buf, "}")
13534         elseif typ == "table" then
13535             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
13536             table.insert(buf, k)
13537             table.insert(buf, "}")
13538         else
13539             local error = self.error(format(
13540                 "Unexpected type %s for value of "
13541                 .. "YAML key %s.", typ, k))
13542             table.insert(buf, error)
13543         end
13544     end
13545 end
13546
13547 if is_sequence then
13548     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
13549 else
13550     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
13551 end
13552
13553 if not p then
13554     table.insert(buf, "\\markdownRendererJekyllDataEnd")
13555 end
13556
13557 return buf
13558 end
13559 end, extend_reader = function(self)
13560     local parsers = self.parsers
13561     local writer = self.writer
13562
13563     local JekyllData
13564     = Cmt( C((parsers.line - P("----") - P("..."))^0)
13565         , function(s, i, text) -- luacheck: ignore s i
13566             local data
13567             local ran_ok, _ = pcall(function()
13568                 local tinyyaml = require("tinyyaml")
13569                 data = tinyyaml.parse(text, {timestamps=false})
13570             end)
13571             if ran_ok and data ~= nil then
13572                 return true, writer.jekyllData(data, function(s)
13573                     return self.parser_functions.parse_blocks_nested(s)

```

```

13574         end, nil)
13575     else
13576         return false
13577     end
13578     end
13579 )
13580
13581 local UnexpectedJekyllData
13582 = P("----")
13583 * parsers.blankline / 0
13584 -- if followed by blank, it's thematic break
13585 * #(-parsers.blankline)
13586 * JekyllData
13587 * (P("----") + P("..."))
13588
13589 local ExpectedJekyllData
13590 = ( P("----")
13591     * parsers.blankline / 0
13592     -- if followed by blank, it's thematic break
13593     * #(-parsers.blankline)
13594     )^-1
13595 * JekyllData
13596 * (P("----") + P("..."))^-1
13597
13598 if ensure_jekyll_data then
13599     ExpectedJekyllData = ExpectedJekyllData
13600     * parsers.eof
13601 else
13602     ExpectedJekyllData = ( ExpectedJekyllData
13603     * (V("Blank")^0 / writer.interblocksep)
13604     )^-1
13605 end
13606
13607 self.insert_pattern("Block before Blockquote",
13608     UnexpectedJekyllData, "UnexpectedJekyllData")
13609 if expect_jekyll_data then
13610     self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
13611 end
13612 end
13613 }
13614 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`.

```
13615 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```
13616 options = options or {}
13617 setmetatable(options, { __index = function (_, key)
13618     return defaultOptions[key] end })
```

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```
13619 local parser_convert = nil
13620 return function(input, include_flat_output)
13621     local function convert(input)
13622         if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
13623         local parser = require("markdown-parser")
13624         if metadata.version ~= parser.metadata.version then
13625             warn("markdown.lua " .. metadata.version .. " used with " ..
13626                 "markdown-parser.lua " .. parser.metadata.version .. ".")
13627         end
13628         parser_convert = parser.new(options)
13629     end
13630     return parser_convert(input)
13631 end
```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```
13632 local raw_output, flat_output
13633 if options.eagerCache or options.finalizeCache then
13634     local salt = util.salt(options)
13635     local name, result = util.cache(options.cacheDir, input, salt,
13636                                     convert, ".md.tex")
13637     raw_output = [[\input{}}] .. name .. [[}\relax]]
13638     flat_output = function()
13639         if result == nil then
13640             local input_file = assert(io.open(name, "r"),
13641                                     [[Could not open file "]] .. name .. [[ for reading]])
13642             result = assert(input_file:read("*a"))
13643             assert(input_file:close())
13644         end
```

```

13645         return result
13646     end

```

Otherwise, return the result of the conversion directly.

```

13647     else
13648         raw_output = convert(input)
13649         flat_output = function()
13650             return raw_output
13651         end
13652     end

```

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

```

13653     if options.finalizeCache then
13654         local file, mode
13655         if options.frozenCacheCounter > 0 then
13656             mode = "a"
13657         else
13658             mode = "w"
13659         end
13660         file = assert(io.open(options.frozenCacheFileName, mode),
13661             [[Could not open file ]] .. options.frozenCacheFileName
13662             .. [[ for writing]])
13663         assert(file:write(
13664             [[\expandafter\global\expandafter\def\csname ]]
13665             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
13666             .. [[\endcsname{}]] .. raw_output .. [[]]] .. "\n"))
13667         assert(file:close())
13668     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.

```

13669     if include_flat_output then
13670         return raw_output, flat_output
13671     else
13672         return raw_output
13673     end
13674 end
13675 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```

13676 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

13677     options = options or {}

```

```

13678 setmetatable(options, { __index = function (_, key)
13679     return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

13680 if options.singletonCache and singletonCache.convert then
13681     for k, v in pairs(defaultOptions) do
13682         if type(v) == "table" then
13683             for i = 1, math.max(#singletonCache.options[k], #options[k]) do
13684                 if singletonCache.options[k][i] ~= options[k][i] then
13685                     goto miss
13686                 end
13687             end

```

The `cacheDir` option is disregarded.

```

13688         elseif k ~= "cacheDir"
13689             and singletonCache.options[k] ~= options[k] then
13690             goto miss
13691         end
13692     end
13693     return singletonCache.convert
13694 end
13695 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

13696 local extensions = {}
13697
13698 if options.bracketedSpans then
13699     local bracketed_spans_extension = M.extensions.bracketed_spans()
13700     table.insert(extensions, bracketed_spans_extension)
13701 end
13702
13703 if options.contentBlocks then
13704     local content_blocks_extension = M.extensions.content_blocks(
13705         options.contentBlocksLanguageMap)
13706     table.insert(extensions, content_blocks_extension)
13707 end
13708
13709 if options.definitionLists then
13710     local definition_lists_extension = M.extensions.definition_lists(
13711         options.tightLists)
13712     table.insert(extensions, definition_lists_extension)
13713 end
13714
13715 if options.fencedCode then
13716     local fenced_code_extension = M.extensions.fenced_code(
13717         options.blankBeforeCodeFence,
13718         options.fencedCodeAttributes,
13719         options.rawAttribute)

```



```

13720     table.insert(extensions, fenced_code_extension)
13721 end
13722
13723 if options.fencedDivs then
13724     local fenced_div_extension = M.extensions.fenced_divs(
13725         options.blankBeforeDivFence)
13726     table.insert(extensions, fenced_div_extension)
13727 end
13728
13729 if options.headerAttributes then
13730     local header_attributes_extension = M.extensions.header_attributes()
13731     table.insert(extensions, header_attributes_extension)
13732 end
13733
13734 if options.inlineCodeAttributes then
13735     local inline_code_attributes_extension =
13736         M.extensions.inline_code_attributes()
13737     table.insert(extensions, inline_code_attributes_extension)
13738 end
13739
13740 if options.jekyllData then
13741     local jekyll_data_extension = M.extensions.jekyll_data(
13742         options.expectJekyllData, options.ensureJekyllData)
13743     table.insert(extensions, jekyll_data_extension)
13744 end
13745
13746 if options.linkAttributes then
13747     local link_attributes_extension =
13748         M.extensions.link_attributes()
13749     table.insert(extensions, link_attributes_extension)
13750 end
13751
13752 if options.lineBlocks then
13753     local line_block_extension = M.extensions.line_blocks()
13754     table.insert(extensions, line_block_extension)
13755 end
13756
13757 if options.mark then
13758     local mark_extension = M.extensions.mark()
13759     table.insert(extensions, mark_extension)
13760 end
13761
13762 if options.pipeTables then
13763     local pipe_tables_extension = M.extensions.pipe_tables(
13764         options.tableCaptions, options.tableAttributes)
13765     table.insert(extensions, pipe_tables_extension)
13766 end

```

```

13767
13768   if options.rawAttribute then
13769       local raw_inline_extension = M.extensions.raw_inline()
13770       table.insert(extensions, raw_inline_extension)
13771   end
13772
13773   if options.strikeThrough then
13774       local strike_through_extension = M.extensions.strike_through()
13775       table.insert(extensions, strike_through_extension)
13776   end
13777
13778   if options.subscripts then
13779       local subscript_extension = M.extensions.subscripts()
13780       table.insert(extensions, subscript_extension)
13781   end
13782
13783   if options.superscripts then
13784       local superscript_extension = M.extensions.superscripts()
13785       table.insert(extensions, superscript_extension)
13786   end
13787
13788   if options.texMathDollars or
13789       options.texMathSingleBackslash or
13790       options.texMathDoubleBackslash then
13791       local tex_math_extension = M.extensions.tex_math(
13792           options.texMathDollars,
13793           options.texMathSingleBackslash,
13794           options.texMathDoubleBackslash)
13795       table.insert(extensions, tex_math_extension)
13796   end
13797
13798   if options.notes or options.inlineNotes then
13799       local notes_extension = M.extensions.notes(
13800           options.notes, options.inlineNotes)
13801       table.insert(extensions, notes_extension)
13802   end
13803
13804   if options.citations then
13805       local citations_extension
13806           = M.extensions.citations(options.citationNbsps)
13807       table.insert(extensions, citations_extension)
13808   end
13809
13810   if options.fancyLists then
13811       local fancy_lists_extension = M.extensions.fancy_lists()
13812       table.insert(extensions, fancy_lists_extension)
13813   end

```

Apply user-defined syntax extensions based on `options.extensions`.

```
13814 for _, user_extension_filename in ipairs(options.extensions) do
13815     local user_extension = (function(filename)
```

First, load and compile the contents of the user-defined syntax extension.

```
13816         local pathname = assert(kpse.find_file(filename),
13817             [[Could not locate user-defined syntax extension "]]
13818             .. filename)
13819         local input_file = assert(io.open(pathname, "r"),
13820             [[Could not open user-defined syntax extension "]]
13821             .. pathname .. [{" for reading}])
13822         local input = assert(input_file:read("*a"))
13823         assert(input_file:close())
13824         local user_extension, err = load([[
13825             local sandbox = {}
13826             setmetatable(sandbox, {_index = _G})
13827             _ENV = sandbox
13828         ]]) .. input)()
13829         assert(user_extension,
13830             [[Failed to compile user-defined syntax extension "]]
13831             .. pathname .. [{": }]] .. (err or [{"}]))
```

Then, validate the user-defined syntax extension.

```
13832         assert(user_extension.api_version ~= nil,
13833             [[User-defined syntax extension "]] .. pathname
13834             .. [{" does not specify mandatory field "api_version"}])
13835         assert(type(user_extension.api_version) == "number",
13836             [[User-defined syntax extension "]] .. pathname
13837             .. [{" specifies field "api_version" of type "]]
13838             .. type(user_extension.api_version)
13839             .. [{" but "number" was expected}])
13840         assert(user_extension.api_version > 0
13841             and user_extension.api_version
13842             <= metadata.user_extension_api_version,
13843             [[User-defined syntax extension "]] .. pathname
13844             .. [{" uses syntax extension API version "]]
13845             .. user_extension.api_version .. [{" but markdown.lua ]]
13846             .. metadata.version .. [{" uses API version ]]
13847             .. metadata.user_extension_api_version
13848             .. [{" , which is incompatible}])
13849
13850         assert(user_extension.grammar_version ~= nil,
13851             [[User-defined syntax extension "]] .. pathname
13852             .. [{" does not specify mandatory field "grammar_version"}])
13853         assert(type(user_extension.grammar_version) == "number",
13854             [[User-defined syntax extension "]] .. pathname
13855             .. [{" specifies field "grammar_version" of type "]]
13856             .. type(user_extension.grammar_version)
```

```

13857     .. [{" but "number" was expected}]
13858     assert(user_extension.grammar_version == metadata.grammar_version,
13859           [[User-defined syntax extension "]] .. pathname
13860           .. [{" uses grammar version "}]
13861           .. user_extension.grammar_version
13862           .. [{" but markdown.lua "}] .. metadata.version
13863           .. [{" uses grammar version "}] .. metadata.grammar_version
13864           .. [{" , which is incompatible}])
13865
13866     assert(user_extension.finalize_grammar ~= nil,
13867           [[User-defined syntax extension "]] .. pathname
13868           .. [{" does not specify mandatory "finalize_grammar" field}])
13869     assert(type(user_extension.finalize_grammar) == "function",
13870           [[User-defined syntax extension "]] .. pathname
13871           .. [{" specifies field "finalize_grammar" of type "}]
13872           .. type(user_extension.finalize_grammar)
13873           .. [{" 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.)

```

13874     local extension = {
13875       name = [[user-defined "]] .. pathname .. [{" syntax extension}],
13876       extend_reader = user_extension.finalize_grammar,
13877       extend_writer = function() end,
13878     }
13879     return extension
13880   end)(user_extension_filename)
13881   table.insert(extensions, user_extension)
13882 end

```

Produce a conversion function from markdown to plain TeX.

```

13883 local writer = M.writer.new(options)
13884 local reader = M.reader.new(writer, options)
13885 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`.

```

13886 collectgarbage("collect")

```

Update the singleton cache.

```

13887 if options.singletonCache then
13888   local singletonCacheOptions = {}
13889   for k, v in pairs(options) do
13890     singletonCacheOptions[k] = v
13891   end
13892   setmetatable(singletonCacheOptions,
13893     { __index = function (_, key)
13894       return defaultOptions[key] end })

```

```

13895     singletonCache.options = singletonCacheOptions
13896     singletonCache.convert = convert
13897 end

```

Return the conversion function from markdown to plain  $\text{\TeX}$ .

```

13898 return convert
13899 end

13900 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.

```

13901
13902 local input
13903 if input_filename then
13904     local input_file = assert(io.open(input_filename, "r"),
13905         [[Could not open file ]] .. input_filename .. [[ for reading]])
13906     input = assert(input_file:read("*a"))
13907     assert(input_file:close())
13908 else
13909     input = assert(io.read("*a"))
13910 end
13911

```

First, ensure that the `options.cacheDir` directory exists.

```

13912 local lfs = require("lfs")
13913 if options.cacheDir and not lfs.isdir(options.cacheDir) then
13914     assert(lfs.mkdir(options["cacheDir"]))
13915 end

```

If Kpathsea has not been loaded before or if Lua $\text{\TeX}$  has not yet been initialized, configure Kpathsea on top of loading it.

```

13916 local kpse
13917 (function()
13918     local should_initialize = package.loaded.kpse == nil
13919         or tex.initialize ~= nil
13920     kpse = require("kpse")
13921     if should_initialize then
13922         kpse.set_program_name("luatex")
13923     end
13924 end)()
13925 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.

```

13926 if metadata.version ~= md.metadata.version then

```

```

13927 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
13928         "markdown.lua " .. md.metadata.version .. ".")
13929 end
13930
13931 local convert = md.new(options)
13932 local raw_output, flat_output = convert(input, true)
13933 local output
13934 if flat_output == nil then
13935     if options.eagerCache then
13936         warn("markdown.lua has not produced flat output, so I am using " ..
13937             "backwards-compatible raw output instead. This may cause " ..
13938             "the conversion result to be hidden behind "\\input'.')
13939     end
13940     output = raw_output
13941 else
13942     output = flat_output()
13943 end
13944
13945 if output_filename then
13946     local output_file = assert(io.open(output_filename, "w"),
13947         [[Could not open file ]] .. output_filename .. [[ for writing]])
13948     assert(output_file:write(output))
13949     assert(output_file:close())
13950 else
13951     assert(io.write(output))
13952 end

```

Remove the `options.cacheDir` directory if it is empty.

```

13953 if options.cacheDir then
13954     lfs.rmdir(options.cacheDir)
13955 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

```

13956 \ExplSyntaxOn
13957 \cs_if_free:NT
13958   \markdownInfo
13959   {
13960     \cs_new:Npn
13961       \markdownInfo #1

```

```

13962     {
13963         \msg_info:nne
13964         { markdown }
13965         { generic-message }
13966         { #1 }
13967     }
13968 }
13969 \cs_if_free:NT
13970 \markdownWarning
13971 {
13972     \cs_new:Npn
13973     \markdownWarning #1
13974     {
13975         \msg_warning:nne
13976         { markdown }
13977         { generic-message }
13978         { #1 }
13979     }
13980 }
13981 \cs_if_free:NT
13982 \markdownError
13983 {
13984     \cs_new:Npn
13985     \markdownError #1 #2
13986     {
13987         \msg_error:nnee
13988         { markdown }
13989         { generic-message-with-help-text }
13990         { #1 }
13991         { #2 }
13992     }
13993 }
13994 \msg_new:nnn
13995 { markdown }
13996 { generic-message }
13997 { #1 }
13998 \msg_new:nnnn
13999 { markdown }
14000 { generic-message-with-help-text }
14001 { #1 }
14002 { #2 }
14003 \cs_generate_variant:Nn
14004 \msg_info:nnn
14005 { nne }
14006 \cs_generate_variant:Nn
14007 \msg_warning:nnn
14008 { nne }

```

```

14009 \cs_generate_variant:Nn
14010   \msg_error:nnnn
14011   { nnee }
14012 \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.

```

14013 \ExplSyntaxOn
14014 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
14015 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
14016 \cs_new:Nn
14017   \@@_plain_tex_load_theme:nnn
14018   {
14019     \prop_get:NnNTF
14020       \g_@@_plain_tex_loaded_themes_linenos_prop
14021       { #1 }
14022     \l_tmpa_tl
14023     {
14024       \prop_get:NnN
14025         \g_@@_plain_tex_loaded_themes_versions_prop
14026         { #1 }
14027       \l_tmpb_tl
14028       \str_if_eq:nVTF
14029         { #2 }
14030         \l_tmpb_tl
14031       {
14032         \msg_warning:nnnVn
14033           { markdown }
14034           { repeatedly-loaded-plain-tex-theme }
14035           { #1 }
14036         \l_tmpa_tl
14037         { #2 }
14038       }
14039     }
14040     \msg_error:nnnnVV
14041       { markdown }
14042       { different-versions-of-plain-tex-theme }
14043       { #1 }
14044       { #2 }
14045     \l_tmpb_tl
14046     \l_tmpa_tl
14047   }
14048 }
14049 {

```



```

14050     \prop_gput:Nnx
14051     \g_@@_plain_tex_loaded_themes_linenos_prop
14052     { #1 }
14053     { \tex_the:D \tex_inputlineno:D } % noqa: W200
14054     \prop_gput:Nnn
14055     \g_@@_plain_tex_loaded_themes_versions_prop
14056     { #1 }
14057     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

14058     \prop_if_in:NnTF
14059     \g_@@_plain_tex_built_in_themes_prop
14060     { #1 }
14061     {
14062         \msg_info:nnnn
14063         { markdown }
14064         { loading-built-in-plain-tex-theme }
14065         { #1 }
14066         { #2 }
14067         \prop_item:Nn
14068         \g_@@_plain_tex_built_in_themes_prop
14069         { #1 }
14070     }
14071     {
14072         \msg_info:nnnn
14073         { markdown }
14074         { loading-plain-tex-theme }
14075         { #1 }
14076         { #2 }
14077         \file_input:n
14078         { markdown theme #3 }
14079     }
14080 }
14081 }
14082 \msg_new:nnn
14083 { markdown }
14084 { loading-plain-tex-theme }
14085 { Loading-version-#2-of-plain-TeX-Markdown-theme-#1 }
14086 \msg_new:nnn
14087 { markdown }
14088 { loading-built-in-plain-tex-theme }
14089 { Loading-version-#2-of-built-in-plain-TeX-Markdown-theme-#1 }
14090 \msg_new:nnn
14091 { markdown }
14092 { repeatedly-loaded-plain-tex-theme }
14093 {

```

```

14094     Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
14095     loaded~on~line~#2,~not~loading~it~again
14096   }
14097   \msg_new:nnn
14098     { markdown }
14099     { different-versions-of-plain-tex-theme }
14100     {
14101       Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
14102       but~version~#3~has~already~been~loaded~on~line~#4
14103     }
14104   \cs_generate_variant:Nn
14105     \prop_gput:Nnn
14106     { Nnx }
14107   \cs_gset_eq:NN
14108     \@@_load_theme:nnn
14109     \@@_plain_tex_load_theme:nnn
14110   \cs_generate_variant:Nn
14111     \@@_load_theme:nnn
14112     { VeV }
14113   \cs_generate_variant:Nn
14114     \msg_error:nnnnn
14115     { nnnnVV }
14116   \cs_generate_variant:Nn
14117     \msg_warning:nnnnn
14118     { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T<sub>E</sub>X theme from within themes for higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

14119   \cs_new:Npn
14120     \markdownLoadPlainTeXTheme
14121     {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

14122     \tl_set:NV
14123       \l_tmpa_tl
14124       \g_@@_current_theme_tl
14125     \tl_reverse:N
14126       \l_tmpa_tl
14127     \tl_set:Ne
14128       \l_tmpb_tl
14129     {
14130       \tl_tail:V
14131         \l_tmpa_tl
14132     }
14133     \tl_reverse:N
14134       \l_tmpb_tl

```

Next, we munge the theme name.

```
14135 \str_set:NV
14136 \l_tmpa_str
14137 \l_tmpb_tl
14138 \str_replace_all:Nnn
14139 \l_tmpa_str
14140 { / }
14141 { _ }
```

Finally, we load the plain TeX theme.

```
14142 \@@_plain_tex_load_theme:VeV
14143 \l_tmpb_tl
14144 { \markdownThemeVersion }
14145 \l_tmpa_str
14146 }
14147 \cs_generate_variant:Nn
14148 \tl_set:Nn
14149 { Ne }
14150 \cs_generate_variant:Nn
14151 \@@_plain_tex_load_theme:nnn
14152 { VeV }
```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```
14153 \prop_gput:Nnn
14154 \g_@@_plain_tex_built_in_themes_prop
14155 { witiko / dot }
14156 {
14157 \str_if_eq:enF
14158 { \markdownThemeVersion }
14159 { silent }
14160 {
14161 \markdownWarning
14162 {
14163 The~theme~name~"witiko/dot"~has~been~soft-deprecated.
14164 \iow_newline:
14165 Consider~changing~the~name~to~"witiko/diagrams@v1".
14166 }
14167 }
```

We enable the `fencedCode` Lua option.

```
14168 \markdownSetup { fencedCode }
```

We store the previous definition of the fenced code token renderer prototype:

```
14169 \cs_set_eq:NN
14170 \@@_dot_previous_definition:nnn
14171 \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:

```

14172 \regex_const:Nn
14173   \c_@@_dot_infostring_regex
14174   { ^dot(\s+(.+)?)? }
14175 \seq_new:N
14176   \l_@@_dot_matches_seq
14177 \markdownSetup {
14178   rendererPrototypes = {
14179     inputFencedCode = {
14180       \regex_extract_once:NnNTF
14181         \c_@@_dot_infostring_regex
14182         { #2 }
14183       \l_@@_dot_matches_seq
14184       {
14185         \@@_if_option:nF
14186         { frozenCache }
14187         {
14188           \sys_shell_now:n
14189           {
14190             if~!~test~-e~#1.pdf.source~
14191             ||~!~diff~#1~#1.pdf.source;
14192             then~
14193               dot~-Tpdf~-o~#1.pdf~#1;
14194               cp~#1~#1.pdf.source;
14195             fi
14196           }
14197         }

```

We include the typeset image using the image token renderer:

```

14198   \exp_args:NNne
14199   \exp_last_unbraced:No
14200   \markdownRendererImage
14201   {
14202     { Graphviz~image }
14203     { #1.pdf }
14204     { #1.pdf }
14205   }
14206   {
14207     \seq_item:Nn
14208     \l_@@_dot_matches_seq
14209     { 3 }
14210   }
14211 }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

14212         {
14213             \@@_dot_previous_definition:nnn
14214             { #1 }
14215             { #2 }
14216             { #3 }
14217         }
14218     },
14219 },
14220 }
14221 }

```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

14222 \prop_gput:Nnn
14223 \g_@@_plain_tex_built_in_themes_prop
14224 { witiko / diagrams }
14225 {
14226     \str_case:enF
14227     { \markdownThemeVersion }
14228     {
14229         { latest }
14230         {
14231             \markdownWarning
14232             {
14233                 Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
14234                 theme~"witiko/diagrams".~This~will~keep~your~documents~
14235                 from~suddenly~breaking~when~we~have~released~future~
14236                 versions~of~the~theme~with~backwards~incompatible~
14237                 syntax~and~behavior.
14238             }
14239             \markdownSetup
14240             {
14241                 import = witiko/diagrams/v2,
14242             }
14243         }
14244         { v2 }
14245         {
14246             \markdownSetup
14247             {
14248                 import = witiko/diagrams/v2,
14249             }
14250         }
14251         { v1 }
14252         {
14253             \markdownSetup

```

```

14254         {
14255             import = witiko/dot@silent,
14256         }
14257     }
14258 }
14259 {
14260     \msg_error:nnen
14261     { markdown }
14262     { unknown-theme-version }
14263     { witiko/diagrams }
14264     { \markdownThemeVersion }
14265     { v1 }
14266 }
14267 }
14268 \cs_generate_variant:Nn
14269 \msg_error:nnnnn
14270 { nnnen }
14271 \msg_new:nnnn
14272 { markdown }
14273 { unknown-theme-version }
14274 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
14275 { Known~versions~are:~#3 }

```

Next, we implement the theme `witiko/diagrams/v2`.

```

14276 \prop_gput:Nnn
14277 \g_@@_plain_tex_built_in_themes_prop
14278 { witiko / diagrams / v2 }
14279 {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

14280 \@@_setup:n
14281 {
14282     fencedCode = true,
14283     fencedCodeAttributes = true,
14284 }

```

Store the previous fenced code token renderer prototype.

```

14285 \cs_set_eq:NN
14286 \@@_diagrams_previous_fenced_code:nnn
14287 \markdownRendererInputFencedCodePrototype

```

Store the caption and the desired format of the diagram.

```

14288 \tl_new:N
14289 \l_@@_diagrams_caption_tl
14290 \tl_new:N
14291 \l_@@_diagrams_format_tl
14292 \tl_set:Nn
14293 \l_@@_diagrams_format_tl
14294 { pdf }

```

```

14295 \@@_setup:n
14296 {
14297     rendererPrototypes = {
Route attributes on fenced code blocks to the image attribute renderer prototypes.
14298     fencedCodeAttributeContextBegin = {
14299         \group_begin:
14300         \markdownRendererImageAttributeContextBegin
14301         \cs_set_eq:NN
14302         \@@_diagrams_previous_key_value:nn
14303         \markdownRendererAttributeKeyValuePrototype
14304         \@@_setup:n
14305         {
14306             rendererPrototypes = {
14307                 attributeKeyValue = {
14308                     \str_case:nnF
14309                     { ##1 }
14310                     {
14311                         { caption }
14312                         {
14313                             \tl_set:Nn
14314                             \l_@@_diagrams_caption_tl
14315                             { ##2 }
14316                         }
14317                         { format }
14318                         {
14319                             \tl_set:Nn
14320                             \l_@@_diagrams_format_tl
14321                             { ##2 }
14322                         }
14323                     }
14324                     {
14325                         \@@_diagrams_previous_key_value:nn
14326                         { ##1 }
14327                         { ##2 }
14328                     }
14329                 },
14330             },
14331         }
14332     },
14333     fencedCodeAttributeContextEnd = {
14334         \markdownRendererImageAttributeContextEnd
14335         \group_end:
14336     },
14337 },
14338 }
14339 \cs_new:Nn
14340 \@@_diagrams_render_diagram:nnnn

```

```

14341     {
14342     \@@_if_option:nF
14343     { frozenCache }
14344     {
14345     \sys_shell_now:n
14346     {
14347     if~!~test~-e~#2.source~
14348     ||~!~diff~#1~#2.source;
14349     then~
14350     (#3);
14351     cp~#1~#2.source;
14352     fi
14353     }
14354     \exp_args:NNnV
14355     \exp_last_unbraced:No
14356     \markdownRendererImage
14357     {
14358     { #4 }
14359     { #2 }
14360     { #2 }
14361     }
14362     \l_@@_diagrams_caption_tl
14363     }
14364     }

```

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.

```

14365     \prop_new:N
14366     \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

14367     \@@_setup:n
14368     {
14369     rendererPrototypes = {
14370     inputFencedCode = {
14371     \prop_get:NnNTF
14372     \g_markdown_diagrams_infostrings_prop
14373     { #2 }
14374     \l_tmpa_tl
14375     {
14376     \cs_set:NV
14377     \@@_diagrams_infostrings_current:n
14378     \l_tmpa_tl
14379     \@@_diagrams_infostrings_current:n
14380     { #1 }
14381     }

```



Otherwise, use the previous fenced code token renderer prototype.

```
14382         {
14383             \@@_diagrams_previous_fenced_code:nm
14384             { #1 }
14385             { #2 }
14386             { #3 }
14387         }
14388     },
14389 },
14390 }
14391 \cs_generate_variant:Nn
14392   \cs_set:Nn
14393   { NV }
```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```
14394   \cs_set:Nn
14395     \@@_diagrams_infostrings_current:n
14396     {
14397       \@@_diagrams_render_diagram:nmm
14398       { #1 }
14399       { #1.pdf }
14400       { dot~Tpdf~o~#1.pdf~#1 }
14401       { Graphviz~image }
14402     }
14403   \@@_tl_set_from_cs:NNn
14404     \l_tmpa_tl
14405     \@@_diagrams_infostrings_current:n
14406     { 1 }
14407   \prop_gput:NnV
14408     \g_markdown_diagrams_infostrings_prop
14409     { dot }
14410     \l_tmpa_tl
```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`.

```
14411   \cs_set:Nn
14412     \@@_diagrams_infostrings_current:n
14413     {
14414       \@@_diagrams_render_diagram:nmm
14415       { #1 }
14416       { #1.pdf }
14417       { mmdc~---pdfFit~i~#1~o~#1.pdf }
14418       { Mermaid~image }
14419     }
14420   \@@_tl_set_from_cs:NNn
14421     \l_tmpa_tl
```

```

14422     \@@_diagrams_infostrings_current:n
14423     { 1 }
14424     \prop_gput:NnV
14425     \g_markdown_diagrams_infostrings_prop
14426     { mermaid }
14427     \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

14428     \regex_const:Nn
14429     \c_@@_diagrams_filename_suffix_regex
14430     { \.[^\.]*$ }
14431     \cs_set:Nn
14432     \@@_diagrams_infostrings_current:n
14433     {

```

Use the output format provided by the user.

```

14434     \tl_set:Nn
14435     \l_tmpa_tl
14436     { #1 }
14437     \regex_replace_once:NxN
14438     \c_@@_diagrams_filename_suffix_regex
14439     {
14440     .
14441     \tl_use:N
14442     \l_@@_diagrams_format_tl
14443     }
14444     \l_tmpa_tl
14445     \tl_set:Nn
14446     \l_tmpb_tl
14447     { plantuml~-t }
14448     \tl_put_right:NV
14449     \l_tmpb_tl
14450     \l__markdown_diagrams_format_tl
14451     \tl_put_right:Nn
14452     \l_tmpb_tl
14453     { ~#1 }

```

For the SVG format, use Inkscape to convert the resulting image to PDF.

```

14454     \str_if_eq:VnT
14455     \l_@@_diagrams_format_tl
14456     { svg }
14457     {
14458     \tl_put_right:Nn
14459     \l_tmpb_tl
14460     { ;~inkscape~ }
14461     \tl_put_right:NV
14462     \l_tmpb_tl

```

```

14463         \l_tmpa_tl
14464     \tl_put_right:Nn
14465         \l_tmpb_tl
14466     { ---export-area-drawing---export-dpi=300--o~ }
14467     \tl_set:Nn
14468         \l_tmpa_tl
14469         { #1 }
14470     \regex_replace_once:NnN
14471         \c_@@_diagrams_filename_suffix_regex
14472         { .pdf }
14473         \l_tmpa_tl
14474     \tl_put_right:NV
14475         \l_tmpb_tl
14476         \l_tmpa_tl
14477     }
14478     \@@_diagrams_render_diagram:nVVn
14479     { #1 }
14480     \l_tmpa_tl
14481     \l_tmpb_tl
14482     { PlantUML~image }
14483 }
14484 \cs_generate_variant:Nn
14485     \@@_diagrams_render_diagram:nnnn
14486     { nVVn }
14487 \cs_generate_variant:Nn
14488     \regex_replace_once:NnN
14489     { NxN }
14490 \@@_tl_set_from_cs:NNn
14491     \l_tmpa_tl
14492     \@@_diagrams_infostings_current:n
14493     { 1 }
14494 \prop_gput:NnV
14495     \g_markdown_diagrams_infostings_prop
14496     { plantuml }
14497     \l_tmpa_tl
14498 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

14499 \group_begin:
14500 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```

14501 \prop_gput:Nnn
14502     \g_@@_plain_tex_built_in_themes_prop
14503     { witiko / graphicx / http }
14504     {

```

```

14505 \cs_set_eq:NN
14506 \g_@@_graphicx_http_previous_definition:nmmn
14507 \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.

```

14508 \int_new:N
14509 \g_@@_graphicx_http_image_number_int
14510 \int_gset:Nn
14511 \g_@@_graphicx_http_image_number_int
14512 { 0 }
14513 \cs_new:Nn
14514 \@@_graphicx_http_filename:
14515 {
14516 \markdownOptionCacheDir
14517 / witiko_graphicx_http .
14518 \int_use:N
14519 \g_@@_graphicx_http_image_number_int
14520 }

```

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.

```

14521 \cs_new:Nn
14522 \@@_graphicx_http_download:nn
14523 {
14524 wget~-0~#2~#1~
14525 ||~curl~--location~-o~#2~#1~
14526 ||~rm~-f~#2
14527 }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

14528 \str_new:N
14529 \l_@@_graphicx_http_filename_str
14530 \ior_new:N
14531 \g_@@_graphicx_http_filename_ior
14532 \markdownSetup {
14533 rendererPrototypes = {
14534 image = {
14535 \@@_if_option:nF
14536 { frozenCache }
14537 {

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```

14538 \sys_shell_now:e

```

```

14539         {
14540             mkdir~-p~" \markdownOptionCacheDir ";
14541             if~printf~'%s'~"#3"~|~grep~q~-E~'^https?:';
14542             then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

14543             OUTPUT_PREFIX=" \markdownOptionCacheDir ";
14544             OUTPUT_BODY="$(printf~'%s'~'#3'
14545                 |~md5sum~|~cut~-d~'~'~-f1)";
14546             OUTPUT_SUFFIX="$(printf~'%s'~'#3'
14547                 |~sed~'s/.*[.]//')";
14548             OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

14549             if~!~[~-e~"$OUTPUT"~];
14550             then~
14551                 \@@_graphicx_http_download:nn
14552                 { '#3' }
14553                 { "$OUTPUT" } ;
14554                 printf~'%s'~"$OUTPUT"~
14555                 >~" \@@_graphicx_http_filename: ";
14556             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:

```

14557             else~
14558                 printf~'%s'~'#3'~
14559                 >~" \@@_graphicx_http_filename: ";
14560             fi
14561         }
14562     }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

14563         \ior_open:Ne
14564         \g_@@_graphicx_http_filename_ior
14565         { \@@_graphicx_http_filename: }
14566         \ior_str_get:NN
14567         \g_@@_graphicx_http_filename_ior
14568         \l_@@_graphicx_http_filename_str
14569         \ior_close:N
14570         \g_@@_graphicx_http_filename_ior
14571         \@@_graphicx_http_previous_definition:nnVn
14572         { #1 }
14573         { #2 }
14574         \l_@@_graphicx_http_filename_str
14575         { #4 }

```

```

14576         \int_gincr:N
14577         \g_@@_graphicx_http_image_number_int
14578     }
14579 }
14580 }
14581 \cs_generate_variant:Nn
14582   \ior_open:Nn
14583   { Ne }
14584 \cs_generate_variant:Nn
14585   \@@_graphicx_http_previous_definition:nnnn
14586   { nnVn }
14587 }
14588 \group_end:

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

14589 \prop_gput:Nnn
14590   \g_@@_plain_tex_built_in_themes_prop
14591   { witiko / tilde }
14592   {
14593     \markdownSetup {
14594       rendererPrototypes = {
14595         tilde = {-},
14596       },
14597     }
14598   }

```

The themes `witiko/example/foo` and `witiko/example/bar` are supposed to be used in code examples. They don't do anything.

```

14599 \clist_map_inline:nn
14600   { foo, bar }
14601   {
14602     \prop_gput:Nnn
14603       \g_@@_plain_tex_built_in_themes_prop
14604       { witiko / example / #1 }
14605       {
14606         \markdownWarning
14607         {
14608           The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
14609           examples.~Using~it~in~actual~code~has~no~effect,~except~
14610           this~warning~message,~and~is~usually~a~mistake.
14611         }
14612       }
14613     }
14614 \ExplSyntaxOff

```

The `witiko/markdown/defaults` plain T<sub>E</sub>X theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
14615 \def\markdownRendererInterblockSeparatorPrototype{\par}%
14616 \def\markdownRendererParagraphSeparatorPrototype{%
14617   \markdownRendererInterblockSeparator}%
14618 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
14619 \def\markdownRendererSoftLineBreakPrototype{ }%
14620 \let\markdownRendererEllipsisPrototype\dots
14621 \def\markdownRendererNbspPrototype{~}%
14622 \def\markdownRendererLeftBracePrototype{\char` \}%
14623 \def\markdownRendererRightBracePrototype{\char` \}%
14624 \def\markdownRendererDollarSignPrototype{\char`$}%
14625 \def\markdownRendererPercentSignPrototype{\char`}%
14626 \def\markdownRendererAmpersandPrototype{&}%
14627 \def\markdownRendererUnderscorePrototype{\char`_}%
14628 \def\markdownRendererHashPrototype{\char`#}%
14629 \def\markdownRendererCircumflexPrototype{\char`^}%
14630 \def\markdownRendererBackslashPrototype{\char`\}%
14631 \def\markdownRendererTildePrototype{\char`~}%
14632 \def\markdownRendererPipePrototype{|}%
14633 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
14634 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
14635 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
14636   \markdownInput{#3}}%
14637 \def\markdownRendererContentBlockOnlineImagePrototype{%
14638   \markdownRendererImage}%
14639 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
14640   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
14641 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
14642 \def\markdownRendererUlBeginPrototype{}%
14643 \def\markdownRendererUlBeginTightPrototype{}%
14644 \def\markdownRendererUlItemPrototype{}%
14645 \def\markdownRendererUlItemEndPrototype{}%
14646 \def\markdownRendererUlEndPrototype{}%
14647 \def\markdownRendererUlEndTightPrototype{}%
14648 \def\markdownRendererOlBeginPrototype{}%
14649 \def\markdownRendererOlBeginTightPrototype{}%
14650 \def\markdownRendererFancyOlBeginPrototype#1#2{%
14651   \markdownRendererOlBegin}%
14652 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
14653   \markdownRendererOlBeginTight}%
14654 \def\markdownRendererOlItemPrototype{}%
14655 \def\markdownRendererOlItemWithNumberPrototype#1{}%
14656 \def\markdownRendererOlItemEndPrototype{}%
14657 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
14658 \def\markdownRendererFancyOlItemWithNumberPrototype{%
```

```

14659 \markdownRendererOlItemWithNumber}%
14660 \def\markdownRendererFancyOlItemEndPrototype{}%
14661 \def\markdownRendererOlEndPrototype{}%
14662 \def\markdownRendererOlEndTightPrototype{}%
14663 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
14664 \def\markdownRendererFancyOlEndTightPrototype{%
14665 \markdownRendererOlEndTight}%
14666 \def\markdownRendererDlBeginPrototype{}%
14667 \def\markdownRendererDlBeginTightPrototype{}%
14668 \def\markdownRendererDlItemPrototype#1{#1}%
14669 \def\markdownRendererDlItemEndPrototype{}%
14670 \def\markdownRendererDlDefinitionBeginPrototype{}%
14671 \def\markdownRendererDlDefinitionEndPrototype{\par}%
14672 \def\markdownRendererDlEndPrototype{}%
14673 \def\markdownRendererDlEndTightPrototype{}%
14674 \def\markdownRendererEmphasisPrototype#1{\it#1}%
14675 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
14676 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
14677 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
14678 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
14679 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
14680 \def\markdownRendererInputVerbatimPrototype#1{%
14681 \par{\tt\input#1\relax}}\par}%
14682 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
14683 \markdownRendererInputVerbatim{#1}}%
14684 \def\markdownRendererHeadingOnePrototype#1{#1}%
14685 \def\markdownRendererHeadingTwoPrototype#1{#1}%
14686 \def\markdownRendererHeadingThreePrototype#1{#1}%
14687 \def\markdownRendererHeadingFourPrototype#1{#1}%
14688 \def\markdownRendererHeadingFivePrototype#1{#1}%
14689 \def\markdownRendererHeadingSixPrototype#1{#1}%
14690 \def\markdownRendererThematicBreakPrototype{}%
14691 \def\markdownRendererNotePrototype#1{#1}%
14692 \def\markdownRendererCitePrototype#1{}%
14693 \def\markdownRendererTextCitePrototype#1{}%
14694 \def\markdownRendererTickedBoxPrototype{[X]}%
14695 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
14696 \def\markdownRendererUntickedBoxPrototype{[ ]}%
14697 \def\markdownRendererStrikeThroughPrototype#1{#1}%
14698 \def\markdownRendererSuperscriptPrototype#1{#1}%
14699 \def\markdownRendererSubscriptPrototype#1{#1}%
14700 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
14701 \def\markdownRendererInlineMathPrototype#1{$#1$}%
14702 \ExplSyntaxOn
14703 \cs_gset:Npn
14704 \markdownRendererHeaderAttributeContextBeginPrototype
14705 {

```



```

14706     \group_begin:
14707     \color_group_begin:
14708   }
14709 \cs_gset:Npn
14710   \markdownRendererHeaderAttributeContextEndPrototype
14711   {
14712     \color_group_end:
14713     \group_end:
14714   }
14715 \cs_gset_eq:NN
14716   \markdownRendererBracketedSpanAttributeContextBeginPrototype
14717   \markdownRendererHeaderAttributeContextBeginPrototype
14718 \cs_gset_eq:NN
14719   \markdownRendererBracketedSpanAttributeContextEndPrototype
14720   \markdownRendererHeaderAttributeContextEndPrototype
14721 \cs_gset_eq:NN
14722   \markdownRendererFencedDivAttributeContextBeginPrototype
14723   \markdownRendererHeaderAttributeContextBeginPrototype
14724 \cs_gset_eq:NN
14725   \markdownRendererFencedDivAttributeContextEndPrototype
14726   \markdownRendererHeaderAttributeContextEndPrototype
14727 \cs_gset_eq:NN
14728   \markdownRendererFencedCodeAttributeContextBeginPrototype
14729   \markdownRendererHeaderAttributeContextBeginPrototype
14730 \cs_gset_eq:NN
14731   \markdownRendererFencedCodeAttributeContextEndPrototype
14732   \markdownRendererHeaderAttributeContextEndPrototype
14733 \cs_gset:Npn
14734   \markdownRendererReplacementCharacterPrototype
14735   { \codepoint_str_generate:n { fffd } }
14736 \ExplSyntaxOff
14737 \def\markdownRendererSectionBeginPrototype{}%
14738 \def\markdownRendererSectionEndPrototype{}%
14739 \ExplSyntaxOn
14740 \cs_gset:Npn
14741   \markdownRendererWarningPrototype
14742   #1#2#3#4
14743   {
14744     \tl_set:Nn
14745       \l_tmpa_tl
14746       { #2 }
14747     \tl_if_empty:nF
14748       { #4 }
14749     {
14750       \tl_put_right:Nn
14751         \l_tmpa_tl
14752         { \iow_newline: #4 }

```

```

14753     }
14754     \exp_args:NV
14755     \markdownWarning
14756     \l_tmpa_tl
14757   }
14758 \ExplSyntaxOff
14759 \def\markdownRendererErrorPrototype#1#2#3#4{%
14760   \markdownError{#2}{#4}}%

```

### 3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

14761 \ExplSyntaxOn
14762 \cs_new:Nn
14763   \@@_plain_tex_default_input_raw_inline:nn
14764   {
14765     \str_case:nn
14766       { #2 }
14767       {
14768         { md } { \markdownInput{#1} }
14769         { tex } { \markdownEscape{#1} \unskip }
14770       }
14771   }
14772 \cs_new:Nn
14773   \@@_plain_tex_default_input_raw_block:nn
14774   {
14775     \str_case:nn
14776       { #2 }
14777       {
14778         { md } { \markdownInput{#1} }
14779         { tex } { \markdownEscape{#1} }
14780       }
14781   }
14782 \cs_gset:Npn
14783   \markdownRendererInputRawInlinePrototype#1#2
14784   {
14785     \@@_plain_tex_default_input_raw_inline:nn
14786       { #1 }
14787       { #2 }
14788   }
14789 \cs_gset:Npn
14790   \markdownRendererInputRawBlockPrototype#1#2
14791   {
14792     \@@_plain_tex_default_input_raw_block:nn
14793       { #1 }

```

```

14794     { #2 }
14795   }
14796 \ExplSyntaxOff

```

### 3.2.3.2 Simple YAML Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key-value `markdown/jekyllData`. See also Section 2.2.6.1.

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

14797 \ExplSyntaxOn
14798 \seq_new:N \g_@@_jekyll_data_datatypes_seq
14799 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
14800 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
14801 \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.

```

14802 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
14803 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
14804 {
14805   \seq_if_empty:NF
14806     \g_@@_jekyll_data_datatypes_seq
14807     {
14808       \seq_get_right:NN
14809       \g_@@_jekyll_data_datatypes_seq
14810       \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.

```

14811   \str_if_eq:NNTF

```

```

14812     \l_tmpa_tl
14813     \c_@@_jekyll_data_sequence_tl
14814     {
14815         \seq_put_right:Nn
14816         \g_@@_jekyll_data_wildcard_absolute_address_seq
14817         { * }
14818     }
14819     {
14820         \seq_put_right:Nn
14821         \g_@@_jekyll_data_wildcard_absolute_address_seq
14822         { #1 }
14823     }
14824 }
14825 }

```

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.

```

14826 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
14827 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
14828 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
14829 {
14830     \seq_pop_left:NN #1 \l_tmpa_tl
14831     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }

```

```

14832     \seq_put_left:NV #1 \l_tmpa_tl
14833   }
14834 \cs_new:Nn \@@_jekyll_data_update_address_tls:
14835   {
14836     \@@_jekyll_data_concatenate_address:NN
14837     \g_@@_jekyll_data_wildcard_absolute_address_seq
14838     \g_@@_jekyll_data_wildcard_absolute_address_tl
14839     \seq_get_right:NN
14840     \g_@@_jekyll_data_wildcard_absolute_address_seq
14841     \g_@@_jekyll_data_wildcard_relative_address_tl
14842   }

```

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.

```

14843 \cs_new:Nn \@@_jekyll_data_push:nN
14844   {
14845     \@@_jekyll_data_push_address_segment:n
14846     { #1 }
14847     \seq_put_right:NV
14848     \g_@@_jekyll_data_datatypes_seq
14849     #2
14850     \@@_jekyll_data_update_address_tls:
14851   }
14852 \cs_new:Nn \@@_jekyll_data_pop:
14853   {
14854     \seq_pop_right:NN
14855     \g_@@_jekyll_data_wildcard_absolute_address_seq
14856     \l_tmpa_tl
14857     \seq_pop_right:NN
14858     \g_@@_jekyll_data_datatypes_seq
14859     \l_tmpa_tl
14860     \@@_jekyll_data_update_address_tls:
14861   }

```

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.

```

14862 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
14863   {
14864     \keys_set_known:nn
14865     { markdown/jekyllData }
14866     { { #1 } = { #2 } }
14867   }
14868 \cs_generate_variant:Nn
14869   \@@_jekyll_data_set_keyval_known:nn
14870   { Vn }
14871 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
14872   {

```

```

14873 \@@_jekyll_data_push:nN
14874   { #1 }
14875   \c_@@_jekyll_data_scalar_tl
14876 \@@_jekyll_data_set_keyval_known:Vn
14877   \g_@@_jekyll_data_wildcard_absolute_address_tl
14878   { #2 }
14879 \@@_jekyll_data_set_keyval_known:Vn
14880   \g_@@_jekyll_data_wildcard_relative_address_tl
14881   { #2 }
14882 \@@_jekyll_data_pop:
14883 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

14884 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
14885   \@@_jekyll_data_push:nN
14886   { #1 }
14887   \c_@@_jekyll_data_sequence_tl
14888 }
14889 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
14890   \@@_jekyll_data_push:nN
14891   { #1 }
14892   \c_@@_jekyll_data_mapping_tl
14893 }
14894 \def\markdownRendererJekyllDataSequenceEndPrototype{
14895   \@@_jekyll_data_pop:
14896 }
14897 \def\markdownRendererJekyllDataMappingEndPrototype{
14898   \@@_jekyll_data_pop:
14899 }
14900 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
14901   \@@_jekyll_data_set_keyvals_known:nn
14902   { #1 }
14903   { #2 }
14904 }
14905 \def\markdownRendererJekyllDataEmptyPrototype#1{}
14906 \def\markdownRendererJekyllDataNumberPrototype#1#2{
14907   \@@_jekyll_data_set_keyvals_known:nn
14908   { #1 }
14909   { #2 }
14910 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

14911 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
14912 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
14913   \@@_jekyll_data_set_keyvals_known:nn
14914   { #1 }

```

```

14915     { #2 }
14916 }
14917 \ExplSyntaxOff

```

### 3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key-values using the option `jeekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

14918 \ExplSyntaxOn
14919 \@@_with_various_cases:nn
14920 { jeekyllDataKeyValue }
14921 {
14922   \keys_define:nn
14923     { markdown/options }
14924     {
14925       #1 .code:n = {
14926         \@@_route_jeekyll_data_to_key_values:n
14927         { ##1 }
14928       },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

14929       #1 .default:n = { },
14930     }
14931 }
14932 \seq_new:N
14933 \l_@@_jeekyll_data_current_position_seq
14934 \tl_new:N
14935 \l_@@_jeekyll_data_current_position_tl
14936 \cs_new:Nn
14937 \@@_route_jeekyll_data_to_key_values:n
14938 {
14939   \markdownSetup
14940   {
14941     renderers = {
14942       jeekyllData(Sequence|Mapping)Begin = {
14943         \bool_lazy_and:nnTF
14944         {
14945           \seq_if_empty_p:N
14946           \l_@@_jeekyll_data_current_position_seq
14947         }
14948         {
14949           \str_if_eq_p:nn
14950           { ##1 }
14951           { null }
14952         }

```

```

14953     {
14954         \tl_if_empty:nF
14955         { #1 }
14956         {
14957             \seq_put_right:Nn
14958             \l_@@_jekyll_data_current_position_seq
14959             { #1 }
14960         }
14961     }
14962     {
14963         \seq_put_right:Nn
14964         \l_@@_jekyll_data_current_position_seq
14965         { ##1 }
14966     }
14967 },
14968 jekyllData(Sequence|Mapping)End = {
14969     \seq_pop_right:NN
14970     \l_@@_jekyll_data_current_position_seq
14971     \l_tmpa_tl
14972 },

```

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`.

```

14973     jekyllDataBoolean = {
14974         \tl_set:Nx
14975         \l_@@_jekyll_data_current_position_tl
14976         {
14977             \seq_use:Nn
14978             \l_@@_jekyll_data_current_position_seq
14979             { / }
14980         }
14981     \keys_if_exist:VnTF
14982     \l_@@_jekyll_data_current_position_tl
14983     { ##1 / boolean }
14984     {
14985         \@@_keys_set:xn
14986         {
14987             \tl_use:N
14988             \l_@@_jekyll_data_current_position_tl
14989             / ##1 / boolean
14990         }
14991         { ##2 }
14992     }
14993     {
14994         \@@_keys_set:xn

```



```

14995         {
14996             \tl_use:N
14997             \l_@@_jekyll_data_current_position_tl
14998             / ##1
14999         }
15000         { ##2 }
15001     }
15002 },
15003 jekyllDataNumber = {
15004     \tl_set:Nx
15005     \l_@@_jekyll_data_current_position_tl
15006     {
15007         \seq_use:Nn
15008         \l_@@_jekyll_data_current_position_seq
15009         { / }
15010     }
15011     \keys_if_exist:VnTF
15012     \l_@@_jekyll_data_current_position_tl
15013     { ##1 / number }
15014     {
15015         \@@_keys_set:xn
15016         {
15017             \tl_use:N
15018             \l_@@_jekyll_data_current_position_tl
15019             / ##1 / number
15020         }
15021         { ##2 }
15022     }
15023     {
15024         \@@_keys_set:xn
15025         {
15026             \tl_use:N
15027             \l_@@_jekyll_data_current_position_tl
15028             / ##1
15029         }
15030         { ##2 }
15031     }
15032 },

```

For the *<non-string type>* of `empty`, no value is passed to the key–value. Therefore, a default value should always be defined for nullable keys using the key property `.default:n`.

```

15033     jekyllDataEmpty = {
15034         \tl_set:Nx
15035         \l_@@_jekyll_data_current_position_tl
15036         {
15037             \seq_use:Nn

```

```

15038         \l_@@_jekyll_data_current_position_seq
15039         { / }
15040     }
15041     \keys_if_exist:VnTF
15042     \l_@@_jekyll_data_current_position_tl
15043     { ##1 / empty }
15044     {
15045         \keys_set:xn
15046         {
15047             \tl_use:N
15048             \l_@@_jekyll_data_current_position_tl
15049             / ##1
15050         }
15051         { empty }
15052     }
15053     {
15054         \keys_set:Vn
15055         \l_@@_jekyll_data_current_position_tl
15056         { ##1 }
15057     }
15058     },

```

For every YAML key `path.to.<key>` with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key–value `<module>/path/to/<key>` if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key `<key>` of the key–value `<module>/path/to` with the typographic string of the value unless the key `typographicString` is known. If the key `programmaticString` is known, only set the key `<key>` if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set `<key>` normally, i.e. regardless of whether it is known or unknown.

```

15059     jekyllDataTypographicString = {
15060         \tl_set:Nx
15061         \l_@@_jekyll_data_current_position_tl
15062         {
15063             \seq_use:Nn
15064             \l_@@_jekyll_data_current_position_seq
15065             { / }
15066         }
15067         \keys_if_exist:VnTF
15068         \l_@@_jekyll_data_current_position_tl
15069         { ##1 / typographicString }
15070         {
15071             \@@_keys_set:xn
15072             {
15073                 \tl_use:N

```

```

15074         \l_@@_jekyll_data_current_position_tl
15075         / ##1 / typographicString
15076     }
15077     { ##2 }
15078 }
15079 {
15080     \keys_if_exist:VnTF
15081     \l_@@_jekyll_data_current_position_tl
15082     { ##1 / programmaticString }
15083     {
15084         \@@_keys_set_known:xn
15085         {
15086             \tl_use:N
15087             \l_@@_jekyll_data_current_position_tl
15088             / ##1
15089         }
15090         { ##2 }
15091     }
15092     {
15093         \@@_keys_set:xn
15094         {
15095             \tl_use:N
15096             \l_@@_jekyll_data_current_position_tl
15097             / ##1
15098         }
15099         { ##2 }
15100     }
15101 }
15102 },
15103 jekyllDataProgrammaticString = {
15104     \tl_set:Nx
15105     \l_@@_jekyll_data_current_position_tl
15106     {
15107         \seq_use:Nn
15108         \l_@@_jekyll_data_current_position_seq
15109         { / }
15110     }
15111     \keys_if_exist:VnT
15112     \l_@@_jekyll_data_current_position_tl
15113     { ##1 / programmaticString }
15114     {
15115         \@@_keys_set:xn
15116         {
15117             \tl_use:N
15118             \l_@@_jekyll_data_current_position_tl
15119             / ##1 / programmaticString
15120         }

```

```

15121             { ##2 }
15122             }
15123         },
15124     },
15125 }
15126 }
15127 \cs_new:Nn
15128   \@@_keys_set:nn
15129   {
15130     \keys_set:nn
15131       { }
15132       { { #1 } = { #2 } }
15133   }
15134 \cs_new:Nn
15135   \@@_keys_set_known:nn
15136   {
15137     \keys_set_known:nn
15138       { }
15139       { { #1 } = { #2 } }
15140   }
15141 \cs_generate_variant:Nn
15142   \@@_keys_set:nn
15143   { xn }
15144 \cs_generate_variant:Nn
15145   \@@_keys_set_known:nn
15146   { xn }
15147 \cs_generate_variant:Nn
15148   \keys_set:nn
15149   { xn, Vn }
15150 \prg_generate_conditional_variant:Nnn
15151   \keys_if_exist:nn
15152   { Vn }
15153   { T, TF }
15154 \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).

```

15155 \ExplSyntaxOn
15156 \str_if_eq:VVT
15157   \c_@@_top_layer_tl
15158   \c_@@_option_layer_plain_tex_tl
15159   {
15160     \use:c
15161       { ExplSyntaxOff }
15162     \@@_if_option:nF
15163       { noDefaults }

```

```

15164     {
15165         \@@_if_option:nTF
15166         { experimental }
15167         {
15168             \@@_setup:n
15169             { theme = witiko/markdown/defaults@experimental }
15170         }
15171         {
15172             \@@_setup:n
15173             { theme = witiko/markdown/defaults }
15174         }
15175     }
15176     \use:c
15177     { ExplSyntaxOn }
15178 }
15179 \ExplSyntaxOff

```

### 3.2.4 Lua Snippets

After the `\markdownPrepareLuaOptions` macro has been fully expanded, the `\markdownLuaOptions` macro will expand to a Lua table that contains the plain TeX options (see Section 2.2.2) in a format recognized by Lua (see Section 2.1.3).

```

15180 \ExplSyntaxOn
15181 \tl_new:N \g_@@_formatted_lua_options_tl
15182 \cs_new:Nn \@@_format_lua_options:
15183 {
15184     \tl_gclear:N
15185     \g_@@_formatted_lua_options_tl
15186     \seq_map_function:NN
15187     \g_@@_lua_options_seq
15188     \@@_format_lua_option:n
15189 }
15190 \cs_new:Nn \@@_format_lua_option:n
15191 {
15192     \@@_typecheck_option:n
15193     { #1 }
15194     \@@_get_option_type:nN
15195     { #1 }
15196     \l_tmpa_tl
15197     \bool_case_true:nF
15198     {
15199         {
15200             \str_if_eq_p:VV
15201             \l_tmpa_tl
15202             \c_@@_option_type_boolean_tl ||
15203             \str_if_eq_p:VV

```

```

15204         \l_tmpa_tl
15205         \c_@@_option_type_number_tl ||
15206     \str_if_eq_p:VV
15207         \l_tmpa_tl
15208         \c_@@_option_type_counter_tl
15209     }
15210     {
15211         \@@_get_option_value:nN
15212         { #1 }
15213         \l_tmpa_tl
15214         \tl_gput_right:Nx
15215         \g_@@_formatted_lua_options_tl
15216         { #1~::~ \l_tmpa_tl ,~ }
15217     }
15218     {
15219         \str_if_eq_p:VV
15220         \l_tmpa_tl
15221         \c_@@_option_type_clist_tl
15222     }
15223     {
15224         \@@_get_option_value:nN
15225         { #1 }
15226         \l_tmpa_tl
15227         \tl_gput_right:Nx
15228         \g_@@_formatted_lua_options_tl
15229         { #1~::~\c_left_brace_str }
15230         \clist_map_inline:Vn
15231         \l_tmpa_tl
15232         {
15233             \@@_lua_escape:xN
15234             { ##1 }
15235             \l_tmpb_tl
15236             \tl_gput_right:Nn
15237             \g_@@_formatted_lua_options_tl
15238             { " }
15239             \tl_gput_right:NV
15240             \g_@@_formatted_lua_options_tl
15241             \l_tmpb_tl
15242             \tl_gput_right:Nn
15243             \g_@@_formatted_lua_options_tl
15244             { " ,~ }
15245         }
15246         \tl_gput_right:Nx
15247         \g_@@_formatted_lua_options_tl
15248         { \c_right_brace_str ,~ }
15249     }
15250 }

```

```

15251     {
15252     \@@_get_option_value:nN
15253     { #1 }
15254     \l_tmpa_tl
15255     \@@_lua_escape:xN
15256     { \l_tmpa_tl }
15257     \l_tmpb_tl
15258     \tl_gput_right:Nn
15259     \g_@@_formatted_lua_options_tl
15260     { #1~::~ " }
15261     \tl_gput_right:NV
15262     \g_@@_formatted_lua_options_tl
15263     \l_tmpb_tl
15264     \tl_gput_right:Nn
15265     \g_@@_formatted_lua_options_tl
15266     { " ,~ }
15267     }
15268   }
15269 \cs_generate_variant:Nn
15270 \clist_map_inline:nn
15271 { Vn }
15272 \let
15273 \markdownPrepareLuaOptions
15274 \@@_format_lua_options:
15275 \def
15276 \markdownLuaOptions
15277 {
15278   {
15279     \g_@@_formatted_lua_options_tl
15280   }
15281 }
15282 \sys_if_engine luatex:TF
15283 {
15284   \cs_new:Nn
15285   \@@_lua_escape:nN
15286   {
15287     \tl_set:Nx
15288     #2
15289     {
15290       \lua_escape:n
15291       { #1 }
15292     }
15293   }
15294 }
15295 {
15296   \regex_const:Nn
15297   \c_@@_lua_escape_regex

```

```

15298     { [\\'' ] }
15299   \cs_new:Nn
15300     \@@_lua_escape:nN
15301     {
15302       \tl_set:Nn
15303         #2
15304         { #1 }
15305       \regex_replace_all:NnN
15306         \c_@@_lua_escape_regex
15307         { \u { c_backslash_str } \0 }
15308         #2
15309     }
15310 }
15311 \cs_generate_variant:Nn
15312   \@@_lua_escape:nN
15313   { 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.

```

15314 \tl_new:N
15315   \markdownInputFilename
15316 \cs_new:Npn
15317   \markdownPrepareInputFilename
15318   #1
15319   {
15320     \@@_lua_escape:xN
15321     { #1 }
15322     \markdownInputFilename
15323     \tl_gset:Nx
15324     \markdownInputFilename
15325     { " \markdownInputFilename " }
15326   }

```

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.

```

15327 \cs_new:Npn
15328   \markdownPrepare
15329   {

```

First, ensure that the `cacheDir` directory exists.

```

15330   local~lfs = require("lfs")
15331   local~options = \markdownLuaOptions
15332   if~not~lfs.isdir(options.cacheDir) then~
15333     assert(lfs.mkdir(options.cacheDir))
15334   end~

```



Next, load the `markdown` module and create a converter function using the plain  $\TeX$  options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
15335     local-md = require("markdown")
15336     local-convert = md.new(options)
15337 }
```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain  $\TeX$ . It opens the input file, converts it, and prints the conversion result.

```
15338 \cs_new:Npn
15339   \markdownConvert
15340   {
15341     local~filename = \markdownInputFilename
15342     local~file = assert(io.open(filename, "r"),
15343       [[Could~not~open~file~]] .. filename .. [[~for~reading]])
15344     local~input = assert(file:read("*a"))
15345     assert(file:close())
15346     print(convert(input))
15347   }
```

```
15348 \ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain  $\TeX$ .

```
15349 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
15350   if options.cacheDir then
15351     lfs.rmdir(options.cacheDir)
15352   end
15353 }%
```

### 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.

```
15354 \csname newread\endcsname\markdownInputFileStream
15355 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
15356 \begingroup
15357   \catcode\^^I=12%
15358   \gdef\markdownReadAndConvertTab{^^I}%
15359 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\LaTeX 2\epsilon$  `\filecontents` macro to plain  $\TeX$ .

```
15360 \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.

```

15361 \catcode\^^M=13%
15362 \catcode\^^I=13%
15363 \catcode|=0%
15364 \catcode\\=12%
15365 |catcode@=14%
15366 |catcode|=12@
15367 |gdef|markdownReadAndConvert#1#2{@
15368   |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

15369   |markdownIfOption{frozenCache}{-}{@
15370     |immediate|openout|markdownOutputFileStream@
15371     |markdownOptionInputTempFileName|relax@
15372     |markdownInfo{@
15373       Buffering block-level markdown input into the temporary @
15374       input file "|markdownOptionInputTempFileName" and scanning @
15375       for the closing token sequence "#1"}@
15376   }@

```

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.

```

15377   |def|do##1{|catcode`##1=12}|dospecials@
15378   |catcode`|=12@
15379   |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.

```

15380   |def|markdownReadAndConvertStripPercentSign##1{@
15381     |markdownIfOption{stripPercentSigns}{-}{@
15382       |if##1%@
15383         |expandafter|expandafter|expandafter@
15384         |markdownReadAndConvertProcessLine@
15385       |else@
15386         |expandafter|expandafter|expandafter@
15387         |markdownReadAndConvertProcessLine@
15388         |expandafter|expandafter|expandafter##1@
15389       |fi@
15390     }{@

```

```

15391         |expandafter@
15392         |markdownReadAndConvertProcessLine@
15393         |expandafter##1@
15394     }@
15395 }@

```

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.

```

15396     |def|markdownReadAndConvertProcessLine##1##2##1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

15397         |ifx|relax##3|relax@
15398         |markdownIfOption{frozenCache}{-}{@
15399         |immediate|write|markdownOutputFileStream{##1}@
15400     }@
15401     |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.

```

15402     |def^^M{@
15403         |markdownInfo{The ending token sequence was found}@
15404         |markdownIfOption{frozenCache}{-}{@
15405         |immediate|closeout|markdownOutputFileStream@
15406     }@
15407     |endgroup@
15408     |markdownInput{@
15409         |markdownOptionOutputDir@
15410         /|markdownOptionInputTempFileName@
15411     }@
15412     #2}@
15413 |fi@

```

Repeat with the next line.

```

15414     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

15415     |catcode`|^I=13@
15416     |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.

```

15417 |catcode`|^M=13@
15418 |def^M##1^M{@
15419 |def^M###1^M{@
15420 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
15421 ^M}@
15422 ^M}@

```

Reset the character categories back to the former state.

```
15423 |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.

```

15424 \ExplSyntaxOn
15425 \cs_new:Npn
15426 \markdownLuaExecute
15427 #1
15428 {
15429 \int_compare:nNt
15430 { \g_luabridge_method_int }
15431 =
15432 { \c_luabridge_method_shell_int }
15433 {
15434 \sys_if_shell_unrestricted:F
15435 {
15436 \sys_if_shell:TF
15437 {
15438 \msg_error:nn
15439 { markdown }
15440 { restricted-shell-access }
15441 }
15442 {
15443 \msg_error:nn
15444 { markdown }
15445 { disabled-shell-access }
15446 }
15447 }
15448 }
15449 \str_gset:NV
15450 \g_luabridge_output_dirname_str
15451 \markdownOptionOutputDir
15452 \luabridge_now:e
15453 { #1 }
15454 }
15455 \cs_generate_variant:Nn
15456 \msg_new:nnnn
15457 { nnnV }
15458 \tl_set:Nn
15459 \l_tmpa_tl

```

```

15460 {
15461     You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
15462     --enable-write18~flag,~or~write~shell_escape=t~in~the~
15463     texmf.cnf~file.
15464 }
15465 \msg_new:nnnV
15466 { markdown }
15467 { restricted-shell-access }
15468 { Shell-escape-is-restricted }
15469 \l_tmpa_tl
15470 \msg_new:nnnV
15471 { markdown }
15472 { disabled-shell-access }
15473 { Shell-escape-is-disabled }
15474 \l_tmpa_tl
15475 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

15476 \ExplSyntaxOn
15477 \tl_new:N
15478   \g_@@_after_markinline_tl
15479 \tl_gset:Nn
15480   \g_@@_after_markinline_tl
15481   { \unskip }
15482 \cs_new:Npn
15483   \markinline
15484   {

```

Locally change the category of the special plain  $\TeX$  characters to *other* in order to prevent unwanted interpretation of the input markdown text as  $\TeX$  code.

```

15485     \group_begin:
15486     \cctab_select:N
15487     \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

15488     \@@_if_option:nF
15489     { frozenCache }
15490     {
15491         \immediate
15492         \openout
15493         \markdownOutputFileStream
15494         \markdownOptionInputTempFileName
15495         \relax
15496         \msg_info:nne

```

```

15497         { markdown }
15498         { buffering-markinline }
15499         { \markdownOptionInputTempFileName }
15500     }

```

Peek ahead and extract the inline markdown text.

```

15501     \peek_regex_replace_once:nnF
15502     { { (.*) } }
15503     {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

15504         \c { @@_if_option:nF }
15505         \cB { frozenCache \cE }
15506         \cB {
15507             \c { immediate }
15508             \c { write }
15509             \c { markdownOutputFileStream }
15510             \cB { \1 \cE }
15511             \c { immediate }
15512             \c { closeout }
15513             \c { markdownOutputFileStream }
15514         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

15515         \c { group_end: }
15516         \c { group_begin: }
15517         \c { @@_setup:n }
15518         \cB { contentLevel = inline \cE }
15519         \c { markdownInput }
15520         \cB {
15521             \c { markdownOptionOutputDir } /
15522             \c { markdownOptionInputTempFileName }
15523         \cE }
15524         \c { group_end: }
15525         \c { tl_use:N }
15526         \c { g_@@_after_markinline_tl }
15527     }
15528     {
15529         \msg_error:nn
15530         { markdown }
15531         { markinline-peek-failure }
15532         \group_end:
15533         \tl_use:N
15534         \g_@@_after_markinline_tl
15535     }
15536 }
15537 \msg_new:nnn

```

```

15538 { markdown }
15539 { buffering-markinline }
15540 { Buffering~inline~markdown~input~into~
15541   the~temporary~input~file~"#1". }
15542 \msg_new:nnnn
15543 { markdown }
15544 { markinline-peek-failure }
15545 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
15546 { The~macro~should~be~followed~by~inline~
15547   markdown~text~in~curly~braces }
15548 \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.

```

15549 \ExplSyntaxOn
15550 \cs_new:Npn
15551   \markdownInput
15552   #1
15553   {
15554     \@@_if_option:nTF
15555       { frozenCache }
15556       {
15557         \markdownInputRaw
15558         { #1 }
15559       }
15560     {

```

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`.

```

15561       \tl_set:Nx
15562         \l_tmpa_tl
15563         { #1 }
15564       \file_get_full_name:VNTF
15565         \l_tmpa_tl
15566         \l_tmpb_tl
15567       {
15568         \exp_args:NV
15569           \markdownInputRaw
15570           \l_tmpb_tl
15571       }
15572     {
15573       \msg_error:nnV
15574         { markdown }

```

```

15575             { markdown-file-does-not-exist }
15576             \l_tmpa_tl
15577         }
15578     }
15579 }
15580 \msg_new:nnn
15581 { markdown }
15582 { markdown-file-does-not-exist }
15583 {
15584     Markdown~file~#1~does-not~exist
15585 }
15586 \ExplSyntaxOff
15587 \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.

```

15588 \catcode`\=0%
15589 \catcode`\|=12%
15590 \catcode`\&=6%
15591 |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.

```

15592 |begingroup
15593 |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.

```

15594 |catcode`|#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

15595 |markdownIfOption{frozenCache}{%
15596 |ifnum|markdownOptionFrozenCacheCounter=0|relax
15597 |markdownInfo{Reading frozen cache from
15598 |"|markdownOptionFrozenCacheFileName"|}%
15599 |input|markdownOptionFrozenCacheFileName|relax
15600 |fi
15601 |markdownInfo{Including markdown document number
15602 |"|the|markdownOptionFrozenCacheCounter" from frozen cache}%
15603 |csname markdownFrozenCache%
15604 |the|markdownOptionFrozenCacheCounter|endcsname
15605 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
15606 }{%
15607 |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  $\LaTeX$ Mk to track changes to the markdown document.

```

15608     |openin|markdownInputFileStream{&1}%
15609     |closein|markdownInputFileStream
15610     |markdownPrepareLuaOptions
15611     |markdownPrepareInputFilename{&1}%
15612     |markdownLuaExecute{%
15613         |markdownPrepare
15614         |markdownConvert
15615         |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

15616     |markdownIfOption{finalizeCache}{%
15617         |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
15618     }%
15619     |endgroup
15620 }%
15621 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of  $\TeX$  to execute a  $\TeX$  document in the middle of a markdown document fragment.

```

15622 \gdef\markdownEscape#1{%
15623     \catcode`\%=14\relax
15624     \catcode`\#=6\relax
15625     \input #1\relax
15626     \catcode`\%=12\relax
15627     \catcode`\#=12\relax
15628 }%

```

### 3.3 $\LaTeX$ Implementation

The  $\LaTeX$  implementation makes use of the fact that, apart from some subtle differences,  $\LaTeX$  implements the majority of the plain  $\TeX$  format [18, Section 9]. As a consequence, we can directly reuse the existing plain  $\TeX$  implementation.

```

15629 \def\markdownVersionSpace{ }%
15630 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
15631     \markdownVersion\markdownVersionSpace markdown renderer]%

```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to

accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

15632 \ExplSyntaxOn
15633 \cs_gset_eq:NN
15634   \markinlinePlainTeX
15635   \markinline
15636 \cs_gset:Npn
15637   \markinline
15638   {
15639     \peek_regex_replace_once:nn
15640     { ( \[ (.*) \] ) ? }
15641     {

```

Apply the options locally.

```

15642       \c { group_begin: }
15643       \c { @@_setup:n }
15644       \cB { \2 \cE }
15645       \c { tl_put_right:Nn }
15646       \c { g_@@_after_markinline_tl }
15647       \cB { \c { group_end: } \cE }
15648       \c { markinlinePlainTeX }
15649     }
15650   }
15651 \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).

```

15652 \let\markdownInputPlainTeX\markdownInput
15653 \renewcommand\markdownInput[2][]{%
15654   \begingroup
15655     \markdownSetup{#1}%
15656     \markdownInputPlainTeX{#2}%
15657   \endgroup}%
15658 \renewcommand\yamlInput[2][]{%
15659   \begingroup
15660     \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
15661     \markdownInputPlainTeX{#2}%
15662   \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

15663 \ExplSyntaxOn
15664 \renewenvironment
15665   { markdown }
15666   {

```

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:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in 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.

```
15667 \group_begin:
15668 \char_set_catcode_active:n { 13 }
```

To prevent doubling the hash signs (`#`, ASCII code 35 in decimal), we switch its category from 6 (parameter) to 12 (letter).

```
15669 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
15670 \peek_regex_replace_once:nF
15671 { \ *[\r*([~]*)\][^~\r]* }
15672 {
```

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.

```
15673 \c { group_end: }
15674 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
15675 \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}
```

```
15676     \c { exp_args:NV }
15677     \c { markdownReadAndConvert@ }
15678     \c { @currenvir }
15679   }
15680   {
15681     \group_end:
15682     \exp_args:NV
15683     \markdownReadAndConvert@
15684     \@currenvir
15685   }
15686 }
15687 { \markdownEnd }
15688 \renewenvironment
15689 { markdown* }
15690 [ 1 ]
15691 {
15692   \@@_if_option:nTF
15693   { experimental }
15694   {
15695     \msg_error:nnn
15696     { markdown }
15697     { latex-markdown-star-deprecated }
15698     { #1 }
15699   }
15700   {
15701     \msg_warning:nnn
15702     { markdown }
15703     { latex-markdown-star-deprecated }
15704     { #1 }
15705   }
15706   \@@_setup:n
15707   { #1 }
15708   \markdownReadAndConvert@
15709   { markdown* }
15710 }
15711 { \markdownEnd }
15712 \renewenvironment
```

```

15713 { yaml }
15714 {
15715   \group_begin:
15716   \yamlSetup
15717     { jekyllData, expectJekyllData, ensureJekyllData }
15718   \markdown
15719 }
15720 { \yamlEnd }
15721 \msg_new:nnn
15722 { markdown }
15723 { latex-markdown-star-deprecated }
15724 {
15725   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
15726   be~removed~in~the~next~major~version~of~the~Markdown~package.
15727 }
15728 \cs_generate_variant:Nn % noqa: e405, w402
15729 \@@_setup:n
15730 { V }
15731 \ExplSyntaxOff
15732 \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*.

```

15733 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
15734 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
15735 |gdef|markdownReadAndConvert@#1<%
15736   |markdownReadAndConvert<\end{#1}>%
15737   <|end<#1>>>%
15738 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

15739 \ExplSyntaxOn
15740 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
15741 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
15742 \cs_gset:Nn % noqa: w401
15743 \@@_load_theme:nnn
15744 {

```

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.

```
15745     \ifmarkdownLaTeXLoaded
15746     \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.

```
15747     \bool_if:nTF
15748     {
15749     \bool_lazy_or_p:nn
15750     {
15751     \prop_if_in_p:Nn
15752     \g_@@_latex_built_in_themes_prop
15753     { #1 }
15754     }
15755     {
15756     \file_if_exist_p:n
15757     { markdown theme #3.sty }
15758     }
15759     }
15760     {
15761     \msg_error:nnn
15762     { markdown }
15763     { latex-theme-after-preamble }
15764     { #1 }
15765     }
```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```
15766     {
15767     \@@_plain_tex_load_theme:nnn
15768     { #1 }
15769     { #2 }
15770     { #3 }
15771     }
15772     \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.

```
15773     \bool_if:nTF
15774     {
15775     \bool_lazy_or_p:nn
15776     {
15777     \prop_if_in_p:Nn
15778     \g_@@_latex_built_in_themes_prop
15779     { #1 }
15780     }
15781     {
15782     \file_if_exist_p:n
```

```

15783         { markdown theme #3.sty }
15784     }
15785 }
15786 {
15787     \prop_get:NnNTF
15788     \g_@@_latex_loaded_themes_linenos_prop
15789     { #1 }
15790     \l_tmpa_tl
15791     {
15792         \prop_get:NnN
15793         \g_@@_latex_loaded_themes_versions_prop
15794         { #1 }
15795         \l_tmpb_tl
15796         \str_if_eq:nVTF
15797         { #2 }
15798         \l_tmpb_tl
15799         {
15800             \msg_warning:nnnVn
15801             { markdown }
15802             { repeatedly-loaded-latex-theme }
15803             { #1 }
15804             \l_tmpa_tl
15805             { #2 }
15806         }
15807         {
15808             \msg_error:nnnnVV
15809             { markdown }
15810             { different-versions-of-latex-theme }
15811             { #1 }
15812             { #2 }
15813             \l_tmpb_tl
15814             \l_tmpa_tl
15815         }
15816     }
15817 }
15818     \prop_gput:Nnx
15819     \g_@@_latex_loaded_themes_linenos_prop
15820     { #1 }
15821     { \tex_the:D \tex_inputlineno:D } % noqa: W200
15822     \prop_gput:Nnn
15823     \g_@@_latex_loaded_themes_versions_prop
15824     { #1 }
15825     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

15826     \prop_if_in:NnTF

```

```

15827         \g_@@_latex_built_in_themes_prop
15828         { #1 }
15829         {
15830             \msg_info:nnnn
15831             { markdown }
15832             { loading-built-in-latex-theme }
15833             { #1 }
15834             { #2 }
15835             \prop_item:Nn
15836             \g_@@_latex_built_in_themes_prop
15837             { #1 }
15838         }
15839         {
15840             \msg_info:nnnn
15841             { markdown }
15842             { loading-latex-theme }
15843             { #1 }
15844             { #2 }
15845             \RequirePackage
15846             { markdown theme #3 }
15847         }
15848     }
15849 }
15850 {
15851     \@@_plain_tex_load_theme:nnn
15852     { #1 }
15853     { #2 }
15854     { #3 }
15855 }
15856 \fi
15857 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

15858     \msg_info:nnnn
15859     { markdown }
15860     { theme-loading-postponed }
15861     { #1 }
15862     { #2 }
15863     \AtEndOfPackage
15864     {
15865         \@@_set_theme:n
15866         { #1 @ #2 }
15867     }
15868 \fi
15869 }
15870 \msg_new:nnn

```



```

15871 { markdown }
15872 { theme-loading-postponed }
15873 {
15874     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
15875     Markdown~package~has~finished~loading
15876 }
15877 \msg_new:nnn
15878 { markdown }
15879 { loading-built-in-latex-theme }
15880 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
15881 \msg_new:nnn
15882 { markdown }
15883 { loading-latex-theme }
15884 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
15885 \msg_new:nnn
15886 { markdown }
15887 { repeatedly-loaded-latex-theme }
15888 {
15889     Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
15890     loaded~on~line~#2,~not~loading~it~again
15891 }
15892 \msg_new:nnn
15893 { markdown }
15894 { different-versions-of-latex-theme }
15895 {
15896     Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
15897     but~version~#3~has~already~been~loaded~on~line~#4
15898 }
15899 \cs_generate_variant:Nn
15900 \msg_new:nnnn
15901 { nnVV }
15902 \tl_set:Nn
15903 \l_tmpa_tl
15904 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
15905 \tl_put_right:NV
15906 \l_tmpa_tl
15907 \c_backslash_str
15908 \tl_put_right:Nn
15909 \l_tmpa_tl
15910 { begin { document } }
15911 \tl_set:Nn
15912 \l_tmpb_tl
15913 { Load~Markdown~theme~#1~before~ }
15914 \tl_put_right:NV
15915 \l_tmpb_tl
15916 \c_backslash_str
15917 \tl_put_right:Nn

```

```

15918 \l_tmpb_tl
15919 { begin { document } }
15920 \msg_new:nnVV
15921 { markdown }
15922 { latex-theme-after-preamble }
15923 \l_tmpa_tl
15924 \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.

```

15925 \tl_set:Nn
15926 \l_tmpa_tl
15927 {
15928   \RequirePackage
15929     { graphicx }
15930   \markdownLoadPlainTeXTheme
15931 }
15932 \prop_gput:NnV
15933 \g_@@_latex_built_in_themes_prop
15934 { witiko / dot }
15935 \l_tmpa_tl
15936 \prop_gput:NnV
15937 \g_@@_latex_built_in_themes_prop
15938 { witiko / graphicx / http }
15939 \l_tmpa_tl
15940 \ExplSyntaxOff

```

The `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>X theme also loads the corresponding plain T<sub>E</sub>X theme.

```
15941 \markdownLoadPlainTeXTheme
```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

15942 \DeclareOption*{%
15943   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
15944 \ProcessOptions\relax

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
15945 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

### 3.3.4.1 Lists

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package `paralist` or the package `enumitem` have already been loaded, use them. Otherwise, if the option `experimental` or the command `\DocumentMetadata` have been used, use the package `enumitem`. Otherwise, use the package `paralist`.

```
15946 \ExplSyntaxOn
15947 \bool_new:N
15948   \g_@@_tight_or_fancy_lists_bool
15949 \bool_gset_false:N
15950   \g_@@_tight_or_fancy_lists_bool
15951 \@@_if_option:nTF
15952   { tightLists }
15953   {
15954     \bool_gset_true:N
15955       \g_@@_tight_or_fancy_lists_bool
15956   }
15957   {
15958     \@@_if_option:nT
15959     { fancyLists }
15960     {
15961       \bool_gset_true:N
15962         \g_@@_tight_or_fancy_lists_bool
15963     }
15964   }
15965 \bool_new:N
15966   \g_@@_beamer_paralist_or_enumitem_bool
15967 \bool_gset_true:N
15968   \g_@@_beamer_paralist_or_enumitem_bool
15969 \@ifclassloaded
15970   { beamer }
15971   { }
15972   {
15973     \@ifpackageloaded
15974       { paralist }
15975       { }
15976     {
15977       \@ifpackageloaded
15978         { enumitem }
15979         { }
15980       {
15981         \bool_gset_false:N
15982           \g_@@_beamer_paralist_or_enumitem_bool
15983       }
15984     }
```

```

15985 }
15986 \prg_generate_conditional_variant:Nnn
15987 \str_if_eq:nn
15988 { en }
15989 { TF }
15990 \bool_if:nT
15991 {
15992   \g_@@_tight_or_fancy_lists_bool &&
15993   ! \g_@@_beamer_paralist_or_enumitem_bool
15994 }
15995 {
15996   \str_if_eq:enTF
15997   { \markdownThemeVersion }
15998   { experimental }
15999   {
16000     \RequirePackage
16001     { enumitem }
16002   }
16003   {
16004     \IfDocumentMetadataTF
16005     {
16006       \RequirePackage
16007       { enumitem }
16008     }
16009     {
16010       \RequirePackage
16011       { paralist }
16012     }
16013   }
16014 }
16015 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16016 \ExplSyntaxOn
16017 \cs_new:Nn
16018 \@@_latex_fancy_list_item_label_number:nn
16019 {
16020   \str_case:nn
16021   { #1 }
16022   {
16023     { Decimal } { #2 }
16024     { LowerRoman } { \int_to_roman:n { #2 } }
16025     { UpperRoman } { \int_to_Roman:n { #2 } }
16026     { LowerAlpha } { \int_to_alph:n { #2 } }
16027     { UpperAlpha } { \int_to_Alph:n { #2 } }
16028   }

```

```

16029 }
16030 \cs_new:Nn
16031 \@@_latex_fancy_list_item_label_delimiter:n
16032 {
16033   \str_case:nn
16034     { #1 }
16035     {
16036       { Default } { . }
16037       { OneParen } { ) }
16038       { Period } { . }
16039     }
16040 }
16041 \cs_new:Nn
16042 \@@_latex_fancy_list_item_label:nnn
16043 {
16044   \@@_latex_fancy_list_item_label_number:nn
16045     { #1 }
16046     { #3 }
16047   \@@_latex_fancy_list_item_label_delimiter:n
16048     { #2 }
16049 }
16050 \cs_generate_variant:Nn
16051 \@@_latex_fancy_list_item_label:nnn
16052 { VVn }
16053 \tl_new:N
16054 \l_@@_latex_fancy_list_item_label_number_style_tl
16055 \tl_new:N
16056 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16057 \ifpackageloaded { enumitem } {
16058   \markdownSetup { rendererPrototypes = {
First, let's define the tight list item renderer prototypes.
16059     ulBeginTight = {
16060       \begin
16061         { itemize }
16062         [ noitemsep ]
16063       },
16064     ulEndTight = {
16065       \end
16066         { itemize }
16067       },
16068     olBeginTight = {
16069       \begin
16070         { enumerate }
16071         [ noitemsep ]
16072       },
16073     olEndTight = {
16074       \end

```

```

16075     { enumerate }
16076 },
16077 dlBeginTight = {
16078   \begin
16079     { description }
16080     [ noitemsep ]
16081 },
16082 dlEndTight = {
16083   \end
16084     { description }
16085 },

```

Second, let's define the fancy list item renderer prototypes.

```

16086 fancyOlBegin = {
16087   \group_begin:
16088   \tl_set:Nn
16089     \l_@@_latex_fancy_list_item_label_number_style_tl
16090     { #1 }
16091   \tl_set:Nn
16092     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16093     { #2 }
16094   \begin
16095     { enumerate }
16096 },
16097 fancyOlBeginTight = {
16098   \group_begin:
16099   \tl_set:Nn
16100     \l_@@_latex_fancy_list_item_label_number_style_tl
16101     { #1 }
16102   \tl_set:Nn
16103     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16104     { #2 }
16105   \begin
16106     { enumerate }
16107     [ noitemsep ]
16108 },
16109 fancyOlEnd(|Tight) = {
16110   \end { enumerate }
16111   \group_end:
16112 },
16113 fancyOlItemWithNumber = {
16114   \item
16115   [
16116     \@@_latex_fancy_list_item_label:VVn
16117     \l_@@_latex_fancy_list_item_label_number_style_tl
16118     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16119     { #1 }
16120   ]

```

```

16121     },
16122   } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

16123 }
16124 { \@ifpackageloaded { paralist } {
16125   \markdownSetup { rendererPrototypes = {

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

16126     ulBeginTight = {
16127       \group_begin:
16128       \pltopsep=\topsep
16129       \plpartopsep=\partopsep
16130       \begin { compactitem }
16131     },
16132     ulEndTight = {
16133       \end { compactitem }
16134       \group_end:
16135     },
16136     fancyOlBegin = {
16137       \group_begin:
16138       \tl_set:Nn
16139         \l_@@_latex_fancy_list_item_label_number_style_tl
16140         { #1 }
16141       \tl_set:Nn
16142         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16143         { #2 }
16144       \begin { enumerate }
16145     },
16146     fancyOlEnd = {
16147       \end { enumerate }
16148       \group_end:
16149     },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

16150     olBeginTight = {
16151       \group_begin:
16152       \plpartopsep=\partopsep
16153       \pltopsep=\topsep
16154       \begin { compactenum }
16155     },
16156     olEndTight = {
16157       \end { compactenum }
16158       \group_end:
16159     },

```

```

16160 fancyO1BeginTight = {
16161   \group_begin:
16162   \tl_set:Nn
16163     \l_@@_latex_fancy_list_item_label_number_style_tl
16164     { #1 }
16165   \tl_set:Nn
16166     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16167     { #2 }
16168   \plpartopsep=\partopsep
16169   \pltopsep=\topsep
16170   \begin { compactenum }
16171 },
16172 fancyO1EndTight = {
16173   \end { compactenum }
16174   \group_end:
16175 },
16176 fancyO1ItemWithNumber = {
16177   \item
16178   [
16179     \@@_latex_fancy_list_item_label:VVn
16180     \l_@@_latex_fancy_list_item_label_number_style_tl
16181     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
16182     { #1 }
16183   ]
16184 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

16185 dlBeginTight = {
16186   \group_begin:
16187   \plpartopsep=\partopsep
16188   \pltopsep=\topsep
16189   \begin { compactdesc }
16190 },
16191 dlEndTight = {
16192   \end { compactdesc }
16193   \group_end:
16194 }
16195 } }
16196 }
16197 {

```

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.

```

16198 \markdownSetup
16199 {
16200   rendererPrototypes = {

```



```

16201         ulBeginTight = \markdownRendererUlBegin,
16202         ulEndTight = \markdownRendererUlEnd,
16203         fancyOlBegin = \markdownRendererOlBegin,
16204         fancyOlEnd = \markdownRendererOlEnd,
16205         olBeginTight = \markdownRendererOlBegin,
16206         olEndTight = \markdownRendererOlEnd,
16207         fancyOlBeginTight = \markdownRendererOlBegin,
16208         fancyOlEndTight = \markdownRendererOlEnd,
16209         dlBeginTight = \markdownRendererDlBegin,
16210         dlEndTight = \markdownRendererDlEnd,
16211     },
16212 }
16213 } }
16214 \ExplSyntaxOff
16215 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

16216 \@ifpackageloaded{unicode-math}{
16217     \markdownSetup{rendererPrototypes={
16218         untickedBox = {\mdlgwhtsquare$},
16219     }}
16220 }{
16221     \RequirePackage{amssymb}
16222     \markdownSetup{rendererPrototypes={
16223         untickedBox = {\square$},
16224     }}
16225 }
16226 \RequirePackage{csvsimple}
16227 \RequirePackage{fancyvrb}
16228 \RequirePackage{graphicx}
16229 \markdownSetup{rendererPrototypes={
16230     hardLineBreak = {\},
16231     leftBrace = {\textbraceleft},
16232     rightBrace = {\textbraceright},
16233     dollarSign = {\textdollar},
16234     underscore = {\textunderscore},
16235     circumflex = {\textasciicircum},
16236     backslash = {\textbackslash},
16237     tilde = {\textasciitilde},
16238     pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\TeX$  during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>36</sup> we can reliably detect math mode inside the renderer.

---

<sup>36</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

16239 codeSpan = {%
16240   \ifmmode
16241     \text{#1}%
16242   \else
16243     \texttt{#1}%
16244   \fi
16245   }}

```

### 3.3.4.2 Content Blocks

In content block renderer prototypes, display the content as a table using the package `csvsimple` when the raw attribute is `csv`, display the content using the default templates of the package `luaxml` when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```

16246 \ExplSyntaxOn
16247 \markdownSetup{
16248   rendererPrototypes = {
16249     contentBlock = {
16250       \str_case:nnF
16251         { #1 }
16252         {
16253           { csv }
16254           {
16255             \begin { table }
16256               \begin { center }
16257                 \csvautotabular { #3 }
16258               \end { center }
16259               \tl_if_empty:nF
16260                 { #4 }
16261                 { \caption { #4 } }
16262             \end { table }
16263           }
16264           { html }
16265           {

```

If we are using `TeX4ht`<sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16266           \cs_if_exist:NTF
16267             \HCode
16268             {

```

---

Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

<sup>37</sup>See <https://tug.org/tex4ht/>.

```

16269         \if_mode_vertical:
16270         \IgnorePar
16271         \fi:
16272         \EndP
16273         \special
16274         { t4ht* < #3 }
16275         \par
16276         \ShowPar
16277     }
16278     {
16279         \@_luaxml_print_html:n
16280         { #3 }
16281     }
16282 }
16283 { tex }
16284 {
16285     \markdownEscape
16286     { #3 }
16287 }
16288 }
16289 {
16290     \markdownInput
16291     { #3 }
16292 }
16293 },
16294 },
16295 }
16296 \ExplSyntaxOff
16297 \markdownSetup{rendererPrototypes={
16298     ulBegin = {\begin{itemize}},
16299     ulEnd = {\end{itemize}},
16300     olBegin = {\begin{enumerate}},
16301     olItem = {\item{}},
16302     olItemWithNumber = {\item[#1.]},
16303     olEnd = {\end{enumerate}},
16304     dlBegin = {\begin{description}},
16305     dlItem = {\item[#1]},
16306     dlEnd = {\end{description}},
16307     emphasis = {\emph{#1}},
16308     tickedBox = {\$\boxtimes$},
16309     halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

16310 \ExplSyntaxOn
16311 \seq_new:N
16312 \g_@_header_identifiers_seq
16313 \markdownSetup
16314 {

```

```

16315     rendererPrototypes = {
16316         headerAttributeContextBegin = {
16317             \markdownSetup
16318             {
16319                 rendererPrototypes = {
16320                     attributeIdentifier = {
16321                         \seq_gput_right:Nn
16322                         \g_@@_header_identifiers_seq
16323                         { ##1 }
16324                     },
16325                 },
16326             }
16327         },
16328         headerAttributeContextEnd = {
16329             \seq_map_inline:Nn
16330             \g_@@_header_identifiers_seq
16331             { \label { ##1 } }
16332             \seq_gclear:N
16333             \g_@@_header_identifiers_seq
16334         },
16335     },
16336 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

16337 \bool_new:N
16338 \l_@@_header_unnumbered_bool
16339 \markdownSetup
16340 {
16341     rendererPrototypes = {
16342         headerAttributeContextBegin += {
16343             \markdownSetup
16344             {
16345                 rendererPrototypes = {
16346                     attributeClassName = {
16347                         \bool_if:nT
16348                         {
16349                             \str_if_eq_p:nn
16350                             { ##1 }
16351                             { unnumbered } &&
16352                             ! \l_@@_header_unnumbered_bool
16353                         }
16354                     {
16355                         \group_begin:
16356                         \bool_set_true:N
16357                         \l_@@_header_unnumbered_bool
16358                         \c@secnumdepth = 0

```

```

16359             \markdownSetup
16360             {
16361                 rendererPrototypes = {
16362                     sectionBegin = {
16363                         \group_begin:
16364                     },
16365                     sectionEnd = {
16366                         \group_end:
16367                     },
16368                 },
16369             }
16370         },
16371     },
16372 },
16373 }
16374 },
16375 },
16376 }
16377 \ExplSyntaxOff
16378 \markdownSetup{rendererPrototypes={
16379     superscript = {\textsuperscript{#1}},
16380     subscript = {\textsubscript{#1}},
16381     blockQuoteBegin = {\begin{quotation}},
16382     blockQuoteEnd = {\end{quotation}},
16383     inputVerbatim = {\VerbatimInput{#1}},
16384     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
16385     note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16386 \RequirePackage{ltxcmds}
16387 \ExplSyntaxOn
16388 \cs_gset_protected:Npn
16389   \markdownRendererInputFencedCodePrototype#1#2#3
16390   {
16391     \tl_if_empty:nTF
16392       { #2 }
16393     { \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.

```

16394     {
16395       \regex_extract_once:nnN
16396         { \w* }
16397         { #2 }
16398         \l_tmpa_seq
16399       \seq_pop_left:NN

```

```

16400         \l_tmpa_seq
16401         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

16402         \ltx@ifpackageloaded
16403         { minted }
16404         {
16405             \catcode`\%=14\relax
16406             \catcode`\#=6\relax
16407             \exp_args:NV
16408             \inputminted
16409             \l_tmpa_tl
16410             { #1 }
16411             \catcode`\%=12\relax
16412             \catcode`\#=12\relax
16413         }
16414         {

```

When the listings package is loaded, use it for syntax highlighting.

```

16415         \ltx@ifpackageloaded
16416         { listings }
16417         { \lstinputlisting [ language = \l_tmpa_tl ] { #1 } }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

16418             { \markdownRendererInputFencedCode { #1 } { } { } }
16419         }
16420     }
16421 }
16422 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

16423 \ExplSyntaxOn
16424 \def\markdownLATEXStrongEmphasis#1{
16425     \str_if_in:NnTF
16426     \f@series
16427     { b }
16428     { \textnormal{#1} }
16429     { \textbf{#1} }
16430 }
16431 \ExplSyntaxOff
16432 \markdownSetup{rendererPrototypes={strongEmphasis={%
16433     \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

16434 \@ifundefined{chapter}{%
16435     \markdownSetup{rendererPrototypes = {
16436         headingOne = {\section{#1}},
16437         headingTwo = {\subsection{#1}},

```

```

16438     headingThree = {\subsubsection{#1}},
16439     headingFour = {\paragraph{#1}},
16440     headingFive = {\subparagraph{#1}}}}
16441 }{%
16442   \markdownSetup{rendererPrototypes = {
16443     headingOne = {\chapter{#1}},
16444     headingTwo = {\section{#1}},
16445     headingThree = {\subsection{#1}},
16446     headingFour = {\subsubsection{#1}},
16447     headingFive = {\paragraph{#1}},
16448     headingSix = {\subparagraph{#1}}}}
16449 }%

```

#### 3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

16450 \markdownSetup{
16451   rendererPrototypes = {
16452     ulItem = {%
16453       \futurelet\markdownLaTeXCheckbox\markdownLaTeXULItem
16454     },
16455   },
16456 }
16457 \def\markdownLaTeXULItem{%
16458   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
16459     \item[\markdownLaTeXCheckbox]%
16460     \expandafter\@gobble
16461   \else
16462     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
16463       \item[\markdownLaTeXCheckbox]%
16464       \expandafter\expandafter\expandafter\@gobble
16465     \else
16466       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
16467         \item[\markdownLaTeXCheckbox]%
16468         \expandafter\expandafter\expandafter\expandafter
16469         \expandafter\expandafter\expandafter\@gobble
16470       \else
16471         \item{}%
16472       \fi
16473     \fi
16474   \fi
16475 }

```

#### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using  $\text{T}\text{E}\text{X}4\text{ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16476 \@ifundefined{HCode}{}{
16477   \markdownSetup{
16478     rendererPrototypes = {
16479       inlineHtmlTag = {%
16480         \ifvmode
16481           \IgnorePar
16482           \EndP
16483         \fi
16484         \HCode{#1}%
16485       },
16486       inputBlockHtmlElement = {%
16487         \ifvmode
16488           \IgnorePar
16489           \fi
16490           \EndP
16491           \special{t4ht* <#1}%
16492           \par
16493           \ShowPar
16494         },
16495       },
16496     }
16497 }

```

### 3.3.4.6 Citations

Here is a basic implementation for citations that uses the  $\text{L}\text{A}\text{T}\text{E}\text{X}$  `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `BibL $\text{A}\text{T}\text{E}\text{X}$`  `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

16498 \newcount\markdownLaTeXCitationsCounter
16499
16500 % Basic implementation
16501 \long\def@gobblethree#1#2#3{}%
16502 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
16503   \advance\markdownLaTeXCitationsCounter by 1\relax
16504   \ifx\relax#4\relax
16505     \ifx\relax#5\relax
16506       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16507         \relax
16508         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
16509         \expandafter\expandafter\expandafter
16510         \expandafter\expandafter\expandafter\expandafter
16511         \@gobblethree

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.



```

16512     \fi
16513 \else% Before a postnote (#5), dump the accumulator
16514     \ifx\relax#1\relax\else
16515         \cite{#1}%
16516     \fi
16517     \cite[#5]{#6}%
16518     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16519     \relax
16520     \else
16521         \expandafter\expandafter\expandafter
16522         \expandafter\expandafter\expandafter\expandafter
16523         \expandafter\expandafter\expandafter
16524         \expandafter\expandafter\expandafter\expandafter
16525         \markdownLaTeXBasicCitations
16526     \fi
16527     \expandafter\expandafter\expandafter
16528     \expandafter\expandafter\expandafter\expandafter{%
16529     \expandafter\expandafter\expandafter
16530     \expandafter\expandafter\expandafter\expandafter}%
16531     \expandafter\expandafter\expandafter
16532     \expandafter\expandafter\expandafter\expandafter{%
16533     \expandafter\expandafter\expandafter
16534     \expandafter\expandafter\expandafter\expandafter}%
16535     \expandafter\expandafter\expandafter
16536     \@gobblethree
16537 \fi
16538 \else% Before a prenote (#4), dump the accumulator
16539     \ifx\relax#1\relax\else
16540         \cite{#1}%
16541     \fi
16542     \ifnum\markdownLaTeXCitationsCounter>1\relax
16543         \space % Insert a space before the prenote in later citations
16544     \fi
16545     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
16546     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16547     \relax
16548     \else
16549         \expandafter\expandafter\expandafter
16550         \expandafter\expandafter\expandafter\expandafter
16551         \markdownLaTeXBasicCitations
16552     \fi
16553     \expandafter\expandafter\expandafter{%
16554     \expandafter\expandafter\expandafter}%
16555     \expandafter\expandafter\expandafter{%
16556     \expandafter\expandafter\expandafter}%
16557     \expandafter
16558     \@gobblethree

```

```

16559 \fi\markdownLaTeXBasicCitations{#1#2#6},}
16560 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
16561
16562 % Natbib implementation
16563 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
16564 \advance\markdownLaTeXCitationsCounter by 1\relax
16565 \ifx\relax#3\relax
16566 \ifx\relax#4\relax
16567 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16568 \relax
16569 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
16570 \expandafter\expandafter\expandafter
16571 \expandafter\expandafter\expandafter\expandafter
16572 \@gobbletwo
16573 \fi
16574 \else% Before a postnote (#4), dump the accumulator
16575 \ifx\relax#1\relax\else
16576 \citep{#1}%
16577 \fi
16578 \citep[] [#4]{#5}%
16579 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16580 \relax
16581 \else
16582 \expandafter\expandafter\expandafter
16583 \expandafter\expandafter\expandafter\expandafter
16584 \expandafter\expandafter\expandafter
16585 \expandafter\expandafter\expandafter\expandafter
16586 \markdownLaTeXNatbibCitations
16587 \fi
16588 \expandafter\expandafter\expandafter
16589 \expandafter\expandafter\expandafter\expandafter{%
16590 \expandafter\expandafter\expandafter
16591 \expandafter\expandafter\expandafter\expandafter}%
16592 \expandafter\expandafter\expandafter
16593 \@gobbletwo
16594 \fi
16595 \else% Before a prenote (#3), dump the accumulator
16596 \ifx\relax#1\relax\relax\else
16597 \citep{#1}%
16598 \fi
16599 \citep[#3] [#4]{#5}%
16600 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16601 \relax
16602 \else
16603 \expandafter\expandafter\expandafter
16604 \expandafter\expandafter\expandafter\expandafter
16605 \markdownLaTeXNatbibCitations

```

```

16606     \fi
16607     \expandafter\expandafter\expandafter{%
16608     \expandafter\expandafter\expandafter}%
16609     \expandafter
16610     \@gobbletwo
16611     \fi\markdownLaTeXNatbibCitations{#1,#5}}
16612 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
16613     \advance\markdownLaTeXCitationsCounter by 1\relax
16614     \ifx\relax#3\relax
16615         \ifx\relax#4\relax
16616             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16617                 \relax
16618                 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
16619                 \expandafter\expandafter\expandafter
16620                 \expandafter\expandafter\expandafter\expandafter
16621                 \@gobbletwo
16622             \fi
16623         \else% After a prenote or a postnote, dump the accumulator
16624             \ifx\relax#1\relax\relax\else
16625                 \citet{#1}%
16626             \fi
16627             , \citet[#3][#4]{#5}%
16628             \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
16629                 \relax
16630                 ,
16631             \else
16632                 \ifnum
16633                     \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
16634                 \relax
16635                 ,
16636             \fi
16637         \fi
16638         \expandafter\expandafter\expandafter
16639         \expandafter\expandafter\expandafter\expandafter
16640         \markdownLaTeXNatbibTextCitations
16641         \expandafter\expandafter\expandafter
16642         \expandafter\expandafter\expandafter\expandafter{%
16643         \expandafter\expandafter\expandafter
16644         \expandafter\expandafter\expandafter\expandafter}%
16645         \expandafter\expandafter\expandafter
16646         \@gobbletwo
16647     \fi
16648     \else% After a prenote or a postnote, dump the accumulator
16649     \ifx\relax#1\relax\relax\relax\else
16650         \citet{#1}%
16651     \fi
16652     , \citet[#3][#4]{#5}%

```

```

16653 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
16654 \relax
16655 ,
16656 \else
16657 \ifnum
16658 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
16659 \relax
16660 ,
16661 \fi
16662 \fi
16663 \expandafter\expandafter\expandafter
16664 \markdownLaTeXNatbibTextCitations
16665 \expandafter\expandafter\expandafter{%
16666 \expandafter\expandafter\expandafter}%
16667 \expandafter
16668 \@gobbletwo
16669 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
16670
16671 % BibLaTeX implementation
16672 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
16673 \advance\markdownLaTeXCitationsCounter by 1\relax
16674 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16675 \relax
16676 \autocites#1[#3][#4]{#5}%
16677 \expandafter\@gobbletwo
16678 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
16679 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
16680 \advance\markdownLaTeXCitationsCounter by 1\relax
16681 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
16682 \relax
16683 \textcites#1[#3][#4]{#5}%
16684 \expandafter\@gobbletwo
16685 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
16686
16687 \markdownSetup{rendererPrototypes = {
16688 cite = {%
16689 \markdownLaTeXCitationsCounter=1%
16690 \def\markdownLaTeXCitationsTotal{#1}%
16691 \@ifundefined{autocites}{%
16692 \@ifundefined{citep}{%
16693 \expandafter\expandafter\expandafter
16694 \markdownLaTeXBasicCitations
16695 \expandafter\expandafter\expandafter{%
16696 \expandafter\expandafter\expandafter}%
16697 \expandafter\expandafter\expandafter{%
16698 \expandafter\expandafter\expandafter}%
16699 }{%

```

```

16700     \expandafter\expandafter\expandafter
16701     \markdownLaTeXNatbibCitations
16702     \expandafter\expandafter\expandafter{%
16703     \expandafter\expandafter\expandafter}%
16704   }%
16705 }{%
16706   \expandafter\expandafter\expandafter
16707   \markdownLaTeXBibLaTeXCitations
16708   \expandafter{\expandafter}%
16709 }},
16710 textCite = {%
16711   \markdownLaTeXCitationsCounter=1%
16712   \def\markdownLaTeXCitationsTotal{#1}%
16713   \@ifundefined{autocites}{%
16714     \@ifundefined{citep}{%
16715       \expandafter\expandafter\expandafter
16716       \markdownLaTeXBasicTextCitations
16717       \expandafter\expandafter\expandafter{%
16718       \expandafter\expandafter\expandafter}%
16719       \expandafter\expandafter\expandafter{%
16720       \expandafter\expandafter\expandafter}%
16721     }{%
16722       \expandafter\expandafter\expandafter
16723       \markdownLaTeXNatbibTextCitations
16724       \expandafter\expandafter\expandafter{%
16725       \expandafter\expandafter\expandafter}%
16726     }%
16727   }{%
16728     \expandafter\expandafter\expandafter
16729     \markdownLaTeXBibLaTeXTextCitations
16730     \expandafter{\expandafter}%
16731   }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

16732 \RequirePackage{url}
16733 \RequirePackage{expl3}
16734 \ExplSyntaxOn
16735 \cs_gset_protected:Npn
16736   \markdownRendererLinkPrototype
16737   #1#2#3#4
16738   {
16739     \tl_set:Nn \l_tmpa_tl { #1 }
16740     \tl_set:Nn \l_tmpb_tl { #2 }
16741     \bool_set:Nn
16742       \l_tmpa_bool

```

```

16743     {
16744         \tl_if_eq_p:NN
16745         \l_tmpa_tl
16746         \l_tmpb_tl
16747     }
16748     \tl_set:Nn \l_tmpa_tl { #4 }
16749     \bool_set:Nn
16750     \l_tmpb_bool
16751     {
16752         \tl_if_empty_p:N
16753         \l_tmpa_tl
16754     }

```

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.

```

16755     \bool_if:nTF
16756     {
16757         \l_tmpa_bool && \l_tmpb_bool
16758     }
16759     {
16760         \markdownLaTeXRendererAutolink { #2 } { #3 }
16761     }
16762     {
16763         \markdownLaTeXRendererDirectOrIndirectLink
16764         { #1 } { #2 } { #3 } { #4 }
16765     }
16766 }
16767 \def\markdownLaTeXRendererAutolink#1#2{

```

If the URL begins with a hash sign, then we assume that it is a relative reference. Otherwise, we assume that it is an absolute URL.

```

16768     \tl_set:Nn
16769     \l_tmpa_tl
16770     { #2 }
16771     \tl_trim_spaces:N
16772     \l_tmpa_tl
16773     \tl_set:Nx
16774     \l_tmpb_tl
16775     {
16776         \tl_range:Nnn
16777         \l_tmpa_tl
16778         { 1 }
16779         { 1 }
16780     }
16781     \str_if_eq:NNTF
16782     \l_tmpb_tl
16783     \c_hash_str

```

```

16784 {
16785   \tl_set:Nx
16786     \l_tmpb_tl
16787     {
16788       \tl_range:Nnn
16789         \l_tmpa_tl
16790         { 2 }
16791         { -1 }
16792     }
16793   \exp_args:NV
16794     \ref
16795     \l_tmpb_tl
16796 }
16797 {
16798   \url { #2 }
16799 }
16800 }
16801 \ExplSyntaxOff
16802 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
16803   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

### 3.3.4.8 Tables

Here is a basic implementation of tables. If the `booktabs` package is loaded, then it is used to produce horizontal lines.

```

16804 \newcount\markdownLaTeXRowCount
16805 \newcount\markdownLaTeXRowTotal
16806 \newcount\markdownLaTeXColumnCounter
16807 \newcount\markdownLaTeXColumnTotal
16808 \newtoks\markdownLaTeXTable
16809 \newtoks\markdownLaTeXTableAlignment
16810 \newtoks\markdownLaTeXTableEnd
16811 \AtBeginDocument{%
16812   \@ifpackageloaded{booktabs}{%
16813     \def\markdownLaTeXTopRule{\toprule}%
16814     \def\markdownLaTeXMidRule{\midrule}%
16815     \def\markdownLaTeXBottomRule{\bottomrule}%
16816   }{%
16817     \def\markdownLaTeXTopRule{\hline}%
16818     \def\markdownLaTeXMidRule{\hline}%
16819     \def\markdownLaTeXBottomRule{\hline}%
16820   }%
16821 }
16822 \markdownSetup{rendererPrototypes={
16823   table = {%
16824     \markdownLaTeXTable={}%
16825     \markdownLaTeXTableAlignment={}%

```

```

16826 \markdownLaTeXTableEnd={%
16827 \markdownLaTeXBottomRule
16828 \end{tabular}}%
16829 \ifx\empty#1\empty\else
16830 \addto@hook\markdownLaTeXTable{%
16831 \begin{table}
16832 \centering}%
16833 \addto@hook\markdownLaTeXTableEnd{%
16834 \caption{#1}}%
16835 \fi
16836 }
16837 }}

```

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.

```

16838 \ExplSyntaxOn
16839 \seq_new:N
16840 \l_@@_table_identifiers_seq
16841 \markdownSetup {
16842   rendererPrototypes = {
16843     table += {
16844       \seq_map_inline:Nn
16845         \l_@@_table_identifiers_seq
16846         {
16847           \addto@hook
16848             \markdownLaTeXTableEnd
16849             { \label { ##1 } }
16850         }
16851     },
16852   }
16853 }
16854 \markdownSetup {
16855   rendererPrototypes = {
16856     tableAttributeContextBegin = {
16857       \group_begin:
16858       \markdownSetup {
16859         rendererPrototypes = {
16860           attributeIdentifier = {
16861             \seq_put_right:Nn
16862               \l_@@_table_identifiers_seq
16863               { ##1 }
16864           },
16865         },
16866       }
16867     },
16868     tableAttributeContextEnd = {
16869       \group_end:

```



```

16870     },
16871   },
16872 }
16873 \ExplSyntaxOff
16874 \markdownSetup{rendererPrototypes={
16875   table += {%
16876     \ifx\empty#1\empty\else
16877       \addto@hook\markdownLaTeXTableEnd{%
16878         \end{table}}%
16879     \fi
16880     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
16881     \markdownLaTeXRowCounter=0%
16882     \markdownLaTeXRowTotal=#2%
16883     \markdownLaTeXColumnTotal=#3%
16884     \markdownLaTeXRenderTableRow
16885   }
16886 }}
16887 \def\markdownLaTeXRenderTableRow#1{%
16888   \markdownLaTeXColumnCounter=0%
16889   \ifnum\markdownLaTeXRowCounter=0\relax
16890     \markdownLaTeXReadAlignments#1%
16891     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
16892       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
16893         \the\markdownLaTeXTableAlignment}}%
16894     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
16895   \else
16896     \markdownLaTeXRenderTableCell#1%
16897   \fi
16898   \ifnum\markdownLaTeXRowCounter=1\relax
16899     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
16900   \fi
16901   \advance\markdownLaTeXRowCounter by 1\relax
16902   \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
16903     \the\markdownLaTeXTable
16904     \the\markdownLaTeXTableEnd
16905     \expandafter@gobble
16906   \fi\markdownLaTeXRenderTableRow}
16907 \def\markdownLaTeXReadAlignments#1{%
16908   \advance\markdownLaTeXColumnCounter by 1\relax
16909   \if#1d%
16910     \addto@hook\markdownLaTeXTableAlignment{1}%
16911   \else
16912     \addto@hook\markdownLaTeXTableAlignment{#1}%
16913   \fi
16914   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
16915     \expandafter@gobble
16916   \fi\markdownLaTeXReadAlignments}

```

```

16917 \def\markdownLaTeXRenderTableCell#1{%
16918   \advance\markdownLaTeXColumnCounter by 1\relax
16919   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
16920     \addto@hook\markdownLaTeXTable{#1&}%
16921   \else
16922     \addto@hook\markdownLaTeXTable{#1\\}%
16923     \expandafter\@gobble
16924   \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.9 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

16925
16926 \markdownIfOption{lineBlocks}{%
16927   \RequirePackage{verse}
16928   \markdownSetup{rendererPrototypes={
16929     lineBlockBegin = {%
16930       \begingroup
16931       \def\markdownRendererHardLineBreak{\\}%
16932       \begin{verse}%
16933     },
16934     lineBlockEnd = {%
16935       \end{verse}%
16936     \endgroup
16937   },
16938   }}
16939 }{}
16940

```

### 3.3.4.10 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

16941 \ExplSyntaxOn
16942 \keys_define:nn
16943 { markdown / jekyllData }
16944 {
16945   author .code:n = {
16946     \author
16947     { #1 }
16948   },
16949   date .code:n = {
16950     \date
16951     { #1 }
16952   },

```

```

16953     title .code:n = {
16954         \title
16955         { #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.

```

16956     \AddToHook
16957         { begindocument / end }
16958         { \maketitle }
16959     },
16960 }

```

### 3.3.4.11 Marked Text

If the `mark` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement marked text.

```

16961 \@@_if_option:nT
16962 { mark }
16963 {
16964     \sys_if_engine_luatex:TF
16965     {
16966         \RequirePackage
16967         { luacolor }
16968         \RequirePackage
16969         { lua-ul }
16970         \markdownSetup
16971         {
16972             rendererPrototypes = {
16973                 mark = {
16974                     \highLight
16975                     { #1 }
16976                 },
16977             }
16978         }
16979     }
16980     {
16981         \RequirePackage
16982         { xcolor }
16983         \RequirePackage
16984         { soul }
16985         \markdownSetup
16986         {
16987             rendererPrototypes = {
16988                 mark = {
16989                     \hl

```

```

16990         { #1 }
16991     },
16992 }
16993 }
16994 }
16995 }

```

### 3.3.4.12 Strike-Through

If the `strikeThrough` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement strike-throughs.

```

16996 \@@_if_option:nT
16997 { strikeThrough }
16998 {
16999   \sys_if_engine_luatex:TF
17000   {
17001     \RequirePackage
17002     { lua-ul }
17003     \markdownSetup
17004     {
17005       rendererPrototypes = {
17006         strikeThrough = {
17007           \strikeThrough
17008             { #1 }
17009         },
17010     }
17011   }
17012 }
17013 {
17014   \RequirePackage
17015   { soul }
17016   \markdownSetup
17017   {
17018     rendererPrototypes = {
17019       strikeThrough = {
17020         \st
17021           { #1 }
17022       },
17023   }
17024 }
17025 }
17026 }

```

### 3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form `<key>=<value>` 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.

```

17027 \seq_new:N
17028   \l_@@_image_identifiers_seq
17029 \markdownSetup {
17030   rendererPrototypes = {
17031     image = {
17032       \tl_if_empty:nTF
17033         { #4 }
17034         {
17035           \begin { center }
17036             \includegraphics
17037               [ alt = { #1 } ]
17038               { #3 }
17039           \end { center }
17040         }
17041       {
17042         \begin { figure }
17043           \begin { center }
17044             \includegraphics
17045               [ alt = { #1 } ]
17046               { #3 }
17047             \caption { #4 }
17048             \seq_map_inline:Nn
17049               \l_@@_image_identifiers_seq
17050               { \label { ##1 } }
17051           \end { center }
17052         \end { figure }
17053       }
17054     },
17055   }
17056 }
17057 \@@_if_option:nT
17058   { linkAttributes }
17059   {
17060     \RequirePackage { graphicx }
17061   }
17062 \markdownSetup {
17063   rendererPrototypes = {
17064     imageAttributeContextBegin = {
17065       \group_begin:

```

```

17066     \markdownSetup {
17067         rendererPrototypes = {
17068             attributeIdentifier = {
17069                 \seq_put_right:Nn
17070                 \l_@@_image_identifiers_seq
17071                 { ##1 }
17072             },
17073             attributeKeyValue = {
17074                 \setkeys
17075                 { Gin }
17076                 { { ##1 } = { ##2 } }
17077             },
17078         },
17079     }
17080 },
17081 imageAttributeContextEnd = {
17082     \group_end:
17083 },
17084 },
17085 }
17086 \ExplSyntaxOff

```

### 3.3.4.14 Raw Attributes

In the raw block and inline raw span renderer prototypes, display the content using the default templates of the package `luaxml` when the raw attribute is `html` and default to the plain TeX renderer prototypes otherwise, translating raw attribute `latex` to `tex`.

```

17087 \ExplSyntaxOn
17088 \cs_new:Nn
17089   \@@_luaxml_print_html:n
17090   {
17091     \luabridge_now:n
17092     {
17093       local~input_file = assert(io.open(" #1 ", "r"))
17094       local~input = assert(input_file:read("*a"))
17095       assert(input_file:close())
17096       input = "<body>" .. input .. "</body>"
17097       local~dom = require("luaxml-domobject").html_parse(input)
17098       local~output = require("luaxml-htmltemplates"):process_dom(dom)
17099       print(output)
17100     }
17101   }
17102 \cs_gset_protected:Npn
17103   \markdownRendererInputRawInlinePrototype#1#2
17104   {
17105     \str_case:nnF

```

```

17106     { #2 }
17107     {
17108         { latex }
17109         {
17110             \@@_plain_tex_default_input_raw_inline:nn
17111             { #1 }
17112             { tex }
17113         }
17114     { html }
17115     {

```

If we are using  $\text{T}\text{E}\text{X}4\text{ht}$ <sup>39</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17116         \cs_if_exist:NTF
17117         \HCode
17118         {
17119             \if_mode_vertical:
17120             \IgnorePar
17121             \EndP
17122             \fi:
17123             \special
17124             { t4ht* < #1 }
17125         }
17126     {
17127         \@@_luaxml_print_html:n
17128         { #1 }
17129     }
17130 }
17131 }
17132 {
17133     \@@_plain_tex_default_input_raw_inline:nn
17134     { #1 }
17135     { #2 }
17136 }
17137 }
17138 \cs_gset_protected:Npn
17139 \markdownRendererInputRawBlockPrototype#1#2
17140 {
17141     \str_case:nnF
17142     { #2 }
17143     {
17144         { latex }
17145         {
17146             \@@_plain_tex_default_input_raw_block:nn
17147             { #1 }
17148             { tex }

```

---

<sup>39</sup>See <https://tug.org/tex4ht/>.

```

17149     }
17150     { html }
17151     {

```

If we are using  $\text{T}_{\text{E}}\text{X}4\text{ht}$ <sup>40</sup>, we will pass HTML elements to the output HTML document unchanged.

```

17152         \cs_if_exist:NTF
17153         \HCode
17154         {
17155             \if_mode_vertical:
17156             \IgnorePar
17157             \fi:
17158             \EndP
17159             \special
17160             { t4ht* < #1 }
17161             \par
17162             \ShowPar
17163         }
17164     {
17165         \@@_luaxml_print_html:n
17166         { #1 }
17167     }
17168 }
17169 {
17170 {
17171     \@@_plain_tex_default_input_raw_block:nn
17172     { #1 }
17173     { #2 }
17174 }
17175 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  labels for referencing the last  $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$  counter that has been incremented in e.g. ordered lists.

```

17176 \seq_new:N
17177 \l_@@_bracketed_span_identifiers_seq
17178 \markdownSetup {
17179     rendererPrototypes = {
17180         bracketedSpanAttributeContextBegin = {
17181             \group_begin:
17182             \markdownSetup {
17183                 rendererPrototypes = {
17184                     attributeIdentifier = {
17185                         \seq_put_right:Nn

```

---

<sup>40</sup>See <https://tug.org/tex4ht/>.



```

17186             \l_@@_bracketed_span_identifiers_seq
17187             { ##1 }
17188         },
17189     },
17190 }
17191 },
17192 bracketedSpanAttributeContextEnd = {
17193     \seq_map_inline:Nn
17194     \l_@@_bracketed_span_identifiers_seq
17195     { \label { ##1 } }
17196     \group_end:
17197 },
17198 },
17199 }
17200 \ExplSyntaxOff
17201 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}`

```

### 3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `inputenc` package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` package.

```

17202 \newcommand\markdownMakeOther{%
17203     \count0=128\relax
17204     \loop
17205         \catcode\count0=11\relax
17206         \advance\count0 by 1\relax
17207     \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.

```

17208 \def\markdownMakeOther{%
17209     \count0=128\relax
17210     \loop
17211         \catcode\count0=11\relax

```

```

17212     \advance\count0 by 1\relax
17213     \ifnum\count0<256\repeat

```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTeXt.

```

17214     \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```

17215 \long\def\inputmarkdown{%
17216     \dosingleempty
17217     \doinputmarkdown}%
17218 \long\def\doinputmarkdown[#1]#2{%
17219     \begingroup
17220     \iffirstargument
17221         \setupmarkdown[#1]%
17222     \fi
17223     \markdownInput{#2}%
17224 \endgroup}%
17225 \long\def\inputyaml{%
17226     \dosingleempty
17227     \doinputyaml}%
17228 \long\def\doinputyaml[#1]#2{%
17229     \doinputmarkdown
17230     [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s TeX, trailing spaces are removed very early on when a line is being put to the input buffer. [19, sec. 31]. According to Eijkhout [20, 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)TeX, but ConTeXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTeXt MkIV and therefore to insert hard line breaks into markdown text.

```

17231 \startluacode
17232     document.markdown_buffering = false
17233     local function preserve_trailing_spaces(line)
17234         if document.markdown_buffering then
17235             line = line:gsub("[ \t][ \t]$", "\t\t")
17236         end
17237         return line
17238     end
17239     resolvers.installinputlinehandler(preserve_trailing_spaces)
17240 \stopluacode

```

```

17241 \begingroup
17242 \catcode`\|=0%
17243 \catcode`\|=12%
17244 |gdef|startmarkdown{%
17245     |ctlua{document.markdown_buffering = true}%
17246     |markdownReadAndConvert{\stopmarkdown}%
17247         {|stopmarkdown}}%
17248 |gdef|stopmarkdown{%
17249     |ctlua{document.markdown_buffering = false}%
17250     |markdownEnd}%
17251 |gdef|startyaml{%
17252     |begingroup
17253     |ctlua{document.markdown_buffering = true}%
17254     |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
17255     |markdownReadAndConvert{\stopyaml}%
17256         {|stopyaml}}%
17257 |gdef|stopyaml{%
17258     |ctlua{document.markdown_buffering = false}%
17259     |yamlEnd}%
17260 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in Con $\TeX$ t themes provided with the Markdown package.

```

17261 \ExplSyntaxOn
17262 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
17263 \prop_new:N \g_@@_context_loaded_themes_versions_prop
17264 \cs_gset:Nn % noqa: w401
17265 \@@_load_theme:nnn
17266 {

```

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  $\TeX$  theme instead.

```

17267     \bool_if:nTF
17268     {
17269         \bool_lazy_or_p:nn
17270         {
17271             \prop_if_in_p:Nn
17272             \g_@@_context_built_in_themes_prop
17273             { #1 }
17274         }
17275     {

```

```

17276         \file_if_exist_p:n
17277         { t - markdown theme #3.tex }
17278     }
17279 }
17280 {
17281     \prop_get:NnNTF
17282     \g_@@_context_loaded_themes_linenos_prop
17283     { #1 }
17284     \l_tmpa_tl
17285     {
17286         \prop_get:NnN
17287         \g_@@_context_loaded_themes_versions_prop
17288         { #1 }
17289         \l_tmpb_tl
17290         \str_if_eq:nVTF
17291         { #2 }
17292         \l_tmpb_tl
17293         {
17294             \msg_warning:nnnVn
17295             { markdown }
17296             { repeatedly-loaded-context-theme }
17297             { #1 }
17298             \l_tmpa_tl
17299             { #2 }
17300         }
17301     }
17302     \msg_error:nnnnVV
17303     { markdown }
17304     { different-versions-of-context-theme }
17305     { #1 }
17306     { #2 }
17307     \l_tmpb_tl
17308     \l_tmpa_tl
17309 }
17310 }
17311 {
17312     \prop_gput:Nnx
17313     \g_@@_context_loaded_themes_linenos_prop
17314     { #1 }
17315     { \tex_the:D \tex_inputlineno:D } % noqa: W200
17316     \prop_gput:Nnn
17317     \g_@@_context_loaded_themes_versions_prop
17318     { #1 }
17319     { #2 }

```

Load built-in plain T<sub>E</sub>X themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

17320         \prop_if_in:NnTF
17321         \g_@@_context_built_in_themes_prop
17322         { #1 }
17323         {
17324             \msg_info:nnnn
17325             { markdown }
17326             { loading-built-in-context-theme }
17327             { #1 }
17328             { #2 }
17329         \prop_item:Nn
17330         \g_@@_context_built_in_themes_prop
17331         { #1 }
17332     }
17333     {
17334         \msg_info:nnnn
17335         { markdown }
17336         { loading-context-theme }
17337         { #1 }
17338         { #2 }
17339         \usemodule
17340         [ t ]
17341         [ markdown theme #3 ]
17342     }
17343 }
17344 }
17345 {
17346     \@@_plain_tex_load_theme:nnn
17347     { #1 }
17348     { #2 }
17349     { #3 }
17350 }
17351 }
17352 \msg_new:nnn
17353 { markdown }
17354 { loading-built-in-context-theme }
17355 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
17356 \msg_new:nnn
17357 { markdown }
17358 { loading-context-theme }
17359 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
17360 \msg_new:nnn
17361 { markdown }
17362 { repeatedly-loaded-context-theme }
17363 {
17364     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
17365     loaded~on~line~#2,~not~loading~it~again
17366 }

```

```

17367 \msg_new:nnn
17368   { markdown }
17369   { different-versions-of-context-theme }
17370   {
17371     Tried-to-load-version-#2-of-ConTeXt-Markdown-theme-#1~
17372     but-version-#3-has-already-been-loaded-on-line-#4
17373   }
17374 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain TeX theme with the default definitions for plain TeX:

```
17375 \markdownLoadPlainTeXTheme
```

Next, the ConTeXt theme overrides some of the plain TeX definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

17376 \markdownIfOption{plain}{\iffalse}{\iftrue}
17377 \def\markdownRendererHardLineBreakPrototype{\blank}%
17378 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
17379 \def\markdownRendererRightBracePrototype{\textbraceright}%
17380 \def\markdownRendererDollarSignPrototype{\textdollar}%
17381 \def\markdownRendererPercentSignPrototype{\percent}%
17382 \def\markdownRendererUnderscorePrototype{\textunderscore}%
17383 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
17384 \def\markdownRendererBackslashPrototype{\textbackslash}%
17385 \def\markdownRendererTildePrototype{\textasciitilde}%
17386 \def\markdownRendererPipePrototype{\char`|}%
17387 \def\markdownRendererLinkPrototype#1#2#3#4{%
17388   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
17389   \fi\texttt<\hyphenatedurl{#3}>}}%
17390 \usemodule[database]
17391 \defineseparatedlist
17392   [MarkdownConTeXtCSV]
17393   [separator={,},
17394   before=\bTABLE,after=\eTABLE,
17395   first=\bTR,last=\eTR,
17396   left=\bTD,right=\eTD]
17397 \def\markdownConTeXtCSV{csv}
17398 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
17399   \def\markdownConTeXtCSV@arg{#1}%
17400   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
17401     \placetable[] [tab:#1]{#4}{%

```

```

17402     \processseparatedfile [MarkdownConTeXtCSV] [#3]}%
17403     \else
17404     \markdownInput{#3}%
17405     \fi}%
17406 \def\markdownRendererImagePrototype#1#2#3#4{%
17407     \placefigure[] []{#4}{\externalfigure[#3]}%
17408 \def\markdownRendererUlBeginPrototype{\startitemize}%
17409 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
17410 \def\markdownRendererUlItemPrototype{\item}%
17411 \def\markdownRendererUlEndPrototype{\stopitemize}%
17412 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
17413 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
17414 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
17415 \def\markdownRendererOlItemPrototype{\item}%
17416 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
17417 \def\markdownRendererOlEndPrototype{\stopitemize}%
17418 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
17419 \definedescription
17420     [MarkdownConTeXtDlItemPrototype]
17421     [location=hanging,
17422     margin=standard,
17423     headstyle=bold]%
17424 \definestartstop
17425     [MarkdownConTeXtDlPrototype]
17426     [before=\blank,
17427     after=\blank]%
17428 \definestartstop
17429     [MarkdownConTeXtDlTightPrototype]
17430     [before=\blank\startpacked,
17431     after=\stoppacked\blank]%
17432 \def\markdownRendererDlBeginPrototype{%
17433     \startMarkdownConTeXtDlPrototype}%
17434 \def\markdownRendererDlBeginTightPrototype{%
17435     \startMarkdownConTeXtDlTightPrototype}%
17436 \def\markdownRendererDlItemPrototype#1{%
17437     \startMarkdownConTeXtDlItemPrototype{#1}}%
17438 \def\markdownRendererDlItemEndPrototype{%
17439     \stopMarkdownConTeXtDlItemPrototype}%
17440 \def\markdownRendererDlEndPrototype{%
17441     \stopMarkdownConTeXtDlPrototype}%
17442 \def\markdownRendererDlEndTightPrototype{%
17443     \stopMarkdownConTeXtDlTightPrototype}%
17444 \def\markdownRendererEmphasisPrototype#1{\em#1}%
17445 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
17446 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
17447 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
17448 \def\markdownRendererLineBlockBeginPrototype{%

```

```

17449 \begingroup
17450 \def\markdownRendererHardLineBreak{
17451 }%
17452 \startlines
17453 }%
17454 \def\markdownRendererLineBlockEndPrototype{%
17455 \stoptlines
17456 \endgroup
17457 }%
17458 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

17459 \ExplSyntaxOn
17460 \cs_gset:Npn
17461 \markdownRendererInputFencedCodePrototype#1#2#3
17462 {
17463 \tl_if_empty:nTF
17464 { #2 }
17465 { \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 `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptyping [latex] [option=TEX]

\starttext
 \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
 Hello world!
\end{document}
~~~
 \stopmarkdown
\stoptext

```

```

17466 {
17467 \regex_extract_once:nnN
17468 { \w* }
17469 { #2 }
17470 \l_tmpa_seq

```



```

17471     \seq_pop_left:NN
17472     \l_tmpa_seq
17473     \l_tmpa_tl
17474     \typefile[ \l_tmpa_tl ][] {#1}
17475   }
17476 }
17477 \ExplSyntaxOff
17478 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
17479 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
17480 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
17481 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
17482 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
17483 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
17484 \def\markdownRendererThematicBreakPrototype{%
17485   \blackrule[height=1pt, width=\hsize]}%
17486 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
17487 \def\markdownRendererTickedBoxPrototype{\$ \boxtimes $}
17488 \def\markdownRendererHalfTickedBoxPrototype{\$ \boxdot $}
17489 \def\markdownRendererUntickedBoxPrototype{\$ \square $}
17490 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
17491 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
17492 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
17493 \def\markdownRendererDisplayMathPrototype#1{%
17494   \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

17495 \newcount\markdownConTeXtRowCounter
17496 \newcount\markdownConTeXtRowTotal
17497 \newcount\markdownConTeXtColumnCounter
17498 \newcount\markdownConTeXtColumnTotal
17499 \newtoks\markdownConTeXtTable
17500 \newtoks\markdownConTeXtTableFloat
17501 \def\markdownRendererTablePrototype#1#2#3{%
17502   \markdownConTeXtTable={}%
17503   \ifx\empty#1\empty
17504     \markdownConTeXtTableFloat={%
17505       \the\markdownConTeXtTable}%
17506   \else
17507     \markdownConTeXtTableFloat={%
17508       \placetable{#1}{\the\markdownConTeXtTable}}%
17509   \fi
17510   \begingroup
17511   \setupTABLE[r] [each] [topframe=off, bottomframe=off,
17512     leftframe=off, rightframe=off]
17513   \setupTABLE[c] [each] [topframe=off, bottomframe=off,

```

```

17514             leftframe=off, rightframe=off]
17515 \setupTABLE[r][1][topframe=on, bottomframe=on]
17516 \setupTABLE[r][#1][bottomframe=on]
17517 \markdownConTeXtRowCounter=0%
17518 \markdownConTeXtRowTotal=#2%
17519 \markdownConTeXtColumnTotal=#3%
17520 \markdownConTeXtRenderTableRow}
17521 \def\markdownConTeXtRenderTableRow#1{%
17522 \markdownConTeXtColumnCounter=0%
17523 \ifnum\markdownConTeXtRowCounter=0\relax
17524 \markdownConTeXtReadAlignments#1%
17525 \markdownConTeXtTable={\bTABLE}%
17526 \else
17527 \markdownConTeXtTable=\expandafter{%
17528 \the\markdownConTeXtTable\bTR}%
17529 \markdownConTeXtRenderTableCell#1%
17530 \markdownConTeXtTable=\expandafter{%
17531 \the\markdownConTeXtTable\eTR}%
17532 \fi
17533 \advance\markdownConTeXtRowCounter by 1\relax
17534 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
17535 \markdownConTeXtTable=\expandafter{%
17536 \the\markdownConTeXtTable\eTABLE}%
17537 \the\markdownConTeXtTableFloat
17538 \endgroup
17539 \expandafter\gobbleoneargument
17540 \fi\markdownConTeXtRenderTableRow}
17541 \def\markdownConTeXtReadAlignments#1{%
17542 \advance\markdownConTeXtColumnCounter by 1\relax
17543 \if#1d%
17544 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
17545 \fi\if#1l%
17546 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
17547 \fi\if#1c%
17548 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
17549 \fi\if#1r%
17550 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
17551 \fi
17552 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
17553 \else
17554 \expandafter\gobbleoneargument
17555 \fi\markdownConTeXtReadAlignments}
17556 \def\markdownConTeXtRenderTableCell#1{%
17557 \advance\markdownConTeXtColumnCounter by 1\relax
17558 \markdownConTeXtTable=\expandafter{%
17559 \the\markdownConTeXtTable\bTD#1\eTD}%
17560 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax

```

```

17561 \else
17562   \expandafter\gobbleoneargument
17563 \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`.

```

17564 \ExplSyntaxOn
17565 \cs_gset:Npn
17566   \markdownRendererInputRawInlinePrototype#1#2
17567   {
17568     \str_case:nnF
17569       { #2 }
17570       {
17571         { latex }
17572         {
17573           \@@_plain_tex_default_input_raw_inline:nn
17574             { #1 }
17575             { context }
17576         }
17577       }
17578     {
17579       \@@_plain_tex_default_input_raw_inline:nn
17580         { #1 }
17581         { #2 }
17582     }
17583   }
17584 \cs_gset:Npn
17585   \markdownRendererInputRawBlockPrototype#1#2
17586   {
17587     \str_case:nnF
17588       { #2 }
17589       {
17590         { context }
17591         {
17592           \@@_plain_tex_default_input_raw_block:nn
17593             { #1 }
17594             { tex }
17595         }
17596       }
17597     {
17598       \@@_plain_tex_default_input_raw_block:nn
17599         { #1 }
17600         { #2 }
17601     }
17602   }

```

```

17603 \cs_gset_eq:NN
17604 \markdownRendererInputRawBlockPrototype
17605 \markdownRendererInputRawInlinePrototype
17606 \fi % Closes ` \markdownIfOption{plain}{\iffalse}{\iftrue}`
17607 \ExplSyntaxOff
17608 \stopmodule
17609 \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).

```

17610 \ExplSyntaxOn
17611 \str_if_eq:VVT
17612 \c_@@_top_layer_tl
17613 \c_@@_option_layer_context_tl
17614 {
17615   \use:c
17616     { ExplSyntaxOff }
17617   \@@_if_option:nF
17618     { noDefaults }
17619     {
17620       \@@_if_option:nTF
17621         { experimental }
17622         {
17623           \@@_setup:n
17624             { theme = witiko/markdown/defaults@experimental }
17625         }
17626         {
17627           \@@_setup:n
17628             { theme = witiko/markdown/defaults }
17629         }
17630     }
17631   \use:c
17632     { ExplSyntaxOn }
17633 }
17634 \ExplSyntaxOff
17635 \stopmodule
17636 \protect

```

## References

- [1] 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).
- [2] 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).

- [3] 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).
- [4] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [5] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [6] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [7] 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).
- [8] 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.
- [9] Frank Mittelbach. *The doc and shortverb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [10] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [11] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [12] 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).
- [13] 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).
- [14] 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).
- [15] 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).

- [16] 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).
- [17] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [18] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [19] 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.
- [20] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician’s Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

<code>autoIdentifiers</code>	21, 33, 86, 102
<code>blankBeforeBlockquote</code>	22
<code>blankBeforeCodeFence</code>	22
<code>blankBeforeDivFence</code>	22
<code>blankBeforeHeading</code>	23
<code>blankBeforeList</code>	23
<code>bracketedSpans</code>	23, 88, 488
<code>breakableBlockquotes</code>	24
<code>cacheDir</code>	4, 16, 19, 59, 60, 160, 191, 400, 421, 440
<code>citationNbsps</code>	24
<code>citations</code>	24, 91, 92
<code>codeSpans</code>	25
<code>contentBlocks</code>	20, 25, 35
<code>contentBlocksLanguageMap</code>	20
<code>contentLevel</code>	26
<code>debugExtensions</code>	9, 20, 26, 344
<code>debugExtensionsFileName</code>	20, 26
<code>defaultOptions</code>	10, 52, 398, 399
<code>definitionLists</code>	27, 96
<code>depth_first_search</code>	168
<code>\DocumentMetadata</code>	459
<code>eagerCache</code>	16, 397

ensureJekyllData	27
entities.char_entity	245
entities.dec_entity	245
entities.hex_entity	245
entities.hex_entity_with_x_char	245
escape_minimal	250
escape_programmatic_text	250
escape_typographic_text	250
expandtabs	305
expectJekyllData	27, 28
experimental	5, 17, 459
extensions	29, 168, 349
extensions.bracketed_spans	349
extensions.citations	350
extensions.content_blocks	354
extensions.definition_lists	357
extensions.fancy_lists	360
extensions.fenced_code	365
extensions.fenced_divs	371
extensions.header_attributes	375
extensions.inline_code_attributes	377
extensions.jekyll_data	393
extensions.line_blocks	377
extensions.link_attributes	379
extensions.mark	378
extensions.notes	380
extensions.pipe_table	383
extensions.raw_inline	387
extensions.strike_through	388
extensions.subscripts	388
extensions.superscripts	389
extensions.tex_math	390
fancyLists	30, 113–118, 459
fencedCode	31, 40, 93, 100, 119, 411, 414
fencedCodeAttributes	31, 86, 100, 414
fencedDiv	101
fencedDivs	32, 42
finalizeCache	17, 21, 32, 33, 59, 60, 159, 397, 399
frozenCache	21, 32, 60, 76, 79, 159, 412, 420
frozenCacheCounter	33, 399, 448, 449
frozenCacheFileName	21, 32, 59, 399

<code>\g_markdown_diagrams_infostrings_prop</code>	416
<code>gfmAutoIdentifiers</code>	21, 33, 86, 102
<code>hashEnumerators</code>	33
<code>headerAttributes</code>	34, 42, 86, 102
<code>html</code>	34, 105, 472
<code>hybrid</code>	35, 35, 40, 46, 48, 63, 79, 120, 160, 250, 306, 448
<code>inlineCodeAttributes</code>	36, 86, 94
<code>inlineNotes</code>	36
<code>\input</code>	56, 399
<code>\inputmarkdown</code>	163, 164, 165, 490
<code>inputTempFileName</code>	61, 63, 442, 443, 445, 446
<code>\inputyaml</code>	163, 165, 490
<code>iterlines</code>	305
<code>jekyllData</code>	3, 27, 28, 37, 129–132, 135
<code>\l_file_search_path_seq</code>	447
<code>languages_json</code>	354, 354
<code>lineBlocks</code>	38, 108
<code>linkAttributes</code>	37, 86, 106, 110, 325, 485
<code>mark</code>	38, 111, 483
<code>\markdown</code>	155, 156, 452
<code>markdown</code>	154, 154, 155, 450, 451
<code>markdown*</code>	154, 154, 155, 159, 450
<code>\markdownBegin</code>	54, 54, 55, 56, 152–154, 156, 163
<code>\markdownCleanup</code>	441
<code>\markdownConvert</code>	441
<code>\markdownEnd</code>	54, 54, 55, 56, 153–156, 163
<code>\markdownError</code>	152, 152
<code>\markdownEscape</code>	54, 57, 449
<code>\markdownIfOption</code>	58
<code>\markdownIfSnippetExists</code>	81
<code>\markdownInfo</code>	152, 152
<code>\markdownInput</code>	54, 56, 154, 157, 159, 164, 447, 450
<code>\markdownInputFilename</code>	440
<code>\markdownInputFileStream</code>	441
<code>\markdownInputPlainTeX</code>	450
<code>\markdownLoadPlainTeXTheme</code>	160, 167, 410
<code>\markdownLuaExecute</code>	444, 447
<code>\markdownLuaOptions</code>	437, 441



<code>\markdownMakeOther</code>	152, 489
<code>\markdownOptionFinalizeCache</code>	59
<code>\markdownOptionFrozenCache</code>	59
<code>\markdownOptionHybrid</code>	63
<code>\markdownOptionInputTempFileName</code>	60
<code>\markdownOptionNoDefaults</code>	62
<code>\markdownOptionOutputDir</code>	60, 61, 63, 64
<code>\markdownOptionPlain</code>	61
<code>\markdownOptionStripPercentSigns</code>	62
<code>\markdownOutputFileStream</code>	441
<code>\markdownPrepare</code>	440
<code>\markdownPrepareInputFilename</code>	440
<code>\markdownPrepareLuaOptions</code>	437
<code>\markdownReadAndConvert</code>	152, 441, 450–452, 490
<code>\markdownReadAndConvertProcessLine</code>	443, 443
<code>\markdownReadAndConvertStripPercentSigns</code>	442
<code>\markdownReadAndConvertTab</code>	441
<code>\markdownRendererAttributeClassName</code>	86
<code>\markdownRendererAttributeIdentifier</code>	86
<code>\markdownRendererAttributeKeyValue</code>	86
<code>\markdownRendererBlockQuoteBegin</code>	87
<code>\markdownRendererBlockQuoteEnd</code>	88
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	88
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	88
<code>\markdownRendererCite</code>	91, 92
<code>\markdownRendererCodeSpan</code>	93
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	94
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	94
<code>\markdownRendererContentBlock</code>	94, 95
<code>\markdownRendererContentBlockCode</code>	95
<code>\markdownRendererContentBlockOnlineImage</code>	95
<code>\markdownRendererDisplayMath</code>	126
<code>\markdownRendererDlBegin</code>	96
<code>\markdownRendererDlBeginTight</code>	96
<code>\markdownRendererDlDefinitionBegin</code>	97
<code>\markdownRendererDlDefinitionEnd</code>	98
<code>\markdownRendererDlEnd</code>	98
<code>\markdownRendererDlEndTight</code>	99
<code>\markdownRendererDlItem</code>	97
<code>\markdownRendererDlItemEnd</code>	97
<code>\markdownRendererDocumentBegin</code>	111
<code>\markdownRendererDocumentEnd</code>	111

<code>\markdownRendererEllipsis</code>	42, 99
<code>\markdownRendererEmphasis</code>	99, 139
<code>\markdownRendererError</code>	128
<code>\markdownRendererFancyOlBegin</code>	113, 114
<code>\markdownRendererFancyOlBeginTight</code>	114
<code>\markdownRendererFancyOlEnd</code>	118
<code>\markdownRendererFancyOlEndTight</code>	118
<code>\markdownRendererFancyOlItem</code>	116
<code>\markdownRendererFancyOlItemEnd</code>	116
<code>\markdownRendererFancyOlItemWithNumber</code>	116
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	100
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	100
<code>\markdownRendererFencedDivAttributeContextBegin</code>	101
<code>\markdownRendererFencedDivAttributeContextEnd</code>	101
<code>\markdownRendererHalfTickedBox</code>	127
<code>\markdownRendererHardLineBreak</code>	109
<code>\markdownRendererHeaderAttributeContextBegin</code>	102
<code>\markdownRendererHeaderAttributeContextEnd</code>	102
<code>\markdownRendererHeadingFive</code>	104
<code>\markdownRendererHeadingFour</code>	104
<code>\markdownRendererHeadingOne</code>	102
<code>\markdownRendererHeadingSix</code>	104
<code>\markdownRendererHeadingThree</code>	103
<code>\markdownRendererHeadingTwo</code>	103
<code>\markdownRendererImage</code>	106
<code>\markdownRendererImageAttributeContextBegin</code>	106
<code>\markdownRendererImageAttributeContextEnd</code>	106
<code>\markdownRendererInlineHtmlComment</code>	105
<code>\markdownRendererInlineHtmlTag</code>	105
<code>\markdownRendererInlineMath</code>	126
<code>\markdownRendererInputBlockHtmlElement</code>	105
<code>\markdownRendererInputFencedCode</code>	93
<code>\markdownRendererInputRawBlock</code>	119
<code>\markdownRendererInputRawInline</code>	118
<code>\markdownRendererInputVerbatim</code>	92
<code>\markdownRendererInterblockSeparator</code>	107
<code>\markdownRendererJekyllDataBegin</code>	129
<code>\markdownRendererJekyllDataBoolean</code>	131
<code>\markdownRendererJekyllDataEmpty</code>	135
<code>\markdownRendererJekyllDataEnd</code>	129
<code>\markdownRendererJekyllDataMappingBegin</code>	130
<code>\markdownRendererJekyllDataMappingEnd</code>	130

<code>\markdownRendererJekyllDataNumber</code>	132
<code>\markdownRendererJekyllDataProgrammaticString</code>	132, 132, 133
<code>\markdownRendererJekyllDataSequenceBegin</code>	131
<code>\markdownRendererJekyllDataSequenceEnd</code>	131
<code>\markdownRendererJekyllDataString</code>	133, 137
<code>\markdownRendererJekyllDataStringPrototype</code>	148
<code>\markdownRendererJekyllDataTypographicString</code>	132, 132, 133, 394
<code>\markdownRendererLineBlockBegin</code>	108
<code>\markdownRendererLineBlockEnd</code>	108
<code>\markdownRendererLink</code>	109, 139
<code>\markdownRendererLinkAttributeContextBegin</code>	110
<code>\markdownRendererLinkAttributeContextEnd</code>	110
<code>\markdownRendererMark</code>	111
<code>\markdownRendererNbsp</code>	112
<code>\markdownRendererNote</code>	112
<code>\markdownRendererOlBegin</code>	113
<code>\markdownRendererOlBeginTight</code>	113
<code>\markdownRendererOlEnd</code>	117
<code>\markdownRendererOlEndTight</code>	117
<code>\markdownRendererOlItem</code>	43, 114
<code>\markdownRendererOlItemEnd</code>	115
<code>\markdownRendererOlItemWithNumber</code>	42, 115
<code>\markdownRendererParagraphSeparator</code>	108
<code>\markdownRendererReplacementCharacter</code>	120
<code>\markdownRendererSectionBegin</code>	119
<code>\markdownRendererSectionEnd</code>	119
<code>\markdownRendererSoftLineBreak</code>	109
<code>\markdownRendererStrikeThrough</code>	123
<code>\markdownRendererStrongEmphasis</code>	100
<code>\markdownRendererSubscript</code>	124
<code>\markdownRendererSuperscript</code>	124
<code>\markdownRendererTable</code>	125
<code>\markdownRendererTableAttributeContextBegin</code>	125
<code>\markdownRendererTableAttributeContextEnd</code>	125
<code>\markdownRendererTextCite</code>	92
<code>\markdownRendererThematicBreak</code>	127
<code>\markdownRendererTickedBox</code>	127
<code>\markdownRendererUlBegin</code>	89
<code>\markdownRendererUlBeginTight</code>	89
<code>\markdownRendererUlEnd</code>	91
<code>\markdownRendererUlEndTight</code>	91
<code>\markdownRendererUlItem</code>	90

<code>\markdownRendererUListEnd</code>	<i>90</i>
<code>\markdownRendererUntickedBox</code>	<i>127</i>
<code>\markdownRendererWarning</code>	<i>128</i>
<code>\markdownSetup</code>	<i>58, 58, 63, 158, 159, 165, 451, 458</i>
<code>\markdownSetupSnippet</code>	<i>80, 80</i>
<code>\markdownThemeVersion</code>	<i>70, 70</i>
<code>\markdownWarning</code>	<i>152, 152</i>
<code>\markinline</code>	<i>54, 55, 56, 154, 156, 157, 445, 449</i>
<code>\markinlinePlainTeX</code>	<i>449</i>
<code>new</code>	<i>7, 18, 397, 399</i>
<code>notes</code>	<i>39, 112</i>
<code>parsers</code>	<i>267, 304</i>
<code>parsers.commented_line</code>	<i>285</i>
<code>parsers.unicode</code>	<i>269</i>
<code>pipeTables</code>	<i>7, 39, 45, 125</i>
<code>preserveTabs</code>	<i>40, 43, 305</i>
<code>rawAttribute</code>	<i>35, 40, 40, 118, 119</i>
<code>read_decompositions</code>	<i>170</i>
<code>reader</code>	<i>8, 30, 168, 267, 304, 349</i>
<code>reader-&gt;add_special_character</code>	<i>8, 9, 30, 343</i>
<code>reader-&gt;auto_link_email</code>	<i>332</i>
<code>reader-&gt;auto_link_url</code>	<i>332</i>
<code>reader-&gt;create_parser</code>	<i>305</i>
<code>reader-&gt;finalize_grammar</code>	<i>339, 404</i>
<code>reader-&gt;initialize_named_group</code>	<i>343</i>
<code>reader-&gt;insert_pattern</code>	<i>8, 9, 30, 339, 345</i>
<code>reader-&gt;lookup_note_reference</code>	<i>317</i>
<code>reader-&gt;lookup_reference</code>	<i>317</i>
<code>reader-&gt;normalize_tag</code>	<i>305</i>
<code>reader-&gt;options</code>	<i>304</i>
<code>reader-&gt;parser_functions</code>	<i>305</i>
<code>reader-&gt;parser_functions.name</code>	<i>305</i>
<code>reader-&gt;parsers</code>	<i>304, 304</i>
<code>reader-&gt;register_link</code>	<i>317</i>
<code>reader-&gt;update_rule</code>	<i>339, 342, 345</i>
<code>reader-&gt;writer</code>	<i>304</i>
<code>reader.new</code>	<i>304, 304, 404</i>
<code>relativeReferences</code>	<i>41</i>
<code>serialize_byte_parser</code>	<i>169</i>

<code>serialize_byte_range_parser</code>	169
<code>serialize_replacement</code>	169
<code>\setupmarkdown</code>	165, 166
<code>\setupyaml</code>	166
<code>shiftHeadings</code>	7, 41
<code>singletonCache</code>	18
<code>slice</code>	7, 42, 247, 259, 260
<code>smartEllipses</code>	42, 99, 160
<code>\startmarkdown</code>	163, 163, 490
<code>startNumber</code>	42, 114–116
<code>\startyaml</code>	163, 163, 164, 490
<code>\stopmarkdown</code>	163, 163, 490
<code>\stopyaml</code>	163, 163, 164, 490
<code>strikeThrough</code>	43, 123, 484
<code>stripIndent</code>	43, 306
<code>stripPercentSigns</code>	442
<code>subscripts</code>	44, 124
<code>superscripts</code>	44, 124
<code>syntax</code>	340, 345
<code>tableAttributes</code>	44, 125, 480
<code>tableCaptions</code>	7, 44, 45, 125
<code>taskLists</code>	45, 127, 471
<code>texComments</code>	46, 306
<code>texMathDollars</code>	36, 46, 126
<code>texMathDoubleBackslash</code>	36, 47, 126
<code>texMathSingleBackslash</code>	36, 47, 126
<code>tightLists</code>	47, 89, 91, 96, 99, 113, 114, 117, 118, 459
<code>underscores</code>	48
<code>unicode_data.casefold_mapping</code>	181, 192
<code>unicode_data.categories</code>	183, 269
<code>unicode_data.ccc</code>	185, 193
<code>unicode_data.composition_mapping</code>	177, 197
<code>unicode_data.decomposition_mapping</code>	171, 173, 195
<code>unicodeNormalization</code>	18, 19
<code>unicodeNormalizationForm</code>	18, 19
<code>util.cache</code>	187, 187
<code>util.cache_verbatim</code>	187
<code>util.canonically_order</code>	193
<code>util.casefold</code>	192
<code>util.compose</code>	197

util.decompose	195
util.encode_json_string	188
util.err	187
util.escaper	190
util.expand_tabs_in_line	188
util.flatten	189
util.intersperse	189
util.map	190
util.normalize	199
util.pathname	191
util.rope_last	189
util.rope_to_string	189
util.salt	191
util.table_copy	187
util.walk	188, 189
util.warning	192
walkable_syntax	8, 20, 26, 339, 342, 344, 345
writer	168, 168, 246, 349
writer->active_attributes	258, 258, 259, 260
writer->attribute_type_levels	259
writer->attributes	256
writer->block_html_element	254
writer->blockquote	255
writer->bulletitem	253
writer->bulletlist	252
writer->citations	350
writer->code	251
writer->contentblock	355
writer->defer_call	267, 267
writer->definitionlist	357
writer->display_math	390
writer->div_begin	371
writer->div_end	371
writer->document	256
writer->ellipsis	248
writer->emphasis	254
writer->error	251
writer->escape	250
writer->escaped_chars	249, 250
writer->escaped_minimal_strings	249, 250
writer->escaped_strings	249

writer->escaped_uri_chars	249, 250
writer->fancyitem	361
writer->fancylist	360
writer->fencedCode	366
writer->flatten_inlines	246, 246
writer->get_state	266
writer->hard_line_break	248
writer->heading	264
writer->identifier	250
writer->image	252
writer->infostring	250
writer->inline_html_comment	254
writer->inline_html_tag	254
writer->inline_math	390
writer->interblocksep	248
writer->is_writing	247, 247
writer->jekyllData	394
writer->lineblock	377
writer->link	251
writer->mark	378
writer->math	250
writer->nbsp	247
writer->note	381
writer->options	246
writer->ordereditem	254
writer->orderedlist	253
writer->paragraph	248
writer->paragraphsep	248
writer->plain	247
writer->pop_attributes	259, 259, 260
writer->push_attributes	259, 259, 260
writer->rawBlock	366
writer->rawInline	387
writer->set_state	266
writer->slice_begin	247
writer->slice_end	247
writer->soft_line_break	248
writer->space	247
writer->span	349
writer->strike_through	388
writer->string	250
writer->strong	255

writer->subscript	388
writer->superscript	389
writer->table	384
writer->thematic_break	248
writer->checkbox	255
writer->undosep	248, 347
writer->uri	250
writer->verbatim	255
writer->warning	192, 251
writer.new	246, 246, 404
\yaml	156
yaml	154, 156, 450
\yamlBegin	54, 55, 152, 153, 156, 163
\yamlEnd	54, 55, 153, 156, 163
\yamlInput	54, 56, 154, 157, 165, 450
\yamlSetup	58