

A Markdown Interpreter for T_EX

Vít Starý Novotný
witiko@mail.muni.cz

Version 3.5.0-0-gfd01a252
2024-04-29

Contents

1	Introduction	1	3	Implementation	138
1.1	Requirements	2	3.1	Lua Implementation	139
1.2	Feedback	6	3.2	Plain T _E X Implementation	335
1.3	Acknowledgements	6	3.3	L ^A T _E X Implementation	356
2	Interfaces	6	3.4	ConT _E Xt Implementation	386
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	49			
2.3	L ^A T _E X Interface	128			
2.4	ConT _E Xt Interface	135			
				References	395
				Index	396

List of Figures

1	A block diagram of the Markdown package	7
2	A sequence diagram of typesetting a document using the T _E X interface	45
3	A sequence diagram of typesetting a document using the Lua CLI	46
4	Various formats of mathematical formulae	133
5	The banner of the Markdown package	134

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. Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

number of tutorials and code examples. You can find more of these in the user manual.³

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

1.1 Requirements

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

1.1.1 Lua Requirements

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

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

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

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

```
13 local unicode = require("unicode")
```

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

```
14 local md5 = require("md5");
```

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

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

```
15 (function()
```

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

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

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

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

```
23 local uni_algos = require("lua-uni-algos")
```

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

1.1.2 Plain \TeX Requirements

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

expl3 A package that enables the expl3 language from the L \AA \TeX 3 kernel in \TeX Live \leq 2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
24 </tex>
25 <*context>
26 \unprotect
27 </context>
28 <*context, tex>
29 \ifx\ExplSyntaxOn\undefined
30   \input expl3-generic
31 \fi
32 </context, tex>
33 <*tex>
```

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

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

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

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

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

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

1.1.3 L^ATeX Requirements

The L^ATeX part of the package requires that the L^ATeX 2_ε format is loaded,

```
34 \NeedsTeXFormat{LaTeX2e}%
```

a TeX engine that extends ε -TeX, and all the plain TeX prerequisites (see Section 1.1.2):

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or L^ATeX themes (see Section 2.3.3) 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.

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

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

- ifthen** A package that provides a concise syntax for the inspection of macro values. It is used in the [witiko/dot](#) L^AT_EX theme (see Section 2.3.3).
- fancyvrb** A package that provides the `\VerbatimInput` macros for the verbatim inclusion of files containing code.
- csvsimple** A package that provides the `\csvautotabular` macro for typesetting CSV files in the default renderer prototypes for iA Writer content blocks.
- gobble** A package that provides the `\@gobblethree` T_EX command that is used in the default renderer prototype for citations. The package is included in T_EXLive \geq 2016.
- amsmath and amssymb** Packages that provide symbols used for drawing ticked and unticked boxes.
- catchfile** A package that catches the contents of a file and puts it in a macro. It is used in the [witiko/graphicx/http](#) L^AT_EX theme, see Section 2.3.3.
- graphicx** A package that builds upon the graphics package, which is part of the L^AT_EX 2_ε kernel. It provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.
- grffile** A package that extends the name processing of the graphics package to support a larger range of file names in $2006 \leq \text{T_EX Live} \leq 2019$. Since T_EX Live \geq 2020, the functionality of the package has been integrated in the L^AT_EX 2_ε kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#) L^AT_EX themes, see Section 2.3.3.
- etoolbox** A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.8, and also in the default renderer prototype for identifier attributes.
- soulutf8** A package that is used in the default renderer prototype for strike-throughs and marked text.
- ltxcmds** A package that is used to detect whether the minted and listings packages are loaded in the default renderer prototype for fenced code blocks.
- verse** A package that is used in the default renderer prototypes for line blocks.

³⁵ `\RequirePackage{expl3}`

1.1.4 ConT_EXt Prerequisites

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

m-database A module that provides the default token renderer prototype for iA Writer content blocks with the csv filename extension (see Section 2.2.6).

1.2 Feedback

Please use the Markdown project page on GitHub⁴ to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the T_EX-~~L~~A_TE_X Stack Exchange.⁵ community question answering web site under the `markdown` tag.

1.3 Acknowledgements

The Lunamark Lua module provides speedy markdown parsing for the package. I would like to thank John Macfarlane, the creator of Lunamark, for releasing Lunamark under a permissive license, which enabled its use in the Markdown package.

Extensive user documentation for the Markdown package was kindly written by Lian Tze Lim and published by Overleaf.

Funding by the Faculty of Informatics at the Masaryk University in Brno [2] is gratefully acknowledged.

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

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

2 Interfaces

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

Since neither T_EX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of

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

⁵See <https://tex.stackexchange.com>.

structuring this documentation and as a promise to the user that if they only access the package through the interface, the future minor versions of the package should remain backwards compatible.

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

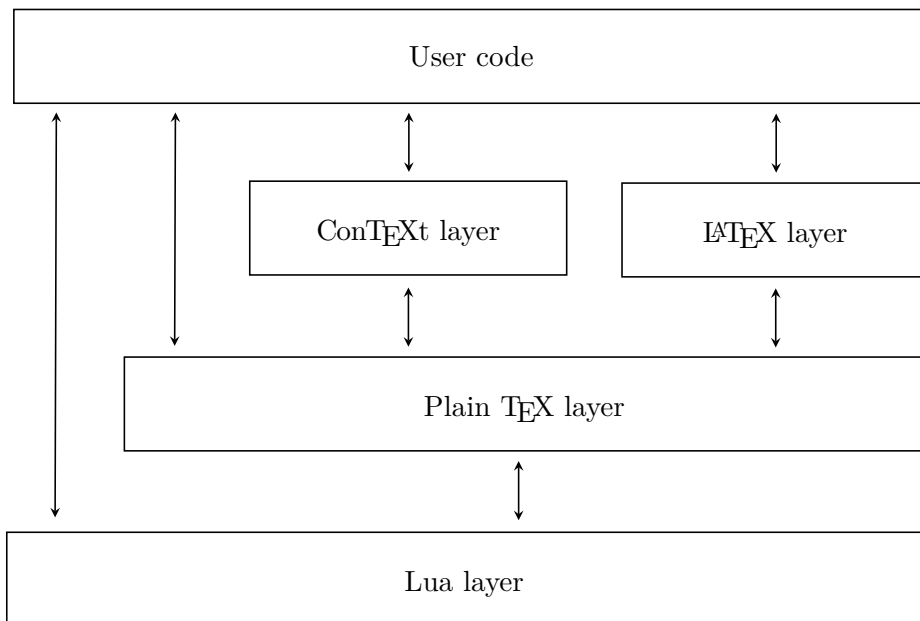


Figure 1: A block diagram of the Markdown package

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.

```
36 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` that contains options recognized by the Lua interface (see Section 2.1.3). The

`options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

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

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

2.1.2 User-Defined Syntax Extensions

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

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

```
37 local walkable_syntax = {
38   Block = {
39     "Blockquote",
40     "Verbatim",
41     "ThematicBreak",
42     "BulletList",
43     "OrderedList",
44     "DisplayHtml",
45     "Heading",
46   },
47   BlockOrParagraph = {
48     "Block",
49     "Paragraph",
50     "Plain",
51   },
52   Inline = {
53     "Str",
54     "Space",
55     "Endline",
56     "EndlineBreak",
57     "LinkAndEmph",
58     "Code",
59     "AutoLinkUrl",
60     "AutoLinkEmail",
61     "AutoLinkRelativeReference",
```



```

62     "InlineHtml",
63     "HtmlEntity",
64     "EscapedChar",
65     "Smart",
66     "Symbol",
67   },
68 }

```

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.

```

69 local defaultOptions = {}

```

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

```

70 \ExplSyntaxOn
71 \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.

```

72 \prop_new:N \g_@@_lua_option_types_prop
73 \prop_new:N \g_@@_default_lua_options_prop
74 \seq_new:N \g_@@_option_layers_seq
75 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
76 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
77 \cs_new:Nn
78   \@@_add_lua_option:nnn
79   {
80     \@@_add_option:Vnnn

```

```

81     \c_@@_option_layer_lua_tl
82     { #1 }
83     { #2 }
84     { #3 }
85   }
86 \cs_new:Nn
87   \@@_add_option:nmmn
88   {
89     \seq_gput_right:cn
90     { g_@@_ #1 _options_seq }
91     { #2 }
92     \prop_gput:cnn
93     { g_@@_ #1 _option_types_prop }
94     { #2 }
95     { #3 }
96     \prop_gput:cnn
97     { g_@@_default_ #1 _options_prop }
98     { #2 }
99     { #4 }
100   \@@_typecheck_option:n
101     { #2 }
102   }
103 \cs_generate_variant:Nn
104   \@@_add_option:nmmn
105   { Vmmn }
106 \tl_const:Nn \c_@@_option_value_true_tl { true }
107 \tl_const:Nn \c_@@_option_value_false_tl { false }
108 \cs_new:Nn \@@_typecheck_option:n
109   {
110     \@@_get_option_type:nN
111     { #1 }
112     \l_tmpa_tl
113     \str_case_e:Vn
114     \l_tmpa_tl
115     {
116       { \c_@@_option_type_boolean_tl }
117       {
118         \@@_get_option_value:nN
119         { #1 }
120         \l_tmpa_tl
121         \bool_if:nF
122         {
123           \str_if_eq_p:VV
124           \l_tmpa_tl
125           \c_@@_option_value_true_tl ||
126           \str_if_eq_p:VV
127           \l_tmpa_tl

```

```

128         \c_@@_option_value_false_tl
129     }
130     {
131         \msg_error:nnnV
132         { markdown }
133         { failed-typecheck-for-boolean-option }
134         { #1 }
135         \l_tmpa_tl
136     }
137 }
138 }
139 }
140 \msg_new:nnn
141 { markdown }
142 { failed-typecheck-for-boolean-option }
143 {
144     Option~#1~has~value~#2,~
145     but~a~boolean~(true~or~false)~was~expected.
146 }
147 \cs_generate_variant:Nn
148   \str_case_e:nn
149   { Vn }
150 \cs_generate_variant:Nn
151   \msg_error:nnnn
152   { nnnV }
153 \seq_new:N \g_@@_option_types_seq
154 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
155 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
156 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
157 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
158 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
159 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
160 \tl_const:Nn \c_@@_option_type_number_tl { number }
161 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
162 \tl_const:Nn \c_@@_option_type_path_tl { path }
163 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
164 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
165 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
166 \tl_const:Nn \c_@@_option_type_string_tl { string }
167 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
168 \cs_new:Nn
169   \@@_get_option_type:nN
170   {
171     \bool_set_false:N
172       \l_tmpa_bool
173     \seq_map_inline:Nn
174       \g_@@_option_layers_seq

```

```

175     {
176     \prop_get:cnNT
177     { g_@@_ ##1 _option_types_prop }
178     { #1 }
179     \l_tmpa_tl
180     {
181     \bool_set_true:N
182     \l_tmpa_bool
183     \seq_map_break:
184     }
185     }
186 \bool_if:nF
187 \l_tmpa_bool
188 {
189 \msg_error:nnn
190 { markdown }
191 { undefined-option }
192 { #1 }
193 }
194 \seq_if_in:NVF
195 \g_@@_option_types_seq
196 \l_tmpa_tl
197 {
198 \msg_error:nnnV
199 { markdown }
200 { unknown-option-type }
201 { #1 }
202 \l_tmpa_tl
203 }
204 \tl_set_eq:NN
205 #2
206 \l_tmpa_tl
207 }
208 \msg_new:nnn
209 { markdown }
210 { unknown-option-type }
211 {
212 Option~#1~has~unknown~type~#2.
213 }
214 \msg_new:nnn
215 { markdown }
216 { undefined-option }
217 {
218 Option~#1~is~undefined.
219 }
220 \cs_new:Nn
221 \@@_get_default_option_value:nN

```

```

222 {
223   \bool_set_false:N
224   \l_tmpa_bool
225   \seq_map_inline:Nn
226   \g_@@_option_layers_seq
227   {
228     \prop_get:cnNT
229     { g_@@_default_ ##1 _options_prop }
230     { #1 }
231     #2
232     {
233       \bool_set_true:N
234       \l_tmpa_bool
235       \seq_map_break:
236     }
237   }
238   \bool_if:nF
239   \l_tmpa_bool
240   {
241     \msg_error:nnn
242     { markdown }
243     { undefined-option }
244     { #1 }
245   }
246 }
247 \cs_new:Nn
248 \@@_get_option_value:nN
249 {
250   \@@_option_tl_to_csname:nN
251   { #1 }
252   \l_tmpa_tl
253   \cs_if_free:cTF
254   { \l_tmpa_tl }
255   {
256     \@@_get_default_option_value:nN
257     { #1 }
258     #2
259   }
260   {
261     \@@_get_option_type:nN
262     { #1 }
263     \l_tmpa_tl
264     \str_if_eq:NNTF
265     \c_@@_option_type_counter_tl
266     \l_tmpa_tl
267     {
268       \@@_option_tl_to_csname:nN

```

```

269         { #1 }
270         \l_tmpa_tl
271         \tl_set:Nx
272         #2
273         { \the \cs:w \l_tmpa_tl \cs_end: }
274     }
275     {
276         \@@_option_tl_to_csname:nN
277         { #1 }
278         \l_tmpa_tl
279         \tl_set:Nv
280         #2
281         { \l_tmpa_tl }
282     }
283 }
284 }
285 \cs_new:Nn \@@_option_tl_to_csname:nN
286 {
287     \tl_set:Nn
288     \l_tmpa_tl
289     { \str_uppercase:n { #1 } }
290     \tl_set:Nx
291     #2
292     {
293         markdownOption
294         \tl_head:f { \l_tmpa_tl }
295         \tl_tail:n { #1 }
296     }
297 }

```

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.

```

298 \cs_new:Nn \@@_with_various_cases:nn
299 {
300     \seq_clear:N
301     \l_tmpa_seq
302     \seq_map_inline:Nn
303     \g_@@_cases_seq
304     {
305         \tl_set:Nn
306         \l_tmpa_tl
307         { #1 }
308         \use:c { ##1 }
309         \l_tmpa_tl
310         \seq_put_right:NV
311         \l_tmpa_seq

```

```

312         \l_tmpa_tl
313     }
314     \seq_map_inline:Nn
315         \l_tmpa_seq
316         { #2 }
317 }

```

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

```

318 \cs_new:Nn \@@_with_various_cases_break:
319 {
320     \seq_map_break:
321 }

```

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

```

322 \seq_new:N \g_@@_cases_seq
323 \cs_new:Nn \@@_camel_case:N
324 {
325     \regex_replace_all:mnN
326         { _ ([a-z]) }
327         { \c { str_uppercase:n } \cB\{ \1 \cE\} }
328         #1
329     \tl_set:Nx
330         #1
331         { #1 }
332 }
333 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
334 \cs_new:Nn \@@_snake_case:N
335 {
336     \regex_replace_all:mnN
337         { ([a-z])([A-Z]) }
338         { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
339         #1
340     \tl_set:Nx
341         #1
342         { #1 }
343 }
344 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 General Behavior

`eagerCache=true, false` default: `false`

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

produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

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

false Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing.

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

```
345 \@@_add_lua_option:nnn
346   { eagerCache }
347   { boolean }
348   { false }

349 defaultOptions.eagerCache = 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 issue #226⁶.

This was the default behavior until version 3.0.0 of the Markdown package.

```
350 \@@_add_lua_option:nnn
351   { singletonCache }
352   { boolean }
353   { true }

354 defaultOptions.singletonCache = true

355 local singletonCache = {
356   convert = nil,
357   options = nil,
358 }
```

⁶See <https://github.com/witiko/markdown/pull/226#issuecomment-1599641634>.

2.1.5 File and Directory Names

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.1.6 Parser Options

`autoIdentifiers`=true, false default: false

true Enable the Pandoc auto identifiers syntax extension⁷:

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

```
# 123 Sesame Street
```

false Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```
379 \@@_add_lua_option:nnn
380   { autoIdentifiers }
381   { boolean }
382   { false }
383 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.

⁷See https://pandoc.org/MANUAL.html#extension-auto_identifiers.

```

384 \@@_add_lua_option:nnn
385   { blankBeforeBlockquote }
386   { boolean }
387   { false }

388 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.

```

389 \@@_add_lua_option:nnn
390   { blankBeforeCodeFence }
391   { boolean }
392   { false }

393 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.

```

394 \@@_add_lua_option:nnn
395   { blankBeforeDivFence }
396   { boolean }
397   { false }

398 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.

```

399 \@@_add_lua_option:nnn
400   { blankBeforeHeading }
401   { boolean }
402   { false }

403 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.

```
404 \@@_add_lua_option:nnn
405   { blankBeforeList }
406   { boolean }
407   { false }

408 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

- `true` Enable the Pandoc bracketed span syntax extension⁸:

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

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

```
409 \@@_add_lua_option:nnn
410   { bracketedSpans }
411   { boolean }
412   { false }

413 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false` default: true

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

```
414 \@@_add_lua_option:nnn
415   { breakableBlockquotes }
416   { boolean }
417   { true }

418 defaultOptions.breakableBlockquotes = true
```

⁸See https://pandoc.org/MANUAL.html#extension-bracketed_spans.

`citationNbsps=true, false`

default: `false`

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

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

```
419 \@@_add_lua_option:nnn
420 { citationNbsps }
421 { boolean }
422 { true }

423 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: `false`

`true` Enable the Pandoc citation syntax extension⁹:

Here is a simple parenthetical citation [`@doe99`] and here is a string of several [`see @doe99, pp. 33-35; also @smith04, chap. 1`].

A parenthetical citation can have a [`prenote @doe99`] and a [`@smith04 postnote`]. The name of the author can be suppressed by inserting a dash before the name of an author as follows [`-@smith04`].

Here is a simple text citation `@doe99` and here is a string of several `@doe99` [`pp. 33-35; also @smith04, chap. 1`]. Here is one with the name of the author suppressed `-@doe99`.

`false` Disable the Pandoc citation syntax extension.

```
424 \@@_add_lua_option:nnn
425 { citations }
426 { boolean }
427 { false }

428 defaultOptions.citations = false
```

⁹See <https://pandoc.org/MANUAL.html#extension-citations>.

`codeSpans=true, false`

default: true

true Enable the code span syntax:

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

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

```
``This is a quote.``
```

```
429 \@@_add_lua_option:nnn  
430 { codeSpans }  
431 { boolean }  
432 { true }  
  
433 defaultOptions.codeSpans = true
```

`contentBlocks=true, false`

default: false

true

: Enable the iA Writer content blocks syntax extension [3]:

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

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

```
434 \@@_add_lua_option:nnn
435 { contentBlocks }
436 { boolean }
437 { false }

438 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
```

```
439 \@@_add_lua_option:nnn
440 { contentLevel }
441 { string }
442 { block }
443 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.

```
444 \@@_add_lua_option:nnn
445 { debugExtensions }
446 { boolean }
447 { false }
448 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.

```

449 \@@_add_lua_option:nnn
450 { definitionLists }
451 { boolean }
452 { false }

453 defaultOptions.definitionLists = false

```

**expectJekyllData=true, false**

default: false

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

```

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

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

```



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

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

```
454 \@@_add_lua_option:nnn
455 { expectJekyllData }
456 { boolean }
457 { false }
458 defaultOptions.expectJekyllData = false
```

`extensions=<filenames>`

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

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

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

```

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

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

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

return strike_through

```

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

```

459 metadata.user_extension_api_version = 2
460 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.

```

461 \cs_generate_variant:Nn
462 \@@_add_lua_option:nnn
463 { nnV }
464 \@@_add_lua_option:nnV
465 { extensions }
466 { clist }
467 \c_empty_clist
468 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

**true** Enable the Pandoc fancy list syntax extension<sup>10</sup>:

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

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

```
469 \@@_add_lua_option:nnn
470 { fancyLists }
471 { boolean }
472 { false }
473 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: true

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

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

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

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

```
474 \@@_add_lua_option:nnn
475 { fencedCode }
476 { boolean }
477 { true }
478 defaultOptions.fencedCode = true
```

<sup>10</sup>See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

`fencedCodeAttributes=true, false`

default: false

**true** Enable the Pandoc fenced code attribute syntax extension<sup>11</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.

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

483 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

**true** Enable the Pandoc fenced div syntax extension<sup>12</sup>:

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

And another.
:::::
```

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

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

488 defaultOptions.fencedDivs = false
```

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

<sup>12</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

`finalizeCache=true, false`

default: `false`

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

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

```
489 \@@_add_lua_option:nnn
490 { finalizeCache }
491 { boolean }
492 { false }

493 defaultOptions.finalizeCache = false
```

`frozenCacheCounter=<number>`

default: `0`

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

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

```
494 \@@_add_lua_option:nnn
495 { frozenCacheCounter }
496 { counter }
497 { 0 }

498 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: `false`

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>13</sup>:

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

123 Sesame Street
```

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

---

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

See also the option [autoIdentifiers](#).

```
499 \@@_add_lua_option:nnn
500 { gfmAutoIdentifiers }
501 { boolean }
502 { false }

503 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.

```
504 \@@_add_lua_option:nnn
505 { hashEnumerators }
506 { boolean }
507 { false }

508 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.

```
509 \@@_add_lua_option:nnn
510 { headerAttributes }
511 { boolean }
512 { false }

513 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.

```
514 \@@_add_lua_option:nnn
515 { html }
516 { boolean }
517 { true }

518 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.

```
519 \@@_add_lua_option:nnn
520 { hybrid }
521 { boolean }
522 { false }

523 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false` default: false

- true** Enable the Pandoc inline code span attribute extension<sup>14</sup>:

``<$>`{.haskell}`

---

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

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

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

528 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: false

`true` Enable the Pandoc inline note syntax extension<sup>15</sup>:

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

`false` Disable the Pandoc inline note syntax extension.

```
529 \@@_add_lua_option:nnn
530 { inlineNotes }
531 { boolean }
532 { false }

533 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>16</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.

```

<sup>15</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).



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

```
534 \@@_add_lua_option:nnn
535 { jekyllData }
536 { boolean }
537 { false }
538 defaultOptions.jekyllData = false
```

**linkAttributes=true, false** default: false

**true** Enable the Pandoc link and image attribute syntax extension<sup>17</sup>:

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

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

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

```
539 \@@_add_lua_option:nnn
540 { linkAttributes }
541 { boolean }
542 { false }
543 defaultOptions.linkAttributes = false
```

**lineBlocks=true, false** default: false

**true** Enable the Pandoc line block syntax extension<sup>18</sup>:

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

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

```
544 \@@_add_lua_option:nnn
545 { lineBlocks }
546 { boolean }
547 { false }
548 defaultOptions.lineBlocks = false
```

---

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

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

`mark=true, false` default: false

`true` Enable the Pandoc mark syntax extension<sup>19</sup>:

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

`false` Disable the Pandoc mark syntax extension.

```
549 \@@_add_lua_option:nnn
550 { mark }
551 { boolean }
552 { false }
553 defaultOptions.mark = false
```

`notes=true, false` default: false

`true` Enable the Pandoc note syntax extension<sup>20</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.

```
554 \@@_add_lua_option:nnn
555 { notes }
556 { boolean }
557 { false }
558 defaultOptions.notes = false
```

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

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

`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.

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

563 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.

```
564 \@@_add_lua_option:nnn
565 { preserveTabs }
566 { boolean }
567 { true }

568 defaultOptions.preserveTabs = true
```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>21</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 \\
```

²¹See https://pandoc.org/MANUAL.html#extension-raw_attribute.

```

      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.

```

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

573 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false`

default: `false`

`true` Enable relative references²² 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.

```

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

578 defaultOptions.relativeReferences = false

```

²²See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

`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.

```
579 \@@_add_lua_option:nnn
580   { shiftHeadings }
581   { number }
582   { 0 }

583 defaultOptions.shiftHeadings = 0
```

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

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

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

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

```
584 \@@_add_lua_option:nnn
585   { slice }
586   { slice }
587   { ^-$ }

588 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.

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

593 defaultOptions.smartEllipses = false
```

`startNumber=true, false` default: true

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

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

```
594 \@@_add_lua_option:nnn
595 { startNumber }
596 { boolean }
597 { true }

598 defaultOptions.startNumber = true
```

`strikeThrough=true, false` default: false

`true` Enable the Pandoc strike-through syntax extension²³:

`This is deleted text.`

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

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

603 defaultOptions.strikeThrough = false
```

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

`stripIndent=true, false`

default: `false`

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

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

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

```
604 \@@_add_lua_option:nnn
605   { stripIndent }
606   { boolean }
607   { false }
608 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

`true` Enable the Pandoc subscript syntax extension²⁴:

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

`false` Disable the Pandoc subscript syntax extension.

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

²⁴See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`superscripts=true, false`

default: false

true Enable the Pandoc superscript syntax extension²⁵:

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

false Disable the Pandoc superscript syntax extension.

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

618 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}
```
```

false Disable the assignment of HTML attributes to table captions.

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

623 defaultOptions.tableAttributes = false
```

²⁵See <https://pandoc.org/MANUAL.html#extension-superscript-subscript>.

`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension²⁶ 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.

```
624 \@@_add_lua_option:nnn
625 { tableCaptions }
626 { boolean }
627 { false }

628 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension<sup>27</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.

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

633 defaultOptions.taskLists = false
```

<sup>26</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>27</sup>See [https://pandoc.org/MANUAL.html#extension-task\\_lists](https://pandoc.org/MANUAL.html#extension-task_lists).

`texComments=true, false`

default: false

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

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

Always enabled when `hybrid` is enabled.

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

```
634 \@@_add_lua_option:nnn
635 { texComments }
636 { boolean }
637 { false }
638 defaultOptions.texComments = false
```

`texMathDollars=true, false`

default: false

**true** Enable the Pandoc dollar math syntax extension<sup>28</sup>:

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

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

```
639 \@@_add_lua_option:nnn
640 { texMathDollars }
641 { boolean }
642 { false }
643 defaultOptions.texMathDollars = false
```

---

<sup>28</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>29</sup>:

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

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

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

648 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

```
649 \@@_add_lua_option:nnn
650 { texMathSingleBackslash }
651 { boolean }
652 { false }

653 defaultOptions.texMathSingleBackslash = false
```

`tightLists=true, false` default: true

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

---

<sup>29</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>30</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.

```

654 \@@_add_lua_option:nnn
655 { tightLists }
656 { boolean }
657 { true }

658 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.

```

659 \@@_add_lua_option:nnn
660 { underscores }
661 { boolean }
662 { true }
663 \ExplSyntaxOff

664 defaultOptions.underscores = true

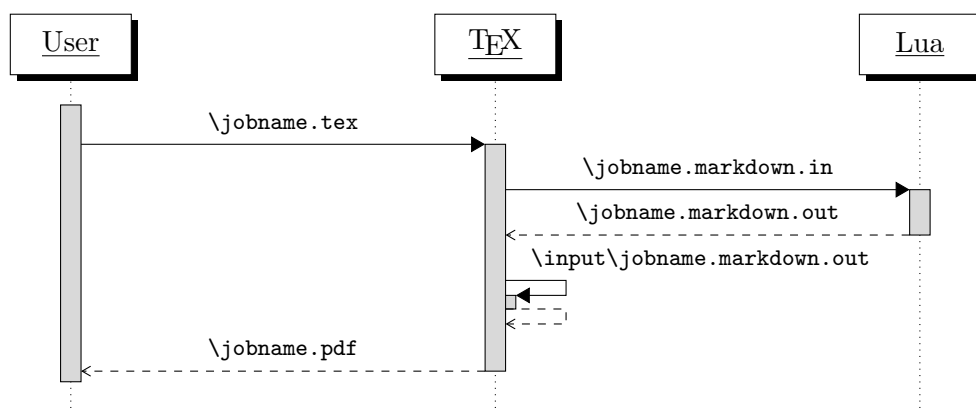
```

### 2.1.7 Command-Line Interface

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

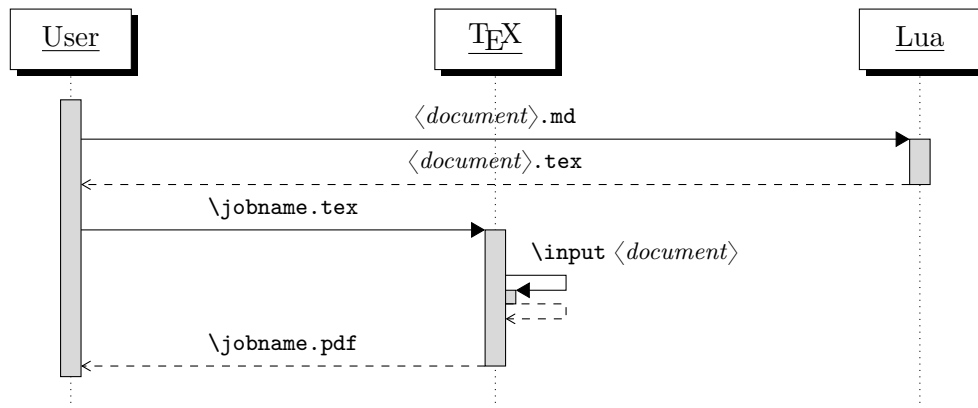
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{T}_{\text{E}}\text{X}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{T}_{\text{E}}\text{X}$  engine includes a Lua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

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



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

```
665
666 local HELP_STRING = [[
667 Usage: texlua]] .. arg[0] .. [[[OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
668 where OPTIONS are documented in the Lua interface section of the
669 technical Markdown package documentation.
670
671 When OUTPUT_FILE is unspecified, the result of the conversion will be
672 written to the standard output. When INPUT_FILE is also unspecified, the
673 result of the conversion will be read from the standard input.
674
675 Report bugs to: witiko@mail.muni.cz
676 Markdown package home page: <https://github.com/witiko/markdown>]]
677
```



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

```

678 local VERSION_STRING = [[
679 markdown-cli.lua (Markdown)]] .. metadata.version .. [[
680
681 Copyright (C)]] .. table.concat(metadata.copyright,
682 "\nCopyright (C) ") .. [[
683
684 License:]] .. metadata.license
685
686 local function warn(s)
687 io.stderr:write("Warning: " .. s .. "\n") end
688
689 local function error(s)
690 io.stderr:write("Error: " .. s .. "\n")
691 os.exit(1)
692 end

```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept snake\_case in addition to camel-Case variants of options. As a bonus, studies [5] also show that snake\_case is faster to read than camelCase.

```

693 local function camel_case(option_name)
694 local cased_option_name = option_name:gsub("_(%l)", function(match)
695 return match:sub(2, 2):upper()
696 end)
697 return cased_option_name
698 end
699
700 local function snake_case(option_name)
701 local cased_option_name = option_name:gsub("%l%u", function(match)
702 return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()

```

```

703 end)
704 return cased_option_name
705 end
706
707 local cases = {camel_case, snake_case}
708 local various_case_options = {}
709 for option_name, _ in pairs(defaultOptions) do
710 for _, case in ipairs(cases) do
711 various_case_options[case(option_name)] = option_name
712 end
713 end
714
715 local process_options = true
716 local options = {}
717 local input_filename
718 local output_filename
719 for i = 1, #arg do
720 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.

```

721 if arg[i] == "--" then
722 process_options = false
723 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.

```

724 elseif arg[i]:match("=") then
725 local key, value = arg[i]:match("(.)=(.*)")
726 if defaultOptions[key] == nil and
727 various_case_options[key] ~= nil then
728 key = various_case_options[key]
729 end

```

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

```

730 local default_type = type(defaultOptions[key])
731 if default_type == "boolean" then
732 options[key] = (value == "true")
733 elseif default_type == "number" then
734 options[key] = tonumber(value)
735 elseif default_type == "table" then
736 options[key] = {}
737 for item in value:gmatch("[^ ,]+") do
738 table.insert(options[key], item)

```

```

739 end
740 else
741 if default_type ~= "string" then
742 if default_type == "nil" then
743 warn('Option "' .. key .. '" not recognized.')
744 else
745 warn('Option "' .. key .. '" type not recognized, please file ' ..
746 'a report to the package maintainer.')
747 end
748 warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
749 key .. '" as a string.')
750 end
751 options[key] = value
752 end
753 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.

```

754 elseif arg[i] == "--help" or arg[i] == "-h" then
755 print(HELP_STRING)
756 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.

```

757 elseif arg[i] == "--version" or arg[i] == "-v" then
758 print(VERSION_STRING)
759 os.exit()
760 end
761 end

```

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

```

762 if input_filename == nil then
763 input_filename = arg[i]

```

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

```

764 elseif output_filename == nil then
765 output_filename = arg[i]
766 else
767 error('Unexpected argument: "' .. arg[i] .. "'.')
768 end
769 ::continue::
770 end

```



The command-line Lua interface is implemented by the `markdown-cli.lua` file that can be invoked from the command line as follows:

```
texlua /path/to/markdown-cli.lua cacheDir=. -- hello.md hello.tex
```

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

```
\input hello
```

## 2.2 Plain TeX Interface

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

```
771 \def\markdownLastModified{((LASTMODIFIED))}%
772 \def\markdownVersion{((VERSION))}%
```

The plain TeX 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

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

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

```
773 \let\markdownBegin\relax
774 \let\markdownEnd\relax
```

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

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [6, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```
775 \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.

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

```
776 \let\markdownInput\relax
```

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

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

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

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.

```
777 \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.

```
778 \ExplSyntaxOn
779 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
780 \cs_generate_variant:Nn
781 \tl_const:Nn
782 { NV }
783 \tl_if_exist:NF
784 \c_@@_top_layer_tl
785 {
```

```

786 \tl_const:NV
787 \c_@@_top_layer_tl
788 \c_@@_option_layer_plain_tex_tl
789 }

```

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

```

790 \seq_new:N \g_@@_plain_tex_options_seq

```

To enable the reflection of default plain T<sub>E</sub>X options and their types, we will maintain the `\g_@@_default_plain_tex_options_prop` and `\g_@@_plain_tex_option_types_prop` property lists, respectively.

```

791 \prop_new:N \g_@@_plain_tex_option_types_prop
792 \prop_new:N \g_@@_default_plain_tex_options_prop
793 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
794 \cs_new:Nn
795 \@@_add_plain_tex_option:nnn
796 {
797 \@@_add_option:Vnnn
798 \c_@@_option_layer_plain_tex_tl
799 { #1 }
800 { #2 }
801 { #3 }
802 }

```

The plain T<sub>E</sub>X options may be also be specified via the `\markdownSetup` macro. Here, the plain T<sub>E</sub>X 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.

```

803 \cs_new:Nn
804 \@@_setup:n
805 {
806 \keys_set:nn
807 { markdown/options }
808 { #1 }
809 }
810 \cs_gset_eq:NN
811 \markdownSetup
812 \@@_setup:n

```

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.

```

813 \prg_new_conditional:Nnn
814 \@@_if_option:n
815 { TF, T, F }
816 {

```

```

817 \@@_get_option_type:nN
818 { #1 }
819 \l_tmpa_tl
820 \str_if_eq:NNF
821 \l_tmpa_tl
822 \c_@@_option_type_boolean_tl
823 {
824 \msg_error:nxxx
825 { markdown }
826 { expected-boolean-option }
827 { #1 }
828 { \l_tmpa_tl }
829 }
830 \@@_get_option_value:nN
831 { #1 }
832 \l_tmpa_tl
833 \str_if_eq:NNTF
834 \l_tmpa_tl
835 \c_@@_option_value_true_tl
836 { \prg_return_true: }
837 { \prg_return_false: }
838 }
839 \msg_new:nnn
840 { markdown }
841 { expected-boolean-option }
842 {
843 Option~#1~has~type~#2,~
844 but~a~boolean~was~expected.
845 }
846 \let\markdownIfOption=\@@_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`.

```

847 \@@_add_plain_tex_option:nnn
848 { frozenCache }
849 { boolean }
850 { false }

```

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

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

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

The expansion of this macro must not contain quotation marks (") or backslash symbols (\). Mind that  $\TeX$  engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
851 \@@_add_plain_tex_option:nnn
852 { inputTempFileName }
853 { path }
854 { \jobname.markdown.in }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain  $\TeX$  implementation. The option defaults to `.` or, since  $\TeX$  Live 2024, to the value of the `-output-directory` option of your  $\TeX$  engine.

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.

The `\markdownOptionOutputDir` macro has been deprecated and will be removed in the next major version of the Markdown package.

```
855 \cs_generate_variant:Nn
856 \@@_add_plain_tex_option:nnn
857 { nnV }
```

Use the `lt3luabridge` library to determine the default value of the `\markdownOptionOutputDir` macro by using the environmental variable `TEXMF_OUTPUT_DIRECTORY` that is available since  $\TeX$  Live 2024.

```
858 \ExplSyntaxOff
859 \input lt3luabridge.tex
860 \ExplSyntaxOn
861 \bool_if:nTF
862 {
863 \cs_if_exist_p:N
864 \luabridge_tl_set:Nn &&
```

```

865 (
866 \int_compare_p:nNn
867 { \g_luabridge_method_int }
868 =
869 { \c_luabridge_method_directlua_int } ||
870 \sys_if_shell_unrestricted_p:
871)
872 }
873 {
874 \luabridge_tl_set:Nn
875 \l_tmpa_tl
876 { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
877 }
878 {
879 \tl_set:Nn
880 \l_tmpa_tl
881 { . }
882 }
883 \@@_add_plain_tex_option:nnV
884 { outputDir }
885 { path }
886 \l_tmpa_tl

```

### 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 [sec:#themes](#)). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

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

```
887 \@@_add_plain_tex_option:nnn
888 { plain }
889 { boolean }
890 { 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.

```
891 \@@_add_plain_tex_option:nnn
892 { noDefaults }
893 { boolean }
894 { 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 [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
895 \seq_gput_right:Nn
896 \g_@@_plain_tex_options_seq
897 { stripPercentSigns }
898 \prop_gput:Nnn
899 \g_@@_plain_tex_option_types_prop
```



```

900 { stripPercentSigns }
901 { boolean }
902 \prop_gput:Nnx
903 \g_@@_default_plain_tex_options_prop
904 { stripPercentSigns }
905 { 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.

```

906 \cs_new:Nn
907 \@@_define_option_commands_and_keyvals:
908 {
909 \seq_map_inline:Nn
910 \g_@@_option_layers_seq
911 {
912 \seq_map_inline:cn
913 { g_@@_ ##1 _options_seq }
914 {
915 \@@_define_option_command:n
916 { #####1 }

```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-Case variants of options. As a bonus, studies [5] also show that `snake_case` is faster to read than `camelCase`.

```

917 \@@_with_various_cases:nn
918 { #####1 }
919 {
920 \@@_define_option_keyval:nnn
921 { ##1 }
922 { #####1 }
923 { #####1 }
924 }
925 }

```

```

926 }
927 }
928 \cs_new:Nn
929 \@@_define_option_command:n
930 {

```

Do not override options defined before loading the package.

```

931 \@@_option_tl_to_csname:nN
932 { #1 }
933 \l_tmpa_tl
934 \cs_if_exist:cF
935 { \l_tmpa_tl }
936 {
937 \@@_get_default_option_value:nN
938 { #1 }
939 \l_tmpa_tl
940 \@@_set_option_value:nV
941 { #1 }
942 \l_tmpa_tl
943 }
944 }
945 \cs_new:Nn
946 \@@_set_option_value:nn
947 {
948 \@@_define_option:n
949 { #1 }
950 \@@_get_option_type:nN
951 { #1 }
952 \l_tmpa_tl
953 \str_if_eq:NNTF
954 \c_@@_option_type_counter_tl
955 \l_tmpa_tl
956 {
957 \@@_option_tl_to_csname:nN
958 { #1 }
959 \l_tmpa_tl
960 \int_gset:cn
961 { \l_tmpa_tl }
962 { #2 }
963 }
964 {
965 \@@_option_tl_to_csname:nN
966 { #1 }
967 \l_tmpa_tl
968 \cs_set:cpn
969 { \l_tmpa_tl }
970 { #2 }
971 }

```

```

972 }
973 \cs_generate_variant:Nn
974 \@@_set_option_value:nn
975 { nV }
976 \cs_new:Nn
977 \@@_define_option:n
978 {
979 \@@_option_tl_to_csname:nN
980 { #1 }
981 \l_tmpa_tl
982 \cs_if_free:cT
983 { \l_tmpa_tl }
984 {
985 \@@_get_option_type:nN
986 { #1 }
987 \l_tmpb_tl
988 \str_if_eq:NNT
989 \c_@@_option_type_counter_tl
990 \l_tmpb_tl
991 {
992 \@@_option_tl_to_csname:nN
993 { #1 }
994 \l_tmpa_tl
995 \int_new:c
996 { \l_tmpa_tl }
997 }
998 }
999 }
1000 \cs_new:Nn
1001 \@@_define_option_keyval:nnn
1002 {
1003 \prop_get:cnN
1004 { g_@@_ #1 _option_types_prop }
1005 { #2 }
1006 \l_tmpa_tl
1007 \str_if_eq:VVTF
1008 \l_tmpa_tl
1009 \c_@@_option_type_boolean_tl
1010 {
1011 \keys_define:nn
1012 { markdown/options }
1013 {

```

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

```

1014 #3 .code:n = {
1015 \tl_set:Nx

```

```

1016 \l_tmpa_tl
1017 {
1018 \str_case:nnF
1019 { ##1 }
1020 {
1021 { yes } { true }
1022 { no } { false }
1023 }
1024 { ##1 }
1025 }
1026 \@@_set_option_value:nV
1027 { #2 }
1028 \l_tmpa_tl
1029 },
1030 #3 .default:n = { true },
1031 }
1032 }
1033 {
1034 \keys_define:nn
1035 { markdown/options }
1036 {
1037 #3 .code:n = {
1038 \@@_set_option_value:nn
1039 { #2 }
1040 { ##1 }
1041 },
1042 }
1043 }

```

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.

```

1044 \str_if_eq:VVT
1045 \l_tmpa_tl
1046 \c_@@_option_type_clist_tl
1047 {
1048 \tl_set:Nn
1049 \l_tmpa_tl
1050 { #3 }
1051 \tl_reverse:N
1052 \l_tmpa_tl
1053 \str_if_eq:enF
1054 {
1055 \tl_head:V
1056 \l_tmpa_tl

```

```

1057 }
1058 { s }
1059 {
1060 \msg_error:nnn
1061 { markdown }
1062 { malformed-name-for-clist-option }
1063 { #3 }
1064 }
1065 \tl_set:Nx
1066 \l_tmpa_tl
1067 {
1068 \tl_tail:V
1069 \l_tmpa_tl
1070 }
1071 \tl_reverse:N
1072 \l_tmpa_tl
1073 \tl_put_right:Nn
1074 \l_tmpa_tl
1075 {
1076 .code:n = {
1077 \@@_get_option_value:nN
1078 { #2 }
1079 \l_tmpa_tl
1080 \clist_set:NV
1081 \l_tmpa_clist
1082 { \l_tmpa_tl, { ##1 } }
1083 \@@_set_option_value:nV
1084 { #2 }
1085 \l_tmpa_clist
1086 }
1087 }
1088 \keys_define:nV
1089 { markdown/options }
1090 \l_tmpa_tl
1091 }
1092 }
1093 \cs_generate_variant:Nn
1094 \clist_set:Nn
1095 { NV }
1096 \cs_generate_variant:Nn
1097 \keys_define:nn
1098 { nV }
1099 \cs_generate_variant:Nn
1100 \@@_set_option_value:nn
1101 { nV }
1102 \prg_generate_conditional_variant:Nnn
1103 \str_if_eq:nn

```

```

1104 { en }
1105 { F }
1106 \msg_new:nnn
1107 { markdown }
1108 { malformed-name-for-clist-option }
1109 {
1110 Clist-option-name~#1~does~not~end~with~-s.
1111 }

```

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.

```

1112 \str_if_eq:VVT
1113 \c_@@_top_layer_tl
1114 \c_@@_option_layer_plain_tex_tl
1115 {
1116 \@@_define_option_commands_and_keyvals:
1117 }
1118 \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>` 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 is *qualified* and contains no underscores. A theme name is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

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

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

```

1119 \ExplSyntaxOn
1120 \keys_define:nn
1121 { markdown/options }

```

```

1122 {
1123 theme .code:n = {
1124 \@@_set_theme:n
1125 { #1 }
1126 },
1127 import .code:n = {
1128 \tl_set:Nn
1129 \l_tmpa_tl
1130 { #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.

```

1131 \tl_replace_all:NnV
1132 \l_tmpa_tl
1133 { / }
1134 \c_backslash_str
1135 \keys_set:nV
1136 { markdown/options/import }
1137 \l_tmpa_tl
1138 },
1139 }

```

To keep track of the current theme when themes are nested, we will maintain the `\g_@@_themes_seq` stack of theme names. For convenience, the name of the current theme is also available in the `\g_@@_current_theme_tl` macro.

```

1140 \seq_new:N
1141 \g_@@_themes_seq
1142 \tl_new:N
1143 \g_@@_current_theme_tl
1144 \tl_gset:Nn
1145 \g_@@_current_theme_tl
1146 { }
1147 \seq_gput_right:NV
1148 \g_@@_themes_seq
1149 \g_@@_current_theme_tl
1150 \cs_new:Nn
1151 \@@_set_theme:n
1152 {

```

First, we validate the theme name.

```

1153 \str_if_in:nnF
1154 { #1 }
1155 { / }
1156 {
1157 \msg_error:nnn

```

```

1158 { markdown }
1159 { unqualified-theme-name }
1160 { #1 }
1161 }
1162 \str_if_in:nnT
1163 { #1 }
1164 { _ }
1165 {
1166 \msg_error:nnn
1167 { markdown }
1168 { underscores-in-theme-name }
1169 { #1 }
1170 }

```

Next, we munge the theme name.

```

1171 \str_set:Nn
1172 \l_tmpa_str
1173 { #1 }
1174 \str_replace_all:Nnn
1175 \l_tmpa_str
1176 { / }
1177 { _ }

```

Finally, we load the theme.

```

1178 \tl_gset:Nn
1179 \g_@@_current_theme_tl
1180 { #1 / }
1181 \seq_gput_right:NV
1182 \g_@@_themes_seq
1183 \g_@@_current_theme_tl
1184 \@@_load_theme:nV
1185 { #1 }
1186 \l_tmpa_str
1187 \seq_gpop_right:NN
1188 \g_@@_themes_seq
1189 \l_tmpa_tl
1190 \seq_get_right:NN
1191 \g_@@_themes_seq
1192 \l_tmpa_tl
1193 \tl_gset:NV
1194 \g_@@_current_theme_tl
1195 \l_tmpa_tl
1196 }
1197 \msg_new:nnnn
1198 { markdown }
1199 { unqualified-theme-name }
1200 { Won't-load-theme-with-unqualified-name-#1 }
1201 { Theme-names-must-contain-at-least-one-forward-slash }

```



```

1202 \msg_new:nnnn
1203 { markdown }
1204 { underscores-in-theme-name }
1205 { Won't-load-theme-with-an-underscore-in-its-name~#1 }
1206 { Theme-names-must-not-contain-underscores-in-their-names }
1207 \cs_generate_variant:Nn
1208 \tl_replace_all:Nnn
1209 { NnV }
1210 \ExplSyntaxOff

```

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

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

```

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

```

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

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

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.

```

1211 \ExplSyntaxOn
1212 \prop_new:N
1213 \g_@@_snippets_prop
1214 \cs_new:Nn
1215 \@@_setup_snippet:nn
1216 {
1217 \tl_if_empty:nT
1218 { #1 }
1219 {

```

```

1220 \msg_error:nnn
1221 { markdown }
1222 { empty-snippet-name }
1223 { #1 }
1224 }
1225 \tl_set:NV
1226 \l_tmpa_tl
1227 \g_@@_current_theme_tl
1228 \tl_put_right:Nn
1229 \l_tmpa_tl
1230 { #1 }
1231 \@@_if_snippet_exists:nT
1232 { #1 }
1233 {
1234 \msg_warning:nnV
1235 { markdown }
1236 { redefined-snippet }
1237 \l_tmpa_tl
1238 }
1239 \prop_gput:NVn
1240 \g_@@_snippets_prop
1241 \l_tmpa_tl
1242 { #2 }
1243 }
1244 \cs_gset_eq:NN
1245 \markdownSetupSnippet
1246 \@@_setup_snippet:nn
1247 \msg_new:nnnn
1248 { markdown }
1249 { empty-snippet-name }
1250 { Empty-snippet-name~#1 }
1251 { Pick~a~non-empty~name~for~your~snippet }
1252 \msg_new:nnn
1253 { markdown }
1254 { redefined-snippet }
1255 { Redefined~snippet~#1 }

```

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

```

1256 \prg_new_conditional:Nnn
1257 \@@_if_snippet_exists:n
1258 { TF, T, F }
1259 {
1260 \tl_set:NV
1261 \l_tmpa_tl
1262 \g_@@_current_theme_tl
1263 \tl_put_right:Nn
1264 \l_tmpa_tl

```

```

1265 { #1 }
1266 \prop_get:NVNTF
1267 \g_@@_snippets_prop
1268 \l_tmpa_tl
1269 \l_tmpb_tl
1270 { \prg_return_true: }
1271 { \prg_return_false: }
1272 }
1273 \cs_gset_eq:NN
1274 \markdownIfSnippetExists
1275 \@@_if_snippet_exists:nTF

```

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

```

1276 \keys_define:nn
1277 { markdown/options }
1278 {
1279 snippet .code:n = {
1280 \tl_set:NV
1281 \l_tmpa_tl
1282 \g_@@_current_theme_tl
1283 \tl_put_right:Nn
1284 \l_tmpa_tl
1285 { #1 }
1286 \@@_if_snippet_exists:nTF
1287 { #1 }
1288 {
1289 \prop_get:NVN
1290 \g_@@_snippets_prop
1291 \l_tmpa_tl
1292 \l_tmpb_tl
1293 \@@_setup:V
1294 \l_tmpb_tl
1295 }
1296 {
1297 \msg_error:nnV
1298 { markdown }
1299 { undefined-snippet }
1300 \l_tmpa_tl
1301 }
1302 }
1303 }
1304 \msg_new:nnn
1305 { markdown }
1306 { undefined-snippet }
1307 { Can't~invoke~undefined~snippet~#1 }
1308 \cs_generate_variant:Nn
1309 \@@_setup:n
1310 { V }

```

1311 \ExplSyntaxOff

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

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

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

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

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

Several themes and/or snippets can be loaded at once using the extended variant of the `import`  $\LaTeX$  option:

```

\markdownSetup{
 import = {
 jdoe/longpackagename/lists = {
 arabic as arabic1,

```

```

 roman,
 alphabetic,
 },
 jdoe/anotherlongpackagename/lists = {
 arabic as arabic2,
 },
 jdoe/yetanotherlongpackagename,
},
}

```

```

1312 \ExplSyntaxOn
1313 \tl_new:N
1314 \l_@@_import_current_theme_tl
1315 \keys_define:nn
1316 { markdown/options/import }
1317 {

```

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

```

1318 unknown .default:n = {},
1319 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.

```

1320 \tl_set_eq:NN
1321 \l_@@_import_current_theme_tl
1322 \l_keys_key_str
1323 \tl_replace_all:NvN
1324 \l_@@_import_current_theme_tl
1325 \c_backslash_str
1326 { / }

```

Here, we import the snippets.

```

1327 \clist_map_inline:nn
1328 { #1 }
1329 {
1330 \regex_extract_once:nnNTF
1331 { ^(.*)\s+as\s+(.*)$ }
1332 { ##1 }
1333 \l_tmpa_seq
1334 {
1335 \seq_pop:NN
1336 \l_tmpa_seq

```

```

1337 \l_tmpa_tl
1338 \seq_pop:NN
1339 \l_tmpa_seq
1340 \l_tmpa_tl
1341 \seq_pop:NN
1342 \l_tmpa_seq
1343 \l_tmpb_tl
1344 }
1345 {
1346 \tl_set:Nn
1347 \l_tmpa_tl
1348 { ##1 }
1349 \tl_set:Nn
1350 \l_tmpb_tl
1351 { ##1 }
1352 }
1353 \tl_put_left:Nn
1354 \l_tmpa_tl
1355 { / }
1356 \tl_put_left:NV
1357 \l_tmpa_tl
1358 \l_@@_import_current_theme_tl
1359 \@@_setup_snippet:Vx
1360 \l_tmpb_tl
1361 { snippet = { \l_tmpa_tl } }
1362 }

```

Here, we load the theme.

```

1363 \@@_set_theme:V
1364 \l_@@_import_current_theme_tl
1365 },
1366 }
1367 \cs_generate_variant:Nn
1368 \tl_replace_all:Nnn
1369 { NVn }
1370 \cs_generate_variant:Nn
1371 \@@_set_theme:n
1372 { V }
1373 \cs_generate_variant:Nn
1374 \@@_setup_snippet:nn
1375 { 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.

```
1376 \ExplSyntaxOn
1377 \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.

```
1378 \prop_new:N \g_@@_renderer_arities_prop
1379 \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.

```
1380 \def\markdownRendererAttributeIdentifier{%
1381 \markdownRendererAttributeIdentifierPrototype}%
1382 \ExplSyntaxOn
1383 \seq_gput_right:Nn
1384 \g_@@_renderers_seq
1385 { attributeIdentifier }
1386 \prop_gput:Nnn
1387 \g_@@_renderer_arities_prop
1388 { attributeIdentifier }
1389 { 1 }
1390 \ExplSyntaxOff
1391 \def\markdownRendererAttributeName{%
```



```

1392 \markdownRendererAttributeClassNamePrototype}%
1393 \ExplSyntaxOn
1394 \seq_gput_right:Nn
1395 \g_@@_renderers_seq
1396 { attributeClassName }
1397 \prop_gput:Nnn
1398 \g_@@_renderer_arities_prop
1399 { attributeClassName }
1400 { 1 }
1401 \ExplSyntaxOff
1402 \def\markdownRendererAttributeKeyValue{%
1403 \markdownRendererAttributeKeyValuePrototype}%
1404 \ExplSyntaxOn
1405 \seq_gput_right:Nn
1406 \g_@@_renderers_seq
1407 { attributeKeyValue }
1408 \prop_gput:Nnn
1409 \g_@@_renderer_arities_prop
1410 { attributeKeyValue }
1411 { 2 }
1412 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1413 \def\markdownRendererBlockQuoteBegin{%
1414 \markdownRendererBlockQuoteBeginPrototype}%
1415 \ExplSyntaxOn
1416 \seq_gput_right:Nn
1417 \g_@@_renderers_seq
1418 { blockQuoteBegin }
1419 \prop_gput:Nnn
1420 \g_@@_renderer_arities_prop
1421 { blockQuoteBegin }
1422 { 0 }
1423 \ExplSyntaxOff

```

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

```

1424 \def\markdownRendererBlockQuoteEnd{%
1425 \markdownRendererBlockQuoteEndPrototype}%
1426 \ExplSyntaxOn
1427 \seq_gput_right:Nn
1428 \g_@@_renderers_seq
1429 { blockQuoteEnd }
1430 \prop_gput:Nnn

```

```

1431 \g_@@_renderer_arities_prop
1432 { blockQuoteEnd }
1433 { 0 }
1434 \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.

```

1435 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1436 \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1437 \ExplSyntaxOn
1438 \seq_gput_right:Nn
1439 \g_@@_renderers_seq
1440 { bracketedSpanAttributeContextBegin }
1441 \prop_gput:Nnn
1442 \g_@@_renderer_arities_prop
1443 { bracketedSpanAttributeContextBegin }
1444 { 0 }
1445 \ExplSyntaxOff
1446 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1447 \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1448 \ExplSyntaxOn
1449 \seq_gput_right:Nn
1450 \g_@@_renderers_seq
1451 { bracketedSpanAttributeContextEnd }
1452 \prop_gput:Nnn
1453 \g_@@_renderer_arities_prop
1454 { bracketedSpanAttributeContextEnd }
1455 { 0 }
1456 \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.

```

1457 \def\markdownRendererUlBegin{%
1458 \markdownRendererUlBeginPrototype}%
1459 \ExplSyntaxOn
1460 \seq_gput_right:Nn
1461 \g_@@_renderers_seq
1462 { ulBegin }

```

```

1463 \prop_gput:Nnn
1464 \g_@@_renderer_arities_prop
1465 { ulBegin }
1466 { 0 }
1467 \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.

```

1468 \def\markdownRendererUlBeginTight{%
1469 \markdownRendererUlBeginTightPrototype}%
1470 \ExplSyntaxOn
1471 \seq_gput_right:Nn
1472 \g_@@_renderers_seq
1473 { ulBeginTight }
1474 \prop_gput:Nnn
1475 \g_@@_renderer_arities_prop
1476 { ulBeginTight }
1477 { 0 }
1478 \ExplSyntaxOff

```

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

```

1479 \def\markdownRendererUlItem{%
1480 \markdownRendererUlItemPrototype}%
1481 \ExplSyntaxOn
1482 \seq_gput_right:Nn
1483 \g_@@_renderers_seq
1484 { ulItem }
1485 \prop_gput:Nnn
1486 \g_@@_renderer_arities_prop
1487 { ulItem }
1488 { 0 }
1489 \ExplSyntaxOff

```

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

```

1490 \def\markdownRendererUlItemEnd{%
1491 \markdownRendererUlItemEndPrototype}%
1492 \ExplSyntaxOn
1493 \seq_gput_right:Nn
1494 \g_@@_renderers_seq
1495 { ulItemEnd }
1496 \prop_gput:Nnn
1497 \g_@@_renderer_arities_prop

```

```

1498 { ulItemEnd }
1499 { 0 }
1500 \ExplSyntaxOff

```

The `\markdownRendererUEnd` 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.

```

1501 \def\markdownRendererUEnd{%
1502 \markdownRendererUEndPrototype}%
1503 \ExplSyntaxOn
1504 \seq_gput_right:Nn
1505 \g_@@_renderers_seq
1506 { ulEnd }
1507 \prop_gput:Nnn
1508 \g_@@_renderer_arities_prop
1509 { ulEnd }
1510 { 0 }
1511 \ExplSyntaxOff

```

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

```

1512 \def\markdownRendererUEndTight{%
1513 \markdownRendererUEndTightPrototype}%
1514 \ExplSyntaxOn
1515 \seq_gput_right:Nn
1516 \g_@@_renderers_seq
1517 { ulEndTight }
1518 \prop_gput:Nnn
1519 \g_@@_renderer_arities_prop
1520 { ulEndTight }
1521 { 0 }
1522 \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.

```

1523 \def\markdownRendererCite{%
1524 \markdownRendererCitePrototype}%
1525 \ExplSyntaxOn

```

```

1526 \seq_gput_right:Nn
1527 \g_@@_renderers_seq
1528 { cite }
1529 \prop_gput:Nnn
1530 \g_@@_renderer_arities_prop
1531 { cite }
1532 { 1 }
1533 \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.

```

1534 \def\markdownRendererTextCite{%
1535 \markdownRendererTextCitePrototype}%
1536 \ExplSyntaxOn
1537 \seq_gput_right:Nn
1538 \g_@@_renderers_seq
1539 { textCite }
1540 \prop_gput:Nnn
1541 \g_@@_renderer_arities_prop
1542 { textCite }
1543 { 1 }
1544 \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.

```

1545 \def\markdownRendererInputVerbatim{%
1546 \markdownRendererInputVerbatimPrototype}%
1547 \ExplSyntaxOn
1548 \seq_gput_right:Nn
1549 \g_@@_renderers_seq
1550 { inputVerbatim }
1551 \prop_gput:Nnn
1552 \g_@@_renderer_arities_prop
1553 { inputVerbatim }
1554 { 1 }
1555 \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.

```

1556 \def\markdownRendererInputFencedCode{%
1557 \markdownRendererInputFencedCodePrototype}%
1558 \ExplSyntaxOn
1559 \seq_gput_right:Nn
1560 \g_@@_renderers_seq
1561 { inputFencedCode }
1562 \prop_gput:Nnn
1563 \g_@@_renderer_arities_prop
1564 { inputFencedCode }
1565 { 3 }
1566 \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.

```

1567 \def\markdownRendererCodeSpan{%
1568 \markdownRendererCodeSpanPrototype}%
1569 \ExplSyntaxOn
1570 \seq_gput_right:Nn
1571 \g_@@_renderers_seq
1572 { codeSpan }
1573 \prop_gput:Nnn
1574 \g_@@_renderer_arities_prop
1575 { codeSpan }
1576 { 1 }
1577 \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.

```

1578 \def\markdownRendererCodeSpanAttributeContextBegin{%
1579 \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1580 \ExplSyntaxOn
1581 \seq_gput_right:Nn
1582 \g_@@_renderers_seq
1583 { codeSpanAttributeContextBegin }
1584 \prop_gput:Nnn
1585 \g_@@_renderer_arities_prop
1586 { codeSpanAttributeContextBegin }

```

```

1587 { 0 }
1588 \ExplSyntaxOff
1589 \def\markdownRendererCodeSpanAttributeContextEnd{%
1590 \markdownRendererCodeSpanAttributeContextEndPrototype}%
1591 \ExplSyntaxOn
1592 \seq_gput_right:Nn
1593 \g_@@_renderers_seq
1594 { codeSpanAttributeContextEnd }
1595 \prop_gput:Nnn
1596 \g_@@_renderer_arities_prop
1597 { codeSpanAttributeContextEnd }
1598 { 0 }
1599 \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.

```

1600 \def\markdownRendererContentBlock{%
1601 \markdownRendererContentBlockPrototype}%
1602 \ExplSyntaxOn
1603 \seq_gput_right:Nn
1604 \g_@@_renderers_seq
1605 { contentBlock }
1606 \prop_gput:Nnn
1607 \g_@@_renderer_arities_prop
1608 { contentBlock }
1609 { 4 }
1610 \ExplSyntaxOff

```

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

```

1611 \def\markdownRendererContentBlockOnlineImage{%
1612 \markdownRendererContentBlockOnlineImagePrototype}%
1613 \ExplSyntaxOn
1614 \seq_gput_right:Nn
1615 \g_@@_renderers_seq
1616 { contentBlockOnlineImage }
1617 \prop_gput:Nnn
1618 \g_@@_renderer_arities_prop
1619 { contentBlockOnlineImage }
1620 { 4 }
1621 \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>31</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 [3] is a good starting point.

```

1622 \def\markdownRendererContentBlockCode{%
1623 \markdownRendererContentBlockCodePrototype}%
1624 \ExplSyntaxOn
1625 \seq_gput_right:Nn
1626 \g_@@_renderers_seq
1627 { contentBlockCode }
1628 \prop_gput:Nnn
1629 \g_@@_renderer_arities_prop
1630 { contentBlockCode }
1631 { 5 }
1632 \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.

```

1633 \def\markdownRendererDlBegin{%
1634 \markdownRendererDlBeginPrototype}%
1635 \ExplSyntaxOn
1636 \seq_gput_right:Nn
1637 \g_@@_renderers_seq
1638 { dlBegin }
1639 \prop_gput:Nnn
1640 \g_@@_renderer_arities_prop
1641 { dlBegin }

```

---

<sup>31</sup> Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.



```

1642 { 0 }
1643 \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.

```

1644 \def\markdownRendererDlBeginTight{%
1645 \markdownRendererDlBeginTightPrototype}%
1646 \ExplSyntaxOn
1647 \seq_gput_right:Nn
1648 \g_@@_renderers_seq
1649 { dlBeginTight }
1650 \prop_gput:Nnn
1651 \g_@@_renderer_arities_prop
1652 { dlBeginTight }
1653 { 0 }
1654 \ExplSyntaxOff

```

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

```

1655 \def\markdownRendererDlItem{%
1656 \markdownRendererDlItemPrototype}%
1657 \ExplSyntaxOn
1658 \seq_gput_right:Nn
1659 \g_@@_renderers_seq
1660 { dlItem }
1661 \prop_gput:Nnn
1662 \g_@@_renderer_arities_prop
1663 { dlItem }
1664 { 1 }
1665 \ExplSyntaxOff

```

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

```

1666 \def\markdownRendererDlItemEnd{%
1667 \markdownRendererDlItemEndPrototype}%
1668 \ExplSyntaxOn
1669 \seq_gput_right:Nn
1670 \g_@@_renderers_seq
1671 { dlItemEnd }
1672 \prop_gput:Nnn
1673 \g_@@_renderer_arities_prop
1674 { dlItemEnd }
1675 { 0 }
1676 \ExplSyntaxOff

```

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

```
1677 \def\markdownRendererDlDefinitionBegin{%
1678 \markdownRendererDlDefinitionBeginPrototype}%
1679 \ExplSyntaxOn
1680 \seq_gput_right:Nn
1681 \g_@@_renderers_seq
1682 { dlDefinitionBegin }
1683 \prop_gput:Nnn
1684 \g_@@_renderer_arities_prop
1685 { dlDefinitionBegin }
1686 { 0 }
1687 \ExplSyntaxOff
```

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

```
1688 \def\markdownRendererDlDefinitionEnd{%
1689 \markdownRendererDlDefinitionEndPrototype}%
1690 \ExplSyntaxOn
1691 \seq_gput_right:Nn
1692 \g_@@_renderers_seq
1693 { dlDefinitionEnd }
1694 \prop_gput:Nnn
1695 \g_@@_renderer_arities_prop
1696 { dlDefinitionEnd }
1697 { 0 }
1698 \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.

```
1699 \def\markdownRendererDlEnd{%
1700 \markdownRendererDlEndPrototype}%
1701 \ExplSyntaxOn
1702 \seq_gput_right:Nn
1703 \g_@@_renderers_seq
1704 { dlEnd }
1705 \prop_gput:Nnn
1706 \g_@@_renderer_arities_prop
1707 { dlEnd }
1708 { 0 }
1709 \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.

```
1710 \def\markdownRendererDlEndTight{%
1711 \markdownRendererDlEndTightPrototype}%
1712 \ExplSyntaxOn
1713 \seq_gput_right:Nn
1714 \g_@@_renderers_seq
1715 { dlEndTight }
1716 \prop_gput:Nnn
1717 \g_@@_renderer_arities_prop
1718 { dlEndTight }
1719 { 0 }
1720 \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.

```
1721 \def\markdownRendererEllipsis{%
1722 \markdownRendererEllipsisPrototype}%
1723 \ExplSyntaxOn
1724 \seq_gput_right:Nn
1725 \g_@@_renderers_seq
1726 { ellipsis }
1727 \prop_gput:Nnn
1728 \g_@@_renderer_arities_prop
1729 { ellipsis }
1730 { 0 }
1731 \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.

```
1732 \def\markdownRendererEmphasis{%
1733 \markdownRendererEmphasisPrototype}%
1734 \ExplSyntaxOn
1735 \seq_gput_right:Nn
1736 \g_@@_renderers_seq
1737 { emphasis }
1738 \prop_gput:Nnn
1739 \g_@@_renderer_arities_prop
1740 { emphasis }
1741 { 1 }
```

```
1742 \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.

```
1743 \def\markdownRendererStrongEmphasis{%
1744 \markdownRendererStrongEmphasisPrototype}%
1745 \ExplSyntaxOn
1746 \seq_gput_right:Nn
1747 \g_@@_renderers_seq
1748 { strongEmphasis }
1749 \prop_gput:Nnn
1750 \g_@@_renderer_arities_prop
1751 { strongEmphasis }
1752 { 1 }
1753 \ExplSyntaxOff
```

### 2.2.5.13 Fenced Code Attribute Context Renderers

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

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

```
1754 \def\markdownRendererFencedCodeAttributeContextBegin{%
1755 \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1756 \ExplSyntaxOn
1757 \seq_gput_right:Nn
1758 \g_@@_renderers_seq
1759 { fencedCodeAttributeContextBegin }
1760 \prop_gput:Nnn
1761 \g_@@_renderer_arities_prop
1762 { fencedCodeAttributeContextBegin }
1763 { 0 }
1764 \ExplSyntaxOff
1765 \def\markdownRendererFencedCodeAttributeContextEnd{%
1766 \markdownRendererFencedCodeAttributeContextEndPrototype}%
1767 \ExplSyntaxOn
1768 \seq_gput_right:Nn
1769 \g_@@_renderers_seq
1770 { fencedCodeAttributeContextEnd }
1771 \prop_gput:Nnn
1772 \g_@@_renderer_arities_prop
1773 { fencedCodeAttributeContextEnd }
1774 { 0 }
1775 \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.

```
1776 \def\markdownRendererFencedDivAttributeContextBegin{%
1777 \markdownRendererFencedDivAttributeContextBeginPrototype}%
1778 \ExplSyntaxOn
1779 \seq_gput_right:Nn
1780 \g_@@_renderers_seq
1781 { fencedDivAttributeContextBegin }
1782 \prop_gput:Nnn
1783 \g_@@_renderer_arities_prop
1784 { fencedDivAttributeContextBegin }
1785 { 0 }
1786 \ExplSyntaxOff
1787 \def\markdownRendererFencedDivAttributeContextEnd{%
1788 \markdownRendererFencedDivAttributeContextEndPrototype}%
1789 \ExplSyntaxOn
1790 \seq_gput_right:Nn
1791 \g_@@_renderers_seq
1792 { fencedDivAttributeContextEnd }
1793 \prop_gput:Nnn
1794 \g_@@_renderer_arities_prop
1795 { fencedDivAttributeContextEnd }
1796 { 0 }
1797 \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.

```
1798 \def\markdownRendererHeaderAttributeContextBegin{%
1799 \markdownRendererHeaderAttributeContextBeginPrototype}%
1800 \ExplSyntaxOn
1801 \seq_gput_right:Nn
1802 \g_@@_renderers_seq
1803 { headerAttributeContextBegin }
1804 \prop_gput:Nnn
1805 \g_@@_renderer_arities_prop
1806 { headerAttributeContextBegin }
1807 { 0 }
1808 \ExplSyntaxOff
```

```

1809 \def\markdownRendererHeaderAttributeContextEnd{%
1810 \markdownRendererHeaderAttributeContextEndPrototype}%
1811 \ExplSyntaxOn
1812 \seq_gput_right:Nn
1813 \g_@@_renderers_seq
1814 { headerAttributeContextEnd }
1815 \prop_gput:Nnn
1816 \g_@@_renderer_arities_prop
1817 { headerAttributeContextEnd }
1818 { 0 }
1819 \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.

```

1820 \def\markdownRendererHeadingOne{%
1821 \markdownRendererHeadingOnePrototype}%
1822 \ExplSyntaxOn
1823 \seq_gput_right:Nn
1824 \g_@@_renderers_seq
1825 { headingOne }
1826 \prop_gput:Nnn
1827 \g_@@_renderer_arities_prop
1828 { headingOne }
1829 { 1 }
1830 \ExplSyntaxOff

```

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

```

1831 \def\markdownRendererHeadingTwo{%
1832 \markdownRendererHeadingTwoPrototype}%
1833 \ExplSyntaxOn
1834 \seq_gput_right:Nn
1835 \g_@@_renderers_seq
1836 { headingTwo }
1837 \prop_gput:Nnn
1838 \g_@@_renderer_arities_prop
1839 { headingTwo }
1840 { 1 }
1841 \ExplSyntaxOff

```

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

```

1842 \def\markdownRendererHeadingThree{%
1843 \markdownRendererHeadingThreePrototype}%
1844 \ExplSyntaxOn

```

```

1845 \seq_gput_right:Nn
1846 \g_@@_renderers_seq
1847 { headingThree }
1848 \prop_gput:Nnn
1849 \g_@@_renderer_arities_prop
1850 { headingThree }
1851 { 1 }
1852 \ExplSyntaxOff

```

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

```

1853 \def\markdownRendererHeadingFour{%
1854 \markdownRendererHeadingFourPrototype}%
1855 \ExplSyntaxOn
1856 \seq_gput_right:Nn
1857 \g_@@_renderers_seq
1858 { headingFour }
1859 \prop_gput:Nnn
1860 \g_@@_renderer_arities_prop
1861 { headingFour }
1862 { 1 }
1863 \ExplSyntaxOff

```

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

```

1864 \def\markdownRendererHeadingFive{%
1865 \markdownRendererHeadingFivePrototype}%
1866 \ExplSyntaxOn
1867 \seq_gput_right:Nn
1868 \g_@@_renderers_seq
1869 { headingFive }
1870 \prop_gput:Nnn
1871 \g_@@_renderer_arities_prop
1872 { headingFive }
1873 { 1 }
1874 \ExplSyntaxOff

```

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

```

1875 \def\markdownRendererHeadingSix{%
1876 \markdownRendererHeadingSixPrototype}%
1877 \ExplSyntaxOn
1878 \seq_gput_right:Nn
1879 \g_@@_renderers_seq
1880 { headingSix }
1881 \prop_gput:Nnn
1882 \g_@@_renderer_arities_prop

```

```

1883 { headingSix }
1884 { 1 }
1885 \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.

```

1886 \def\markdownRendererInlineHtmlComment{%
1887 \markdownRendererInlineHtmlCommentPrototype}%
1888 \ExplSyntaxOn
1889 \seq_gput_right:Nn
1890 \g_@@_renderers_seq
1891 { inlineHtmlComment }
1892 \prop_gput:Nnn
1893 \g_@@_renderer_arities_prop
1894 { inlineHtmlComment }
1895 { 1 }
1896 \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.

```

1897 \def\markdownRendererInlineHtmlTag{%
1898 \markdownRendererInlineHtmlTagPrototype}%
1899 \ExplSyntaxOn
1900 \seq_gput_right:Nn
1901 \g_@@_renderers_seq
1902 { inlineHtmlTag }
1903 \prop_gput:Nnn
1904 \g_@@_renderer_arities_prop
1905 { inlineHtmlTag }
1906 { 1 }
1907 \ExplSyntaxOff
1908 \def\markdownRendererInputBlockHtmlElement{%
1909 \markdownRendererInputBlockHtmlElementPrototype}%
1910 \ExplSyntaxOn

```



```

1911 \seq_gput_right:Nn
1912 \g_@@_renderers_seq
1913 { inputBlockHtmlElement }
1914 \prop_gput:Nnn
1915 \g_@@_renderer_arities_prop
1916 { inputBlockHtmlElement }
1917 { 1 }
1918 \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.

```

1919 \def\markdownRendererImage{%
1920 \markdownRendererImagePrototype}%
1921 \ExplSyntaxOn
1922 \seq_gput_right:Nn
1923 \g_@@_renderers_seq
1924 { image }
1925 \prop_gput:Nnn
1926 \g_@@_renderer_arities_prop
1927 { image }
1928 { 4 }
1929 \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.

```

1930 \def\markdownRendererImageAttributeContextBegin{%
1931 \markdownRendererImageAttributeContextBeginPrototype}%
1932 \ExplSyntaxOn
1933 \seq_gput_right:Nn
1934 \g_@@_renderers_seq
1935 { imageAttributeContextBegin }
1936 \prop_gput:Nnn
1937 \g_@@_renderer_arities_prop
1938 { imageAttributeContextBegin }
1939 { 0 }
1940 \ExplSyntaxOff
1941 \def\markdownRendererImageAttributeContextEnd{%
1942 \markdownRendererImageAttributeContextEndPrototype}%

```

```

1943 \ExplSyntaxOn
1944 \seq_gput_right:Nn
1945 \g_@@_renderers_seq
1946 { imageAttributeContextEnd }
1947 \prop_gput:Nnn
1948 \g_@@_renderer_arities_prop
1949 { imageAttributeContextEnd }
1950 { 0 }
1951 \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.

```

1952 \def\markdownRendererInterblockSeparator{%
1953 \markdownRendererInterblockSeparatorPrototype}%
1954 \ExplSyntaxOn
1955 \seq_gput_right:Nn
1956 \g_@@_renderers_seq
1957 { interblockSeparator }
1958 \prop_gput:Nnn
1959 \g_@@_renderer_arities_prop
1960 { interblockSeparator }
1961 { 0 }
1962 \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.

```

1963 \def\markdownRendererParagraphSeparator{%
1964 \markdownRendererParagraphSeparatorPrototype}%
1965 \ExplSyntaxOn
1966 \seq_gput_right:Nn
1967 \g_@@_renderers_seq
1968 { paragraphSeparator }
1969 \prop_gput:Nnn
1970 \g_@@_renderer_arities_prop
1971 { paragraphSeparator }
1972 { 0 }
1973 \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.

```
1974 \def\markdownRendererLineBlockBegin{%
1975 \markdownRendererLineBlockBeginPrototype}%
1976 \ExplSyntaxOn
1977 \seq_gput_right:Nn
1978 \g_@@_renderers_seq
1979 { lineBlockBegin }
1980 \prop_gput:Nnn
1981 \g_@@_renderer_arities_prop
1982 { lineBlockBegin }
1983 { 0 }
1984 \ExplSyntaxOff
1985 \def\markdownRendererLineBlockEnd{%
1986 \markdownRendererLineBlockEndPrototype}%
1987 \ExplSyntaxOn
1988 \seq_gput_right:Nn
1989 \g_@@_renderers_seq
1990 { lineBlockEnd }
1991 \prop_gput:Nnn
1992 \g_@@_renderer_arities_prop
1993 { lineBlockEnd }
1994 { 0 }
1995 \ExplSyntaxOff
```

### 2.2.5.23 Line Break Renderers

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

```
1996 \def\markdownRendererSoftLineBreak{%
1997 \markdownRendererSoftLineBreakPrototype}%
1998 \ExplSyntaxOn
1999 \seq_gput_right:Nn
2000 \g_@@_renderers_seq
2001 { softLineBreak }
2002 \prop_gput:Nnn
2003 \g_@@_renderer_arities_prop
2004 { softLineBreak }
2005 { 0 }
2006 \ExplSyntaxOff
```

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

```
2007 \def\markdownRendererHardLineBreak{%
2008 \markdownRendererHardLineBreakPrototype}%
```

```

2009 \ExplSyntaxOn
2010 \seq_gput_right:Nn
2011 \g_@@_renderers_seq
2012 { hardLineBreak }
2013 \prop_gput:Nnn
2014 \g_@@_renderer_arities_prop
2015 { hardLineBreak }
2016 { 0 }
2017 \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.

```

2018 \def\markdownRendererLink{%
2019 \markdownRendererLinkPrototype}%
2020 \ExplSyntaxOn
2021 \seq_gput_right:Nn
2022 \g_@@_renderers_seq
2023 { link }
2024 \prop_gput:Nnn
2025 \g_@@_renderer_arities_prop
2026 { link }
2027 { 4 }
2028 \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.

```

2029 \def\markdownRendererLinkAttributeContextBegin{%
2030 \markdownRendererLinkAttributeContextBeginPrototype}%
2031 \ExplSyntaxOn
2032 \seq_gput_right:Nn
2033 \g_@@_renderers_seq
2034 { linkAttributeContextBegin }
2035 \prop_gput:Nnn
2036 \g_@@_renderer_arities_prop
2037 { linkAttributeContextBegin }
2038 { 0 }
2039 \ExplSyntaxOff
2040 \def\markdownRendererLinkAttributeContextEnd{%

```

```

2041 \markdownRendererLinkAttributeContextEndPrototype}%
2042 \ExplSyntaxOn
2043 \seq_gput_right:Nn
2044 \g_@@_renderers_seq
2045 { linkAttributeContextEnd }
2046 \prop_gput:Nnn
2047 \g_@@_renderer_arities_prop
2048 { linkAttributeContextEnd }
2049 { 0 }
2050 \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.

```

2051 \def\markdownRendererMark{%
2052 \markdownRendererMarkPrototype}%
2053 \ExplSyntaxOn
2054 \seq_gput_right:Nn
2055 \g_@@_renderers_seq
2056 { mark }
2057 \prop_gput:Nnn
2058 \g_@@_renderer_arities_prop
2059 { mark }
2060 { 1 }
2061 \ExplSyntaxOff

```

### 2.2.5.27 Markdown Document Renderers

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

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

```

2062 \def\markdownRendererDocumentBegin{%
2063 \markdownRendererDocumentBeginPrototype}%
2064 \ExplSyntaxOn
2065 \seq_gput_right:Nn
2066 \g_@@_renderers_seq
2067 { documentBegin }
2068 \prop_gput:Nnn
2069 \g_@@_renderer_arities_prop
2070 { documentBegin }

```

```

2071 { 0 }
2072 \ExplSyntaxOff
2073 \def\markdownRendererDocumentEnd{%
2074 \markdownRendererDocumentEndPrototype}%
2075 \ExplSyntaxOn
2076 \seq_gput_right:Nn
2077 \g_@@_renderers_seq
2078 { documentEnd }
2079 \prop_gput:Nnn
2080 \g_@@_renderer_arities_prop
2081 { documentEnd }
2082 { 0 }
2083 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2084 \def\markdownRendererNbsp{%
2085 \markdownRendererNbspPrototype}%
2086 \ExplSyntaxOn
2087 \seq_gput_right:Nn
2088 \g_@@_renderers_seq
2089 { nbsp }
2090 \prop_gput:Nnn
2091 \g_@@_renderer_arities_prop
2092 { nbsp }
2093 { 0 }
2094 \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.

```

2095 \def\markdownRendererNote{%
2096 \markdownRendererNotePrototype}%
2097 \ExplSyntaxOn
2098 \seq_gput_right:Nn
2099 \g_@@_renderers_seq
2100 { note }
2101 \prop_gput:Nnn
2102 \g_@@_renderer_arities_prop
2103 { note }
2104 { 1 }
2105 \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.

```
2106 \def\markdownRendererOlBegin{%
2107 \markdownRendererOlBeginPrototype}%
2108 \ExplSyntaxOn
2109 \seq_gput_right:Nn
2110 \g_@@_renderers_seq
2111 { olBegin }
2112 \prop_gput:Nnn
2113 \g_@@_renderer_arities_prop
2114 { olBegin }
2115 { 0 }
2116 \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.

```
2117 \def\markdownRendererOlBeginTight{%
2118 \markdownRendererOlBeginTightPrototype}%
2119 \ExplSyntaxOn
2120 \seq_gput_right:Nn
2121 \g_@@_renderers_seq
2122 { olBeginTight }
2123 \prop_gput:Nnn
2124 \g_@@_renderer_arities_prop
2125 { olBeginTight }
2126 { 0 }
2127 \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`).

```
2128 \def\markdownRendererFancyOlBegin{%
2129 \markdownRendererFancyOlBeginPrototype}%
2130 \ExplSyntaxOn
2131 \seq_gput_right:Nn
2132 \g_@@_renderers_seq
2133 { fancyOlBegin }
```

```

2134 \prop_gput:Nnn
2135 \g_@@_renderer_arities_prop
2136 { fancyO1Begin }
2137 { 2 }
2138 \ExplSyntaxOff

```

The `\markdownRendererFancyO1BeginTight` 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 `\markdownRendererFancyO1Begin` macro for the valid style values.

```

2139 \def\markdownRendererFancyO1BeginTight{%
2140 \markdownRendererFancyO1BeginTightPrototype}%
2141 \ExplSyntaxOn
2142 \seq_gput_right:Nn
2143 \g_@@_renderers_seq
2144 { fancyO1BeginTight }
2145 \prop_gput:Nnn
2146 \g_@@_renderer_arities_prop
2147 { fancyO1BeginTight }
2148 { 2 }
2149 \ExplSyntaxOff

```

The `\markdownRendererO1Item` 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.

```

2150 \def\markdownRendererO1Item{%
2151 \markdownRendererO1ItemPrototype}%
2152 \ExplSyntaxOn
2153 \seq_gput_right:Nn
2154 \g_@@_renderers_seq
2155 { olItem }
2156 \prop_gput:Nnn
2157 \g_@@_renderer_arities_prop
2158 { olItem }
2159 { 0 }
2160 \ExplSyntaxOff

```

The `\markdownRendererO1ItemEnd` macro represents the end of an item in an ordered list. This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

2161 \def\markdownRendererO1ItemEnd{%
2162 \markdownRendererO1ItemEndPrototype}%
2163 \ExplSyntaxOn
2164 \seq_gput_right:Nn

```



```

2165 \g_@@_renderers_seq
2166 { olItemEnd }
2167 \prop_gput:Nnn
2168 \g_@@_renderer_arities_prop
2169 { olItemEnd }
2170 { 0 }
2171 \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.

```

2172 \def\markdownRendererOlItemWithNumber{%
2173 \markdownRendererOlItemWithNumberPrototype}%
2174 \ExplSyntaxOn
2175 \seq_gput_right:Nn
2176 \g_@@_renderers_seq
2177 { olItemWithNumber }
2178 \prop_gput:Nnn
2179 \g_@@_renderer_arities_prop
2180 { olItemWithNumber }
2181 { 1 }
2182 \ExplSyntaxOff

```

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

```

2183 \def\markdownRendererFancyOlItem{%
2184 \markdownRendererFancyOlItemPrototype}%
2185 \ExplSyntaxOn
2186 \seq_gput_right:Nn
2187 \g_@@_renderers_seq
2188 { fancyOlItem }
2189 \prop_gput:Nnn
2190 \g_@@_renderer_arities_prop
2191 { fancyOlItem }
2192 { 0 }
2193 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItemEnd` 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.

```

2194 \def\markdownRendererFancyOlItemEnd{%
2195 \markdownRendererFancyOlItemEndPrototype}%
2196 \ExplSyntaxOn
2197 \seq_gput_right:Nn

```

```

2198 \g_@@_renderers_seq
2199 { fancyO1ItemEnd }
2200 \prop_gput:Nnn
2201 \g_@@_renderer_arities_prop
2202 { fancyO1ItemEnd }
2203 { 0 }
2204 \ExplSyntaxOff

```

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

```

2205 \def\markdownRendererFancyO1ItemWithNumber{%
2206 \markdownRendererFancyO1ItemWithNumberPrototype}%
2207 \ExplSyntaxOn
2208 \seq_gput_right:Nn
2209 \g_@@_renderers_seq
2210 { fancyO1ItemWithNumber }
2211 \prop_gput:Nnn
2212 \g_@@_renderer_arities_prop
2213 { fancyO1ItemWithNumber }
2214 { 1 }
2215 \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.

```

2216 \def\markdownRendererO1End{%
2217 \markdownRendererO1EndPrototype}%
2218 \ExplSyntaxOn
2219 \seq_gput_right:Nn
2220 \g_@@_renderers_seq
2221 { olEnd }
2222 \prop_gput:Nnn
2223 \g_@@_renderer_arities_prop
2224 { olEnd }
2225 { 0 }
2226 \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.

```

2227 \def\markdownRendererO1EndTight{%
2228 \markdownRendererO1EndTightPrototype}%

```

```

2229 \ExplSyntaxOn
2230 \seq_gput_right:Nn
2231 \g_@@_renderers_seq
2232 { olEndTight }
2233 \prop_gput:Nnn
2234 \g_@@_renderer_arities_prop
2235 { olEndTight }
2236 { 0 }
2237 \ExplSyntaxOff

```

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

```

2238 \def\markdownRendererFancyOlEnd{%
2239 \markdownRendererFancyOlEndPrototype}%
2240 \ExplSyntaxOn
2241 \seq_gput_right:Nn
2242 \g_@@_renderers_seq
2243 { fancyOlEnd }
2244 \prop_gput:Nnn
2245 \g_@@_renderer_arities_prop
2246 { fancyOlEnd }
2247 { 0 }
2248 \ExplSyntaxOff

```

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

```

2249 \def\markdownRendererFancyOlEndTight{%
2250 \markdownRendererFancyOlEndTightPrototype}%
2251 \ExplSyntaxOn
2252 \seq_gput_right:Nn
2253 \g_@@_renderers_seq
2254 { fancyOlEndTight }
2255 \prop_gput:Nnn
2256 \g_@@_renderer_arities_prop
2257 { fancyOlEndTight }
2258 { 0 }
2259 \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.

```

2260 \def\markdownRendererInputRawInline{%
2261 \markdownRendererInputRawInlinePrototype}%
2262 \ExplSyntaxOn
2263 \seq_gput_right:Nn
2264 \g_@@_renderers_seq
2265 { inputRawInline }
2266 \prop_gput:Nnn
2267 \g_@@_renderer_arities_prop
2268 { inputRawInline }
2269 { 2 }
2270 \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.

```

2271 \def\markdownRendererInputRawBlock{%
2272 \markdownRendererInputRawBlockPrototype}%
2273 \ExplSyntaxOn
2274 \seq_gput_right:Nn
2275 \g_@@_renderers_seq
2276 { inputRawBlock }
2277 \prop_gput:Nnn
2278 \g_@@_renderer_arities_prop
2279 { inputRawBlock }
2280 { 2 }
2281 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2282 \def\markdownRendererSectionBegin{%
2283 \markdownRendererSectionBeginPrototype}%
2284 \ExplSyntaxOn
2285 \seq_gput_right:Nn
2286 \g_@@_renderers_seq
2287 { sectionBegin }
2288 \prop_gput:Nnn
2289 \g_@@_renderer_arities_prop
2290 { sectionBegin }
2291 { 0 }
2292 \ExplSyntaxOff
2293 \def\markdownRendererSectionEnd{%

```

```

2294 \markdownRendererSectionEndPrototype}%
2295 \ExplSyntaxOn
2296 \seq_gput_right:Nn
2297 \g_@@_renderers_seq
2298 { sectionEnd }
2299 \prop_gput:Nnn
2300 \g_@@_renderer_arities_prop
2301 { sectionEnd }
2302 { 0 }
2303 \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.

```

2304 \def\markdownRendererReplacementCharacter{%
2305 \markdownRendererReplacementCharacterPrototype}%
2306 \ExplSyntaxOn
2307 \seq_gput_right:Nn
2308 \g_@@_renderers_seq
2309 { replacementCharacter }
2310 \prop_gput:Nnn
2311 \g_@@_renderer_arities_prop
2312 { replacementCharacter }
2313 { 0 }
2314 \ExplSyntaxOff

```

### 2.2.5.34 Special Character Renderers

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

```

2315 \def\markdownRendererLeftBrace{%
2316 \markdownRendererLeftBracePrototype}%
2317 \ExplSyntaxOn
2318 \seq_gput_right:Nn
2319 \g_@@_renderers_seq
2320 { leftBrace }
2321 \prop_gput:Nnn
2322 \g_@@_renderer_arities_prop
2323 { leftBrace }
2324 { 0 }
2325 \ExplSyntaxOff
2326 \def\markdownRendererRightBrace{%
2327 \markdownRendererRightBracePrototype}%
2328 \ExplSyntaxOn
2329 \seq_gput_right:Nn

```

```

2330 \g_@@_renderers_seq
2331 { rightBrace }
2332 \prop_gput:Nnn
2333 \g_@@_renderer_arities_prop
2334 { rightBrace }
2335 { 0 }
2336 \ExplSyntaxOff
2337 \def\markdownRendererDollarSign{%
2338 \markdownRendererDollarSignPrototype}%
2339 \ExplSyntaxOn
2340 \seq_gput_right:Nn
2341 \g_@@_renderers_seq
2342 { dollarSign }
2343 \prop_gput:Nnn
2344 \g_@@_renderer_arities_prop
2345 { dollarSign }
2346 { 0 }
2347 \ExplSyntaxOff
2348 \def\markdownRendererPercentSign{%
2349 \markdownRendererPercentSignPrototype}%
2350 \ExplSyntaxOn
2351 \seq_gput_right:Nn
2352 \g_@@_renderers_seq
2353 { percentSign }
2354 \prop_gput:Nnn
2355 \g_@@_renderer_arities_prop
2356 { percentSign }
2357 { 0 }
2358 \ExplSyntaxOff
2359 \def\markdownRendererAmpersand{%
2360 \markdownRendererAmpersandPrototype}%
2361 \ExplSyntaxOn
2362 \seq_gput_right:Nn
2363 \g_@@_renderers_seq
2364 { ampersand }
2365 \prop_gput:Nnn
2366 \g_@@_renderer_arities_prop
2367 { ampersand }
2368 { 0 }
2369 \ExplSyntaxOff
2370 \def\markdownRendererUnderscore{%
2371 \markdownRendererUnderscorePrototype}%
2372 \ExplSyntaxOn
2373 \seq_gput_right:Nn
2374 \g_@@_renderers_seq
2375 { underscore }
2376 \prop_gput:Nnn

```

```

2377 \g_@@_renderer_arities_prop
2378 { underscore }
2379 { 0 }
2380 \ExplSyntaxOff
2381 \def\markdownRendererHash{%
2382 \markdownRendererHashPrototype}%
2383 \ExplSyntaxOn
2384 \seq_gput_right:Nn
2385 \g_@@_renderers_seq
2386 { hash }
2387 \prop_gput:Nnn
2388 \g_@@_renderer_arities_prop
2389 { hash }
2390 { 0 }
2391 \ExplSyntaxOff
2392 \def\markdownRendererCircumflex{%
2393 \markdownRendererCircumflexPrototype}%
2394 \ExplSyntaxOn
2395 \seq_gput_right:Nn
2396 \g_@@_renderers_seq
2397 { circumflex }
2398 \prop_gput:Nnn
2399 \g_@@_renderer_arities_prop
2400 { circumflex }
2401 { 0 }
2402 \ExplSyntaxOff
2403 \def\markdownRendererBackslash{%
2404 \markdownRendererBackslashPrototype}%
2405 \ExplSyntaxOn
2406 \seq_gput_right:Nn
2407 \g_@@_renderers_seq
2408 { backslash }
2409 \prop_gput:Nnn
2410 \g_@@_renderer_arities_prop
2411 { backslash }
2412 { 0 }
2413 \ExplSyntaxOff
2414 \def\markdownRendererTilde{%
2415 \markdownRendererTildePrototype}%
2416 \ExplSyntaxOn
2417 \seq_gput_right:Nn
2418 \g_@@_renderers_seq
2419 { tilde }
2420 \prop_gput:Nnn
2421 \g_@@_renderer_arities_prop
2422 { tilde }
2423 { 0 }

```

```

2424 \ExplSyntaxOff
2425 \def\markdownRendererPipe{%
2426 \markdownRendererPipePrototype}%
2427 \ExplSyntaxOn
2428 \seq_gput_right:Nn
2429 \g_@@_renderers_seq
2430 { pipe }
2431 \prop_gput:Nnn
2432 \g_@@_renderer_arities_prop
2433 { pipe }
2434 { 0 }
2435 \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.

```

2436 \def\markdownRendererStrikeThrough{%
2437 \markdownRendererStrikeThroughPrototype}%
2438 \ExplSyntaxOn
2439 \seq_gput_right:Nn
2440 \g_@@_renderers_seq
2441 { strikeThrough }
2442 \prop_gput:Nnn
2443 \g_@@_renderer_arities_prop
2444 { strikeThrough }
2445 { 1 }
2446 \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.

```

2447 \def\markdownRendererSubscript{%
2448 \markdownRendererSubscriptPrototype}%
2449 \ExplSyntaxOn
2450 \seq_gput_right:Nn
2451 \g_@@_renderers_seq
2452 { subscript }
2453 \prop_gput:Nnn
2454 \g_@@_renderer_arities_prop
2455 { subscript }
2456 { 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.

```
2457 \def\markdownRendererSuperscript{%
2458 \markdownRendererSuperscriptPrototype}%
2459 \ExplSyntaxOn
2460 \seq_gput_right:Nn
2461 \g_@@_renderers_seq
2462 { superscript }
2463 \prop_gput:Nnn
2464 \g_@@_renderer_arities_prop
2465 { superscript }
2466 { 1 }
2467 \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.

```
2468 \def\markdownRendererTableAttributeContextBegin{%
2469 \markdownRendererTableAttributeContextBeginPrototype}%
2470 \ExplSyntaxOn
2471 \seq_gput_right:Nn
2472 \g_@@_renderers_seq
2473 { tableAttributeContextBegin }
2474 \prop_gput:Nnn
2475 \g_@@_renderer_arities_prop
2476 { tableAttributeContextBegin }
2477 { 0 }
2478 \ExplSyntaxOff
2479 \def\markdownRendererTableAttributeContextEnd{%
2480 \markdownRendererTableAttributeContextEndPrototype}%
2481 \ExplSyntaxOn
2482 \seq_gput_right:Nn
2483 \g_@@_renderers_seq
2484 { tableAttributeContextEnd }
2485 \prop_gput:Nnn
2486 \g_@@_renderer_arities_prop
2487 { tableAttributeContextEnd }
2488 { 0 }
2489 \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.

```
2490 \def\markdownRendererTable{%
2491 \markdownRendererTablePrototype}%
2492 \ExplSyntaxOn
2493 \seq_gput_right:Nn
2494 \g_@@_renderers_seq
2495 { table }
2496 \prop_gput:Nnn
2497 \g_@@_renderer_arities_prop
2498 { table }
2499 { 3 }
2500 \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.

```
2501 \def\markdownRendererInlineMath{%
2502 \markdownRendererInlineMathPrototype}%
2503 \ExplSyntaxOn
2504 \seq_gput_right:Nn
2505 \g_@@_renderers_seq
2506 { inlineMath }
2507 \prop_gput:Nnn
2508 \g_@@_renderer_arities_prop
2509 { inlineMath }
2510 { 1 }
2511 \ExplSyntaxOff
2512 \def\markdownRendererDisplayMath{%
2513 \markdownRendererDisplayMathPrototype}%
```

```

2514 \ExplSyntaxOn
2515 \seq_gput_right:Nn
2516 \g_@@_renderers_seq
2517 { displayMath }
2518 \prop_gput:Nnn
2519 \g_@@_renderer_arities_prop
2520 { displayMath }
2521 { 1 }
2522 \ExplSyntaxOff

```

### 2.2.5.41 Thematic Break Renderer

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

```

2523 \def\markdownRendererThematicBreak{%
2524 \markdownRendererThematicBreakPrototype}%
2525 \ExplSyntaxOn
2526 \seq_gput_right:Nn
2527 \g_@@_renderers_seq
2528 { thematicBreak }
2529 \prop_gput:Nnn
2530 \g_@@_renderer_arities_prop
2531 { thematicBreak }
2532 { 0 }
2533 \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.

```

2534 \def\markdownRendererTickedBox{%
2535 \markdownRendererTickedBoxPrototype}%
2536 \ExplSyntaxOn
2537 \seq_gput_right:Nn
2538 \g_@@_renderers_seq
2539 { tickedBox }
2540 \prop_gput:Nnn
2541 \g_@@_renderer_arities_prop
2542 { tickedBox }
2543 { 0 }
2544 \ExplSyntaxOff
2545 \def\markdownRendererHalfTickedBox{%
2546 \markdownRendererHalfTickedBoxPrototype}%

```

```

2547 \ExplSyntaxOn
2548 \seq_gput_right:Nn
2549 \g_@@_renderers_seq
2550 { halfTickedBox }
2551 \prop_gput:Nnn
2552 \g_@@_renderer_arities_prop
2553 { halfTickedBox }
2554 { 0 }
2555 \ExplSyntaxOff
2556 \def\markdownRendererUntickedBox{%
2557 \markdownRendererUntickedBoxPrototype}%
2558 \ExplSyntaxOn
2559 \seq_gput_right:Nn
2560 \g_@@_renderers_seq
2561 { untickedBox }
2562 \prop_gput:Nnn
2563 \g_@@_renderer_arities_prop
2564 { untickedBox }
2565 { 0 }
2566 \ExplSyntaxOff

```

### 2.2.5.43 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.

```

2567 \def\markdownRendererJekyllDataBegin{%
2568 \markdownRendererJekyllDataBeginPrototype}%
2569 \ExplSyntaxOn
2570 \seq_gput_right:Nn
2571 \g_@@_renderers_seq
2572 { jekyllDataBegin }
2573 \prop_gput:Nnn
2574 \g_@@_renderer_arities_prop
2575 { jekyllDataBegin }
2576 { 0 }
2577 \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.

```

2578 \def\markdownRendererJekyllDataEnd{%
2579 \markdownRendererJekyllDataEndPrototype}%
2580 \ExplSyntaxOn
2581 \seq_gput_right:Nn
2582 \g_@@_renderers_seq
2583 { jekyllDataEnd }

```

```

2584 \prop_gput:Nnn
2585 \g_@@_renderer_arities_prop
2586 { jekyllDataEnd }
2587 { 0 }
2588 \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.

```

2589 \def\markdownRendererJekyllDataMappingBegin{%
2590 \markdownRendererJekyllDataMappingBeginPrototype}%
2591 \ExplSyntaxOn
2592 \seq_gput_right:Nn
2593 \g_@@_renderers_seq
2594 { jekyllDataMappingBegin }
2595 \prop_gput:Nnn
2596 \g_@@_renderer_arities_prop
2597 { jekyllDataMappingBegin }
2598 { 2 }
2599 \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.

```

2600 \def\markdownRendererJekyllDataMappingEnd{%
2601 \markdownRendererJekyllDataMappingEndPrototype}%
2602 \ExplSyntaxOn
2603 \seq_gput_right:Nn
2604 \g_@@_renderers_seq
2605 { jekyllDataMappingEnd }
2606 \prop_gput:Nnn
2607 \g_@@_renderer_arities_prop
2608 { jekyllDataMappingEnd }
2609 { 0 }
2610 \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.

```

2611 \def\markdownRendererJekyllDataSequenceBegin{%
2612 \markdownRendererJekyllDataSequenceBeginPrototype}%
2613 \ExplSyntaxOn

```

```

2614 \seq_gput_right:Nn
2615 \g_@@_renderers_seq
2616 { jekyllDataSequenceBegin }
2617 \prop_gput:Nnn
2618 \g_@@_renderer_arities_prop
2619 { jekyllDataSequenceBegin }
2620 { 2 }
2621 \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.

```

2622 \def\markdownRendererJekyllDataSequenceEnd{%
2623 \markdownRendererJekyllDataSequenceEndPrototype}%
2624 \ExplSyntaxOn
2625 \seq_gput_right:Nn
2626 \g_@@_renderers_seq
2627 { jekyllDataSequenceEnd }
2628 \prop_gput:Nnn
2629 \g_@@_renderer_arities_prop
2630 { jekyllDataSequenceEnd }
2631 { 0 }
2632 \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.

```

2633 \def\markdownRendererJekyllDataBoolean{%
2634 \markdownRendererJekyllDataBooleanPrototype}%
2635 \ExplSyntaxOn
2636 \seq_gput_right:Nn
2637 \g_@@_renderers_seq
2638 { jekyllDataBoolean }
2639 \prop_gput:Nnn
2640 \g_@@_renderer_arities_prop
2641 { jekyllDataBoolean }
2642 { 2 }
2643 \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.

```

2644 \def\markdownRendererJekyllDataNumber{%
2645 \markdownRendererJekyllDataNumberPrototype}%
2646 \ExplSyntaxOn
2647 \seq_gput_right:Nn
2648 \g_@@_renderers_seq
2649 { jekyllDataNumber }
2650 \prop_gput:Nnn
2651 \g_@@_renderer_arities_prop
2652 { jekyllDataNumber }
2653 { 2 }
2654 \ExplSyntaxOff

```

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

```

2655 \def\markdownRendererJekyllDataString{%
2656 \markdownRendererJekyllDataStringPrototype}%
2657 \ExplSyntaxOn
2658 \seq_gput_right:Nn
2659 \g_@@_renderers_seq
2660 { jekyllDataString }
2661 \prop_gput:Nnn
2662 \g_@@_renderer_arities_prop
2663 { jekyllDataString }
2664 { 2 }
2665 \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.

```

2666 \def\markdownRendererJekyllDataEmpty{%
2667 \markdownRendererJekyllDataEmptyPrototype}%
2668 \ExplSyntaxOn
2669 \seq_gput_right:Nn
2670 \g_@@_renderers_seq
2671 { jekyllDataEmpty }
2672 \prop_gput:Nnn
2673 \g_@@_renderer_arities_prop
2674 { jekyllDataEmpty }
2675 { 1 }
2676 \ExplSyntaxOff

```

#### 2.2.5.44 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` key, whose value 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.

```
2677 \ExplSyntaxOn
2678 \cs_new:Nn \@@_define_renderers:
2679 {
2680 \seq_map_function:NN
2681 \g_@@_renderers_seq
2682 \@@_define_renderer:n
2683 }
2684 \cs_new:Nn \@@_define_renderer:n
2685 {
2686 \@@_renderer_tl_to_csname:nN
2687 { #1 }
2688 \l_tmpa_tl
2689 \prop_get:NnN
2690 \g_@@_renderer_arities_prop
2691 { #1 }
2692 \l_tmpb_tl
2693 \@@_define_renderer:ncV
2694 { #1 }
2695 { \l_tmpa_tl }
2696 \l_tmpb_tl
2697 }
2698 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2699 {
2700 \tl_set:Nn
2701 \l_tmpa_tl
2702 { \str_uppercase:n { #1 } }
2703 \tl_set:Nx
2704 #2
2705 {
2706 markdownRenderer
2707 \tl_head:f { \l_tmpa_tl }
2708 \tl_tail:n { #1 }
2709 }
2710 }
2711 \tl_new:N
2712 \l_@@_renderer_definition_tl
2713 \bool_new:N
2714 \g_@@_appending_renderer_bool
2715 \cs_new:Nn \@@_define_renderer:nNn
2716 {
```



```

2717 \keys_define:nn
2718 { markdown/options/renderers }
2719 {
2720 #1 .code:n = {
2721 \tl_set:Nn
2722 \l_@@_renderer_definition_tl
2723 { ##1 }
2724 \regex_replace_all:nnN
2725 { \cP\#0 }
2726 { #1 }
2727 \l_@@_renderer_definition_tl
2728 \bool_if:NT
2729 \g_@@_appending_renderer_bool
2730 {
2731 \@@_tl_set_from_cs:NNn
2732 \l_tmpa_tl
2733 #2
2734 { #3 }
2735 \tl_put_left:NV
2736 \l_@@_renderer_definition_tl
2737 \l_tmpa_tl
2738 }
2739 \cs_generate_from_arg_count:NNnV
2740 #2
2741 \cs_set:Npn
2742 { #3 }
2743 \l_@@_renderer_definition_tl
2744 },
2745 }
2746 }

```

We define the function `\@@_tl_set_from_cs:NNn` [9]. 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.

```

2747 \cs_new_protected:Nn
2748 \@@_tl_set_from_cs:NNn
2749 {
2750 \tl_set:Nn
2751 \l_tmpa_tl
2752 { #2 }
2753 \int_step_inline:nn
2754 { #3 }
2755 {
2756 \exp_args:Nnc
2757 \tl_put_right:Nn
2758 \l_tmpa_tl

```

```

2759 { @@_tl_set_from_cs_parameter_ ##1 }
2760 }
2761 \exp_args:NNV
2762 \tl_set:No
2763 \l_tmpb_tl
2764 \l_tmpa_tl
2765 \regex_replace_all:nnN
2766 { \cP. }
2767 { \0\0 }
2768 \l_tmpb_tl
2769 \int_step_inline:nn
2770 { #3 }
2771 {
2772 \regex_replace_all:nnN
2773 { \c { @@_tl_set_from_cs_parameter_ ##1 } }
2774 { \cP\# ##1 }
2775 \l_tmpb_tl
2776 }
2777 \tl_set:NV
2778 #1
2779 \l_tmpb_tl
2780 }
2781 \cs_generate_variant:Nn
2782 \@@_define_renderer:nNn
2783 { ncV }
2784 \cs_generate_variant:Nn
2785 \cs_generate_from_arg_count:NNnn
2786 { NNnV }
2787 \cs_generate_variant:Nn
2788 \tl_put_left:Nn
2789 { Nv }
2790 \keys_define:nn
2791 { markdown/options }
2792 {
2793 renderers .code:n = {
2794 \keys_set:nn
2795 { markdown/options/renderers }
2796 { #1 }
2797 },
2798 }

```

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.
 }
}

```

```

2799 \tl_new:N
2800 \l_@@_renderer_glob_definition_tl
2801 \seq_new:N
2802 \l_@@_renderer_glob_results_seq
2803 \regex_const:Nn
2804 \c_@@_appending_key_regex
2805 { \s*+$ }
2806 \keys_define:nn
2807 { markdown/options/renderers }
2808 {
2809 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 += {...},
 },
 }
 },
 },
}

```

```

2810 \regex_match:NVTF
2811 \c_@@_appending_key_regex
2812 \l_keys_key_str
2813 {
2814 \bool_gset_true:N
2815 \g_@@_appending_renderer_bool
2816 \tl_set:NV
2817 \l_tmpa_tl
2818 \l_keys_key_str
2819 \regex_replace_once:NnN
2820 \c_@@_appending_key_regex
2821 { }
2822 \l_tmpa_tl
2823 \tl_set:Nx
2824 \l_tmpb_tl
2825 { { \l_tmpa_tl } = }
2826 \tl_put_right:Nn
2827 \l_tmpb_tl
2828 { { #1 } }
2829 \keys_set:nV
2830 { markdown/options/renderers }
2831 \l_tmpb_tl
2832 \bool_gset_false:N
2833 \g_@@_appending_renderer_bool
2834 }

```

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.
 }
}
```

```
2835 {
2836 \@@_glob_seq:VnN
2837 \l_keys_key_str
2838 { g_@@_renderers_seq }
2839 \l_@@_renderer_glob_results_seq
2840 \seq_if_empty:NTF
2841 \l_@@_renderer_glob_results_seq
2842 {
2843 \msg_error:nnV
2844 { markdown }
2845 { undefined-renderer }
2846 \l_keys_key_str
2847 }
2848 {
2849 \tl_set:Nn
2850 \l_@@_renderer_glob_definition_tl
2851 { \exp_not:n { #1 } }
2852 \seq_map_inline:Nn
2853 \l_@@_renderer_glob_results_seq
2854 {
2855 \tl_set:Nn
2856 \l_tmpa_tl
2857 { { ##1 } = }
2858 \tl_put_right:Nx
2859 \l_tmpa_tl
2860 { { \l_@@_renderer_glob_definition_tl } }
2861 \keys_set:nV
2862 { markdown/options/renderers }
2863 \l_tmpa_tl
2864 }
2865 }
2866 },
2867 },
2868 }
2869 \msg_new:nnn
2870 { markdown }
2871 { undefined-renderer }
2872 {
```

```

2873 Renderer~#1~is~undefined.
2874 }
2875 \cs_generate_variant:Nn
2876 \@@_glob_seq:nnN
2877 { VnN }
2878 \cs_generate_variant:Nn
2879 \cs_generate_from_arg_count:NNnn
2880 { cNVV }
2881 \cs_generate_variant:Nn
2882 \msg_error:nnn
2883 { nnV }
2884 \prg_generate_conditional_variant:Nnn
2885 \regex_match:Nn
2886 { NV }
2887 { TF }
2888 \prop_new:N
2889 \g_@@_glob_cache_prop
2890 \tl_new:N
2891 \l_@@_current_glob_tl
2892 \cs_new:Nn
2893 \@@_glob_seq:nnN
2894 {
2895 \tl_set:Nn
2896 \l_@@_current_glob_tl
2897 { ~ #1 $ }
2898 \prop_get:NeNTF
2899 \g_@@_glob_cache_prop
2900 { #2 / \l_@@_current_glob_tl }
2901 \l_tmpa_clist
2902 {
2903 \seq_set_from_clist:NN
2904 #3
2905 \l_tmpa_clist
2906 }
2907 {
2908 \seq_clear:N
2909 #3
2910 \regex_replace_all:nnN
2911 { * }
2912 { .* }
2913 \l_@@_current_glob_tl
2914 \regex_set:NV
2915 \l_tmpa_regex
2916 \l_@@_current_glob_tl
2917 \seq_map_inline:cn
2918 { #2 }
2919 {

```

```

2920 \regex_match:NnT
2921 \l_tmpa_regex
2922 { ##1 }
2923 {
2924 \seq_put_right:Nn
2925 #3
2926 { ##1 }
2927 }
2928 }
2929 \clist_set_from_seq:NN
2930 \l_tmpa_clist
2931 #3
2932 \prop_gput:NeV
2933 \g_@@_glob_cache_prop
2934 { #2 / \l_@@_current_glob_tl }
2935 \l_tmpa_clist
2936 }
2937 }
2938 % TODO: Remove in TeX Live 2023.
2939 \prg_generate_conditional_variant:Nnn
2940 \prop_get:NnN
2941 { NeN }
2942 { TF }
2943 \cs_generate_variant:Nn
2944 \regex_set:Nn
2945 { NV }
2946 \cs_generate_variant:Nn
2947 \prop_gput:Nnn
2948 { NeV }

```

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

```

2949 \str_if_eq:VVT
2950 \c_@@_top_layer_tl
2951 \c_@@_option_layer_plain_tex_tl
2952 {
2953 \@@_define_renderers:
2954 }
2955 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the `LATEX3` kernel.

```
2956 \ExplSyntaxOn
2957 \keys_define:nn
2958 { markdown/jekyllData }
2959 { }
2960 \ExplSyntaxOff
```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key-values without using the `expl3` language.

```
2961 \ExplSyntaxOn
2962 \@@_with_various_cases:nn
2963 { jekyllDataRenderers }
2964 {
2965 \keys_define:nn
2966 { markdown/options }
2967 {
2968 #1 .code:n = {
2969 \tl_set:Nn
2970 \l_tmpa_tl
2971 { ##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.

```
2972 \tl_replace_all:NnV
2973 \l_tmpa_tl
2974 { / }
2975 \c_backslash_str
2976 \keys_set:nV
2977 { markdown/options/jekyll-data-renderers }
2978 \l_tmpa_tl
2979 },
2980 }
2981 }
2982 \keys_define:nn
2983 { markdown/options/jekyll-data-renderers }
2984 {
2985 unknown .code:n = {
2986 \tl_set_eq:NN
2987 \l_tmpa_tl
2988 \l_keys_key_str
2989 \tl_replace_all:NVn
2990 \l_tmpa_tl
```



```

2991 \c_backslash_str
2992 { / }
2993 \tl_put_right:Nn
2994 \l_tmpa_tl
2995 {
2996 .code:n = { #1 }
2997 }
2998 \keys_define:nV
2999 { markdown/jekyllData }
3000 \l_tmpa_tl
3001 }
3002 }
3003 \cs_generate_variant:Nn
3004 \keys_define:nn
3005 { nV }
3006 \ExplSyntaxOff

```

### 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 `rendererPrototype` key, whose value 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.

```

3007 \ExplSyntaxOn
3008 \cs_new:Nn \@@_define_renderer_prototypes:
3009 {
3010 \seq_map_function:NN
3011 \g_@@_renderers_seq
3012 \@@_define_renderer_prototype:n
3013 }
3014 \cs_new:Nn \@@_define_renderer_prototype:n
3015 {
3016 \@@_renderer_prototype_tl_to_csname:nN
3017 { #1 }
3018 \l_tmpa_tl
3019 \prop_get:NnN
3020 \g_@@_renderer_arities_prop
3021 { #1 }
3022 \l_tmpb_tl
3023 \@@_define_renderer_prototype:ncV
3024 { #1 }
3025 { \l_tmpa_tl }
3026 \l_tmpb_tl
3027 }
3028 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN

```

```

3029 {
3030 \tl_set:Nn
3031 \l_tmpa_tl
3032 { \str_uppercase:n { #1 } }
3033 \tl_set:Nx
3034 #2
3035 {
3036 markdownRenderer
3037 \tl_head:f { \l_tmpa_tl }
3038 \tl_tail:n { #1 }
3039 Prototype
3040 }
3041 }
3042 \tl_new:N
3043 \l_@@_renderer_prototype_definition_tl
3044 \bool_new:N
3045 \g_@@_appending_renderer_prototype_bool
3046 \cs_new:Nn \@@_define_renderer_prototype:nNn
3047 {
3048 \keys_define:nn
3049 { markdown/options/renderer-prototypes }
3050 {
3051 #1 .code:n = {
3052 \tl_set:Nn
3053 \l_@@_renderer_prototype_definition_tl
3054 { ##1 }
3055 \regex_replace_all:nnN
3056 { \cP\#0 }
3057 { #1 }
3058 \l_@@_renderer_prototype_definition_tl
3059 \bool_if:NT
3060 \g_@@_appending_renderer_prototype_bool
3061 {
3062 \@@_tl_set_from_cs:NNn
3063 \l_tmpa_tl
3064 #2
3065 { #3 }
3066 \tl_put_left:NV
3067 \l_@@_renderer_prototype_definition_tl
3068 \l_tmpa_tl
3069 }
3070 \cs_generate_from_arg_count:NNnV
3071 #2
3072 \cs_set:Npn
3073 { #3 }
3074 \l_@@_renderer_prototype_definition_tl
3075 },

```

```
3076 }
```

Unless the token renderer prototype macro has already been defined, we provide an empty definition.

```
3077 \cs_if_free:NT
3078 #2
3079 {
3080 \cs_generate_from_arg_count:NNnn
3081 #2
3082 \cs_set:Npn
3083 { #3 }
3084 { }
3085 }
3086 }
3087 \cs_generate_variant:Nn
3088 \@@_define_renderer_prototype:nNn
3089 { 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.
 }
}
```

```
3090 \keys_define:nn
3091 { markdown/options/renderer-prototypes }
3092 {
3093 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 += {...},
 },
 },
},
}

```

```

3094 \regex_match:NVTF
3095 \c_@@_appending_key_regex
3096 \l_keys_key_str
3097 {
3098 \bool_gset_true:N
3099 \g_@@_appending_renderer_prototype_bool
3100 \tl_set:NV
3101 \l_tmpa_tl
3102 \l_keys_key_str
3103 \regex_replace_once:NnN
3104 \c_@@_appending_key_regex
3105 { }
3106 \l_tmpa_tl
3107 \tl_set:Nx
3108 \l_tmpb_tl
3109 { { \l_tmpa_tl } = }
3110 \tl_put_right:Nn
3111 \l_tmpb_tl
3112 { { #1 } }
3113 \keys_set:nV
3114 { markdown/options/renderer-prototypes }
3115 \l_tmpb_tl
3116 \bool_gset_false:N
3117 \g_@@_appending_renderer_prototype_bool
3118 }

```

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.
}
```

```
3119 {
3120 \@_glob_seq:VnN
3121 \l_keys_key_str
3122 { g_@_renderers_seq }
3123 \l_@_renderer_glob_results_seq
3124 \seq_if_empty:NTF
3125 \l_@_renderer_glob_results_seq
3126 {
3127 \msg_error:nnV
3128 { markdown }
3129 { undefined-renderer-prototype }
3130 \l_keys_key_str
3131 }
```

```

3132 {
3133 \tl_set:Nn
3134 \l_@@_renderer_glob_definition_tl
3135 { \exp_not:n { #1 } }
3136 \seq_map_inline:Nn
3137 \l_@@_renderer_glob_results_seq
3138 {
3139 \tl_set:Nn
3140 \l_tmpa_tl
3141 { { ##1 } = }
3142 \tl_put_right:Nx
3143 \l_tmpa_tl
3144 { { \l_@@_renderer_glob_definition_tl } }
3145 \keys_set:nV
3146 { markdown/options/renderer-prototypes }
3147 \l_tmpa_tl
3148 }
3149 }
3150 }
3151 },
3152 }
3153 \msg_new:nnn
3154 { markdown }
3155 { undefined-renderer-prototype }
3156 {
3157 Renderer~prototype~#1~is~undefined.
3158 }
3159 \@@_with_various_cases:nn
3160 { rendererPrototypes }
3161 {
3162 \keys_define:nn
3163 { markdown/options }
3164 {
3165 #1 .code:n = {
3166 \keys_set:nn
3167 { markdown/options/renderer-prototypes }
3168 { ##1 }
3169 },
3170 }
3171 }

```

If plain T<sub>E</sub>X is the top layer, we use the `\@@_define_renderer_prototypes:` macro to define plain T<sub>E</sub>X token renderer prototype macros and key-values immediately. Otherwise, we postpone the definition until the upper layers have been loaded.

```

3172 \str_if_eq:VVT
3173 \c_@@_top_layer_tl
3174 \c_@@_option_layer_plain_tex_tl

```

```

3175 {
3176 \@@_define_renderer_prototypes:
3177 }
3178 \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  $\TeX$  engine that does not support direct Lua access is starting to buffer a text. The plain  $\TeX$  implementation changes the category code of plain  $\TeX$  special characters to other, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```

3179 \let\markdownMakeOther\relax

```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain  $\TeX$  special characters have had their category changed to *other*. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```

3180 \let\markdownReadAndConvert\relax
3181 \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*.

```

3182 \catcode`\|=0\catcode`\=12%
3183 |gdef|markdownBegin{%
3184 |markdownReadAndConvert{\markdownEnd}%
3185 {|\markdownEnd}}%
3186 |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.

```
3187 \ExplSyntaxOn
3188 \keys_define:nn
3189 { markdown/options }
3190 {
3191 code .code:n = { #1 },
3192 }
3193 \ExplSyntaxOff
```

This can be especially useful in snippets.

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

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

```
3194 \ExplSyntaxOn
3195 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3196 \cs_generate_variant:Nn
3197 \tl_const:Nn
3198 { NV }
3199 \tl_if_exist:NF
3200 \c_@@_top_layer_tl
3201 {
3202 \tl_const:NV
3203 \c_@@_top_layer_tl
3204 \c_@@_option_layer_latex_tl
3205 }
3206 \ExplSyntaxOff
3207 \input markdown/markdown
```

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

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

where `<options>` are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2). Note that `<options>` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.44) 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 2.3.3) 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` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments, and redefines the `\markinline` and `\markdownInput` commands.

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. Both L<sup>A</sup>T<sub>E</sub>X environments accept L<sup>A</sup>T<sub>E</sub>X interface options (see section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

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

```
3208 \newenvironment{markdown}\relax\relax
3209 \newenvironment{markdown*}[1]\relax\relax
```

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

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

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

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

The `\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 2.3.2) as its optional argument. These options will only influence this markdown content.

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  $\text{T}_{\text{E}}\text{X}$  interface, this macro also accepts  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

The following example  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  code showcases the usage of the `\markdownInput` macro:

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

## 2.3.2 Options

The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  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.

$\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  options map directly to the options recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\text{T}_{\text{E}}\text{X}$  interface (see Sections 2.2.5 and 2.2.6).

The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  options may be specified when loading the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  package, when using the `markdown*`  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

### 2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\text{T}_{\text{E}}\text{X}$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

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

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  document sources for distribution.

```
3210 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3211 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

### 2.3.2.2 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.

```
3212 \ExplSyntaxOn
3213 \str_if_eq:VVT
3214 \c_@@_top_layer_tl
3215 \c_@@_option_layer_latex_tl
3216 {
3217 \@@_define_option_commands_and_keyvals:
3218 \@@_define_renderers:
3219 \@@_define_renderer_prototypes:
3220 }
3221 \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,
}
```

### 2.3.3 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 L<sup>A</sup>T<sub>E</sub>X package has been loaded. Otherwise,

the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` L<sup>A</sup>T<sub>E</sub>X package, and finally the `markdownthemewitiko_dot.sty` L<sup>A</sup>T<sub>E</sub>X package:

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

```
3222 \newif\ifmarkdownLaTeXLoaded
3223 \markdownLaTeXLoadedfalse
```

Due to limitations of L<sup>A</sup>T<sub>E</sub>X, themes may not be loaded after the beginning of a L<sup>A</sup>T<sub>E</sub>X document.

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

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

```
\documentclass{article}
\usepackage[import=witiko/dot]{markdown}
\setkeys{Gin}{
 width = \columnwidth,
 height = 0.65\paperheight,
 keepaspectratio}
\begin{document}
\begin{markdown}
``` dot Various formats of mathematical formulae
digraph tree {
  margin = 0;
  rankdir = "LR";

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

```

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

```

Typesetting the above document produces the output shown in Figure 4.

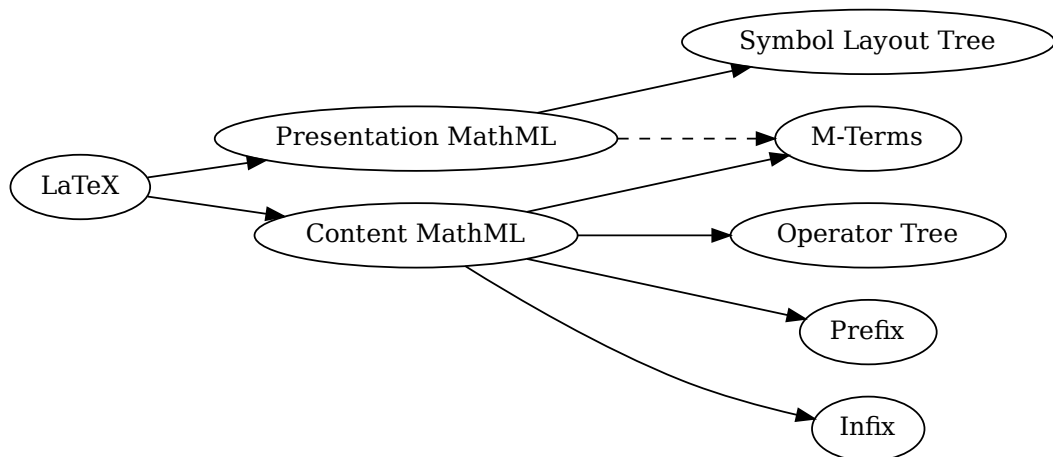


Figure 4: Various formats of mathematical formulae

The theme requires a Unix-like operating system with GNU Diffutils and Graphviz installed. The theme also requires shell access unless the `frozenCache` plain \TeX option is enabled.

3224 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%

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

```

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

```

```

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

```

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

```

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

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

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

```



Chapter 1

Introduction

1.1 Section
 1.1.1 Subsection
 Hello *Markdown!*

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

Table 1.1: Table

Figure 5: The banner of the Markdown package

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

```
3225 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

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.

```
3226 \AtEndOfPackage{
3227   \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).

```

3228 \markdownIfOption{noDefaults}{-}{
3229     \markdownSetup{theme=witiko/markdown/defaults}
3230 }
3231 }

3232 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%

```

Please, see Section 3.3.3 for implementation details of the built-in \LaTeX themes.

2.4 ConTeXt Interface

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

```

3233 \ExplSyntaxOn
3234 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3235 \cs_generate_variant:Nn
3236   \tl_const:Nn
3237   { NV }
3238 \tl_if_exist:NF
3239   \c_@@_top_layer_tl
3240   {
3241     \tl_const:NV
3242       \c_@@_top_layer_tl
3243       \c_@@_option_layer_context_tl
3244   }
3245 \ExplSyntaxOff

```

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

```

3246 \writestatus{loading}{ConTeXt User Module / markdown}%
3247 \startmodule[markdown]
3248 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3249   \do\#\do\^\do\_do\%do\~}%
3250 \input markdown/markdown

```

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

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

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

2.4.1 Typesetting Markdown

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

```
3251 \let\startmarkdown\relax
3252 \let\stopmarkdown\relax
3253 \let\inputmarkdown\relax
```

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

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

The following example Con \TeX t code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

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

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

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

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

2.4.2 Options

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

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}`.

```

3254 \ExplSyntaxOn
3255 \cs_new:Npn
3256   \setupmarkdown
3257   [ #1 ]
3258   {
3259     \@@_setup:n
3260     { #1 }
3261   }
3262 \ExplSyntaxOff

```

2.4.2.1 Generating Plain TeX Option Macros and Key-Values

Unlike plain TeX, we also accept caseless variants of options in line with the style of ConTeXt.

```

3263 \ExplSyntaxOn
3264 \cs_new:Nn \@@_caseless:N
3265   {
3266     \regex_replace_all:nnN
3267     { ([a-z])([A-Z]) }
3268     { \1 \c { str_lowercase:n } \cB{\ \2 \cE\} }
3269     #1
3270     \tl_set:Nx
3271     #1
3272     { #1 }
3273   }
3274 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3275 \str_if_eq:VVT
3276   \c_@@_top_layer_tl
3277   \c_@@_option_layer_context_tl
3278   {
3279     \@@_define_option_commands_and_keyvals:
3280     \@@_define_renderers:
3281     \@@_define_renderer_prototypes:
3282   }
3283 \ExplSyntaxOff

```

2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConTeXt, we expand on the concept of themes by allowing a theme to be a full-blown ConTeXt module. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a ConTeXt module named `t-markdowntheme<munged theme name>.tex` if it exists and a TeX 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]
```

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.

```
3284 \startmodule[markdownthemewitiko_markdown_defaults]
3285 \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^AT_EX 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_EX, 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_EX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3286 local upper, format, length =
3287   string.upper, string.format, string.len
3288 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3289   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3290   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the plain T_EX 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.

```
3291 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.

```
3292 function util.err(msg, exit_code)
3293   io.stderr:write("markdown.lua: " .. msg .. "\n")
3294   os.exit(exit_code or 1)
3295 end
```

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

```
3296 function util.cache(dir, string, salt, transform, suffix)
3297   local digest = md5.sumhexa(string .. (salt or ""))
3298   local name = util.pathname(dir, digest .. suffix)
3299   local file = io.open(name, "r")
3300   if file == nil then -- If no cache entry exists, then create a new one.
3301     file = assert(io.open(name, "w"),
3302       [[Could not open file ]] .. name .. [[ for writing]])
3303     local result = string
3304     if transform ~= nil then
3305       result = transform(result)
3306     end
3307     assert(file:write(result))
```

```

3308     assert(file:close())
3309   end
3310   return name
3311 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.

```

3312 function util.cache_verbatim(dir, string)
3313   local name = util.cache(dir, string, nil, nil, ".verbatim")
3314   return name
3315 end

```

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

```

3316 function util.table_copy(t)
3317   local u = { }
3318   for k, v in pairs(t) do u[k] = v end
3319   return setmetatable(u, getmetatable(t))
3320 end

```

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

```

3321 function util.encode_json_string(s)
3322   s = s:gsub([[\\]], [[\\]])
3323   s = s:gsub([[\"]], [[\"]])
3324   return [[\"]] .. s .. [[\"]]
3325 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [11, Chapter 21].

```

3326 function util.expand_tabs_in_line(s, tabstop)
3327   local tab = tabstop or 4
3328   local corr = 0
3329   return (s:gsub(")\t", function(p)
3330     local sp = tab - (p - 1 + corr) % tab
3331     corr = corr - 1 + sp
3332     return string.rep(" ", sp)
3333   end))
3334 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.

```

3335 function util.walk(t, f)
3336   local typ = type(t)
3337   if typ == "string" then

```

```

3338     f(t)
3339 elseif typ == "table" then
3340     local i = 1
3341     local n
3342     n = t[i]
3343     while n do
3344         util.walk(n, f)
3345         i = i + 1
3346         n = t[i]
3347     end
3348 elseif typ == "function" then
3349     local ok, val = pcall(t)
3350     if ok then
3351         util.walk(val, f)
3352     end
3353 else
3354     f(tostring(t))
3355 end
3356 end

```

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

```

3357 function util.flatten(ary)
3358     local new = {}
3359     for _,v in ipairs(ary) do
3360         if type(v) == "table" then
3361             for _,w in ipairs(util.flatten(v)) do
3362                 new[#new + 1] = w
3363             end
3364         else
3365             new[#new + 1] = v
3366         end
3367     end
3368     return new
3369 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.

```

3370 function util.rope_to_string(rope)
3371     local buffer = {}
3372     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3373     return table.concat(buffer)
3374 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.

```

3375 function util.rope_last(rope)
3376     if #rope == 0 then

```

```

3377     return nil
3378   else
3379     local l = rope[#rope]
3380     if type(l) == "table" then
3381       return util.rope_last(l)
3382     else
3383       return l
3384     end
3385   end
3386 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 $1 \leq i \leq \#ary$.

```

3387 function util.intersperse(ary, x)
3388   local new = {}
3389   local l = #ary
3390   for i,v in ipairs(ary) do
3391     local n = #new
3392     new[n + 1] = v
3393     if i ~= l then
3394       new[n + 2] = x
3395     end
3396   end
3397   return new
3398 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 $1 \leq i \leq \#ary$.

```

3399 function util.map(ary, f)
3400   local new = {}
3401   for i,v in ipairs(ary) do
3402     new[i] = f(v)
3403   end
3404   return new
3405 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.

```

3406 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

3407   local char_escapes_list = ""
3408   for i,_ in pairs(char_escapes) do
3409     char_escapes_list = char_escapes_list .. i

```

```
3410 end
```

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

```
3411 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.

```
3412 if string_escapes then
3413   for k,v in pairs(string_escapes) do
3414     escapable = P(k) / v + escapable
3415   end
3416 end
```

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

```
3417 local escape_string = Cs((escapable + any)^0)
```

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

```
3418 return function(s)
3419   return lpeg.match(escape_string, s)
3420 end
3421 end
```

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

```
3422 function util.pathname(dir, file)
3423   if #dir == 0 then
3424     return file
3425   else
3426     return dir .. "/" .. file
3427   end
3428 end
```

3.1.2 HTML Entities

This section documents the HTML entities recognized by the markdown reader. These functions are encapsulated in the `entities` object. The functions were originally located in the `lunamark/entities.lua` file in the Lunamark Lua module.

```
3429 local entities = {}
```

```
3430
3431 local character_entities = {
3432     ["Tab"] = 9,
3433     ["NewLine"] = 10,
3434     ["excl"] = 33,
3435     ["QUOT"] = 34,
3436     ["quot"] = 34,
3437     ["num"] = 35,
3438     ["dollar"] = 36,
3439     ["percent"] = 37,
3440     ["AMP"] = 38,
3441     ["amp"] = 38,
3442     ["apos"] = 39,
3443     ["lpar"] = 40,
3444     ["rpar"] = 41,
3445     ["ast"] = 42,
3446     ["midast"] = 42,
3447     ["plus"] = 43,
3448     ["comma"] = 44,
3449     ["period"] = 46,
3450     ["sol"] = 47,
3451     ["colon"] = 58,
3452     ["semi"] = 59,
3453     ["LT"] = 60,
3454     ["lt"] = 60,
3455     ["nvlT"] = {60, 8402},
3456     ["bne"] = {61, 8421},
3457     ["equals"] = 61,
3458     ["GT"] = 62,
3459     ["gt"] = 62,
3460     ["nvgt"] = {62, 8402},
3461     ["quest"] = 63,
3462     ["commat"] = 64,
3463     ["lbrack"] = 91,
3464     ["lsqb"] = 91,
3465     ["bsol"] = 92,
3466     ["rbrack"] = 93,
3467     ["rsqb"] = 93,
3468     ["Hat"] = 94,
3469     ["UnderBar"] = 95,
3470     ["lowbar"] = 95,
3471     ["DiacriticalGrave"] = 96,
3472     ["grave"] = 96,
3473     ["fjlig"] = {102, 106},
3474     ["lbrace"] = 123,
3475     ["lcub"] = 123,
3476     ["VerticalLine"] = 124,
```


3477 ["verbar"] = 124,
 3478 ["vert"] = 124,
 3479 ["rbrace"] = 125,
 3480 ["rcub"] = 125,
 3481 ["NonBreakingSpace"] = 160,
 3482 ["nbsp"] = 160,
 3483 ["iexcl"] = 161,
 3484 ["cent"] = 162,
 3485 ["pound"] = 163,
 3486 ["curren"] = 164,
 3487 ["yen"] = 165,
 3488 ["brvbar"] = 166,
 3489 ["sect"] = 167,
 3490 ["Dot"] = 168,
 3491 ["DoubleDot"] = 168,
 3492 ["die"] = 168,
 3493 ["uml"] = 168,
 3494 ["COPY"] = 169,
 3495 ["copy"] = 169,
 3496 ["ordf"] = 170,
 3497 ["laquo"] = 171,
 3498 ["not"] = 172,
 3499 ["shy"] = 173,
 3500 ["REG"] = 174,
 3501 ["circledR"] = 174,
 3502 ["reg"] = 174,
 3503 ["macr"] = 175,
 3504 ["strns"] = 175,
 3505 ["deg"] = 176,
 3506 ["PlusMinus"] = 177,
 3507 ["plusmn"] = 177,
 3508 ["pm"] = 177,
 3509 ["sup2"] = 178,
 3510 ["sup3"] = 179,
 3511 ["DiacriticalAcute"] = 180,
 3512 ["acute"] = 180,
 3513 ["micro"] = 181,
 3514 ["para"] = 182,
 3515 ["CenterDot"] = 183,
 3516 ["centerdot"] = 183,
 3517 ["middot"] = 183,
 3518 ["Cedilla"] = 184,
 3519 ["cedil"] = 184,
 3520 ["sup1"] = 185,
 3521 ["ordm"] = 186,
 3522 ["raquo"] = 187,
 3523 ["frac14"] = 188,

3524 ["frac12"] = 189,
3525 ["half"] = 189,
3526 ["frac34"] = 190,
3527 ["iquest"] = 191,
3528 ["Agrave"] = 192,
3529 ["Aacute"] = 193,
3530 ["Acirc"] = 194,
3531 ["Atilde"] = 195,
3532 ["Auml"] = 196,
3533 ["Aring"] = 197,
3534 ["angst"] = 197,
3535 ["AElig"] = 198,
3536 ["Ccedil"] = 199,
3537 ["Egrave"] = 200,
3538 ["Eacute"] = 201,
3539 ["Ecirc"] = 202,
3540 ["Euml"] = 203,
3541 ["Igrave"] = 204,
3542 ["Iacute"] = 205,
3543 ["Icirc"] = 206,
3544 ["Iuml"] = 207,
3545 ["ETH"] = 208,
3546 ["Ntilde"] = 209,
3547 ["Ograve"] = 210,
3548 ["Oacute"] = 211,
3549 ["Ocirc"] = 212,
3550 ["Otilde"] = 213,
3551 ["Ouml"] = 214,
3552 ["times"] = 215,
3553 ["Oslash"] = 216,
3554 ["Ugrave"] = 217,
3555 ["Uacute"] = 218,
3556 ["Ucirc"] = 219,
3557 ["Uuml"] = 220,
3558 ["Yacute"] = 221,
3559 ["THORN"] = 222,
3560 ["szlig"] = 223,
3561 ["agrave"] = 224,
3562 ["aacute"] = 225,
3563 ["acirc"] = 226,
3564 ["atilde"] = 227,
3565 ["auml"] = 228,
3566 ["aring"] = 229,
3567 ["aelig"] = 230,
3568 ["ccedil"] = 231,
3569 ["egrave"] = 232,
3570 ["eacute"] = 233,

3571 ["ecirc"] = 234,
3572 ["euml"] = 235,
3573 ["igrave"] = 236,
3574 ["iacute"] = 237,
3575 ["icirc"] = 238,
3576 ["iuml"] = 239,
3577 ["eth"] = 240,
3578 ["ntilde"] = 241,
3579 ["ograve"] = 242,
3580 ["oacute"] = 243,
3581 ["ocirc"] = 244,
3582 ["otilde"] = 245,
3583 ["ouml"] = 246,
3584 ["div"] = 247,
3585 ["divide"] = 247,
3586 ["oslash"] = 248,
3587 ["ugrave"] = 249,
3588 ["uacute"] = 250,
3589 ["ucirc"] = 251,
3590 ["uuml"] = 252,
3591 ["yacute"] = 253,
3592 ["thorn"] = 254,
3593 ["yuml"] = 255,
3594 ["Amacr"] = 256,
3595 ["amacr"] = 257,
3596 ["Abreve"] = 258,
3597 ["abreve"] = 259,
3598 ["Aogon"] = 260,
3599 ["aogon"] = 261,
3600 ["Cacute"] = 262,
3601 ["cacute"] = 263,
3602 ["Ccirc"] = 264,
3603 ["ccirc"] = 265,
3604 ["Cdot"] = 266,
3605 ["cdot"] = 267,
3606 ["Ccaron"] = 268,
3607 ["ccaron"] = 269,
3608 ["Dcaron"] = 270,
3609 ["dcaron"] = 271,
3610 ["Dstrok"] = 272,
3611 ["dstrok"] = 273,
3612 ["Emacr"] = 274,
3613 ["emacr"] = 275,
3614 ["Edot"] = 278,
3615 ["edot"] = 279,
3616 ["Eogon"] = 280,
3617 ["eogon"] = 281,

3618 ["Ecaron"] = 282,
3619 ["ecaron"] = 283,
3620 ["Gcirc"] = 284,
3621 ["gcirc"] = 285,
3622 ["Gbreve"] = 286,
3623 ["gbreve"] = 287,
3624 ["Gdot"] = 288,
3625 ["gdot"] = 289,
3626 ["Gcedil"] = 290,
3627 ["Hcirc"] = 292,
3628 ["hcirc"] = 293,
3629 ["Hstrook"] = 294,
3630 ["hstrook"] = 295,
3631 ["Itilde"] = 296,
3632 ["itilde"] = 297,
3633 ["Imacr"] = 298,
3634 ["imacr"] = 299,
3635 ["Iogon"] = 302,
3636 ["iogon"] = 303,
3637 ["Idot"] = 304,
3638 ["imath"] = 305,
3639 ["inodot"] = 305,
3640 ["IJlig"] = 306,
3641 ["ijlig"] = 307,
3642 ["Jcirc"] = 308,
3643 ["jcirc"] = 309,
3644 ["Kcedil"] = 310,
3645 ["kcedil"] = 311,
3646 ["kgreen"] = 312,
3647 ["Lacute"] = 313,
3648 ["lacute"] = 314,
3649 ["Lcedil"] = 315,
3650 ["lcedil"] = 316,
3651 ["Lcaron"] = 317,
3652 ["lcaron"] = 318,
3653 ["Lmidot"] = 319,
3654 ["lmidot"] = 320,
3655 ["Lstrook"] = 321,
3656 ["lstrook"] = 322,
3657 ["Nacute"] = 323,
3658 ["nacute"] = 324,
3659 ["Ncedil"] = 325,
3660 ["ncedil"] = 326,
3661 ["Ncaron"] = 327,
3662 ["ncaron"] = 328,
3663 ["napos"] = 329,
3664 ["ENG"] = 330,

3665 ["eng"] = 331,
3666 ["Omacr"] = 332,
3667 ["omacr"] = 333,
3668 ["Odblac"] = 336,
3669 ["odblac"] = 337,
3670 ["OElig"] = 338,
3671 ["oelig"] = 339,
3672 ["Racute"] = 340,
3673 ["racute"] = 341,
3674 ["Rcedil"] = 342,
3675 ["rcedil"] = 343,
3676 ["Rcaron"] = 344,
3677 ["rcaron"] = 345,
3678 ["Sacute"] = 346,
3679 ["sacute"] = 347,
3680 ["Scirc"] = 348,
3681 ["scirc"] = 349,
3682 ["Scedil"] = 350,
3683 ["scedil"] = 351,
3684 ["Scaron"] = 352,
3685 ["scaron"] = 353,
3686 ["Tcedil"] = 354,
3687 ["tcedil"] = 355,
3688 ["Tcaron"] = 356,
3689 ["tcaron"] = 357,
3690 ["Tstrok"] = 358,
3691 ["tstrok"] = 359,
3692 ["Utilde"] = 360,
3693 ["utilde"] = 361,
3694 ["Umacr"] = 362,
3695 ["umacr"] = 363,
3696 ["Ubreve"] = 364,
3697 ["ubreve"] = 365,
3698 ["Uring"] = 366,
3699 ["uring"] = 367,
3700 ["Udblac"] = 368,
3701 ["udblac"] = 369,
3702 ["Uogon"] = 370,
3703 ["uogon"] = 371,
3704 ["Wcirc"] = 372,
3705 ["wcirc"] = 373,
3706 ["Ycirc"] = 374,
3707 ["ycirc"] = 375,
3708 ["Yuml"] = 376,
3709 ["Zacute"] = 377,
3710 ["zacute"] = 378,
3711 ["Zdot"] = 379,

3712 ["zdot"] = 380,
3713 ["Zcaron"] = 381,
3714 ["zcaron"] = 382,
3715 ["fnof"] = 402,
3716 ["imped"] = 437,
3717 ["gacute"] = 501,
3718 ["jmath"] = 567,
3719 ["circ"] = 710,
3720 ["Hacek"] = 711,
3721 ["caron"] = 711,
3722 ["Breve"] = 728,
3723 ["breve"] = 728,
3724 ["DiacriticalDot"] = 729,
3725 ["dot"] = 729,
3726 ["ring"] = 730,
3727 ["ogon"] = 731,
3728 ["DiacriticalTilde"] = 732,
3729 ["tilde"] = 732,
3730 ["DiacriticalDoubleAcute"] = 733,
3731 ["dblac"] = 733,
3732 ["DownBreve"] = 785,
3733 ["Alpha"] = 913,
3734 ["Beta"] = 914,
3735 ["Gamma"] = 915,
3736 ["Delta"] = 916,
3737 ["Epsilon"] = 917,
3738 ["Zeta"] = 918,
3739 ["Eta"] = 919,
3740 ["Theta"] = 920,
3741 ["Iota"] = 921,
3742 ["Kappa"] = 922,
3743 ["Lambda"] = 923,
3744 ["Mu"] = 924,
3745 ["Nu"] = 925,
3746 ["Xi"] = 926,
3747 ["Omicron"] = 927,
3748 ["Pi"] = 928,
3749 ["Rho"] = 929,
3750 ["Sigma"] = 931,
3751 ["Tau"] = 932,
3752 ["Upsilon"] = 933,
3753 ["Phi"] = 934,
3754 ["Chi"] = 935,
3755 ["Psi"] = 936,
3756 ["Omega"] = 937,
3757 ["ohm"] = 937,
3758 ["alpha"] = 945,

3759 ["beta"] = 946,
3760 ["gamma"] = 947,
3761 ["delta"] = 948,
3762 ["epsi"] = 949,
3763 ["epsilon"] = 949,
3764 ["zeta"] = 950,
3765 ["eta"] = 951,
3766 ["theta"] = 952,
3767 ["iota"] = 953,
3768 ["kappa"] = 954,
3769 ["lambda"] = 955,
3770 ["mu"] = 956,
3771 ["nu"] = 957,
3772 ["xi"] = 958,
3773 ["omicron"] = 959,
3774 ["pi"] = 960,
3775 ["rho"] = 961,
3776 ["sigmaf"] = 962,
3777 ["sigmav"] = 962,
3778 ["varsigma"] = 962,
3779 ["sigma"] = 963,
3780 ["tau"] = 964,
3781 ["upsilon"] = 965,
3782 ["upsilon"] = 965,
3783 ["phi"] = 966,
3784 ["chi"] = 967,
3785 ["psi"] = 968,
3786 ["omega"] = 969,
3787 ["thetasym"] = 977,
3788 ["thetav"] = 977,
3789 ["vartheta"] = 977,
3790 ["Upsilon"] = 978,
3791 ["upsih"] = 978,
3792 ["phiv"] = 981,
3793 ["straightphi"] = 981,
3794 ["varphi"] = 981,
3795 ["piv"] = 982,
3796 ["varpi"] = 982,
3797 ["Gammad"] = 988,
3798 ["digamma"] = 989,
3799 ["gammad"] = 989,
3800 ["kappav"] = 1008,
3801 ["varkappa"] = 1008,
3802 ["rhov"] = 1009,
3803 ["varrho"] = 1009,
3804 ["epsiv"] = 1013,
3805 ["straightepsilon"] = 1013,

3806 ["varepsilon"] = 1013,
3807 ["backepsilon"] = 1014,
3808 ["bepsi"] = 1014,
3809 ["IOcy"] = 1025,
3810 ["DJcy"] = 1026,
3811 ["GJcy"] = 1027,
3812 ["Jukcy"] = 1028,
3813 ["DScy"] = 1029,
3814 ["Iukcy"] = 1030,
3815 ["YIcy"] = 1031,
3816 ["Jsercy"] = 1032,
3817 ["LJcy"] = 1033,
3818 ["NJcy"] = 1034,
3819 ["TSHcy"] = 1035,
3820 ["KJcy"] = 1036,
3821 ["Ubrcy"] = 1038,
3822 ["DZcy"] = 1039,
3823 ["Acy"] = 1040,
3824 ["Bcy"] = 1041,
3825 ["Vcy"] = 1042,
3826 ["Gcy"] = 1043,
3827 ["Dcy"] = 1044,
3828 ["IEcy"] = 1045,
3829 ["ZHcy"] = 1046,
3830 ["Zcy"] = 1047,
3831 ["Icy"] = 1048,
3832 ["Jcy"] = 1049,
3833 ["Kcy"] = 1050,
3834 ["Lcy"] = 1051,
3835 ["Mcy"] = 1052,
3836 ["Ncy"] = 1053,
3837 ["Ocy"] = 1054,
3838 ["Pcy"] = 1055,
3839 ["Rcy"] = 1056,
3840 ["Scy"] = 1057,
3841 ["Tcy"] = 1058,
3842 ["Ucy"] = 1059,
3843 ["Fcy"] = 1060,
3844 ["KHcy"] = 1061,
3845 ["TScy"] = 1062,
3846 ["CHcy"] = 1063,
3847 ["SHcy"] = 1064,
3848 ["SHCHcy"] = 1065,
3849 ["HARDcy"] = 1066,
3850 ["Ycy"] = 1067,
3851 ["SOFTcy"] = 1068,
3852 ["Ecy"] = 1069,

3853 ["YUcy"] = 1070,
3854 ["YAcy"] = 1071,
3855 ["acy"] = 1072,
3856 ["bcy"] = 1073,
3857 ["vcy"] = 1074,
3858 ["gcy"] = 1075,
3859 ["dcy"] = 1076,
3860 ["iecy"] = 1077,
3861 ["zhcy"] = 1078,
3862 ["zcy"] = 1079,
3863 ["icy"] = 1080,
3864 ["jcy"] = 1081,
3865 ["kcy"] = 1082,
3866 ["lcy"] = 1083,
3867 ["mcy"] = 1084,
3868 ["ncy"] = 1085,
3869 ["ocy"] = 1086,
3870 ["pcy"] = 1087,
3871 ["rcy"] = 1088,
3872 ["scy"] = 1089,
3873 ["tcy"] = 1090,
3874 ["ucy"] = 1091,
3875 ["fcy"] = 1092,
3876 ["khcy"] = 1093,
3877 ["tscy"] = 1094,
3878 ["chcy"] = 1095,
3879 ["shcy"] = 1096,
3880 ["shchcy"] = 1097,
3881 ["hardcy"] = 1098,
3882 ["ycy"] = 1099,
3883 ["softcy"] = 1100,
3884 ["ecy"] = 1101,
3885 ["yucy"] = 1102,
3886 ["yacy"] = 1103,
3887 ["iocy"] = 1105,
3888 ["djcy"] = 1106,
3889 ["gjcy"] = 1107,
3890 ["jukcy"] = 1108,
3891 ["dscy"] = 1109,
3892 ["iukcy"] = 1110,
3893 ["yicy"] = 1111,
3894 ["jsercy"] = 1112,
3895 ["ljcy"] = 1113,
3896 ["njcy"] = 1114,
3897 ["tshcy"] = 1115,
3898 ["kjcy"] = 1116,
3899 ["ubrscy"] = 1118,

3900 ["dzcycy"] = 1119,
3901 ["ensp"] = 8194,
3902 ["emsp"] = 8195,
3903 ["emsp13"] = 8196,
3904 ["emsp14"] = 8197,
3905 ["numsp"] = 8199,
3906 ["puncsp"] = 8200,
3907 ["ThinSpace"] = 8201,
3908 ["thinsp"] = 8201,
3909 ["VeryThinSpace"] = 8202,
3910 ["hairsp"] = 8202,
3911 ["NegativeMediumSpace"] = 8203,
3912 ["NegativeThickSpace"] = 8203,
3913 ["NegativeThinSpace"] = 8203,
3914 ["NegativeVeryThinSpace"] = 8203,
3915 ["ZeroWidthSpace"] = 8203,
3916 ["zwnj"] = 8204,
3917 ["zwj"] = 8205,
3918 ["lrm"] = 8206,
3919 ["rlm"] = 8207,
3920 ["dash"] = 8208,
3921 ["hyphen"] = 8208,
3922 ["ndash"] = 8211,
3923 ["mdash"] = 8212,
3924 ["horbar"] = 8213,
3925 ["Verbar"] = 8214,
3926 ["Vert"] = 8214,
3927 ["OpenCurlyQuote"] = 8216,
3928 ["lsquo"] = 8216,
3929 ["CloseCurlyQuote"] = 8217,
3930 ["rsquo"] = 8217,
3931 ["rsquor"] = 8217,
3932 ["lsquor"] = 8218,
3933 ["sbquo"] = 8218,
3934 ["OpenCurlyDoubleQuote"] = 8220,
3935 ["ldquo"] = 8220,
3936 ["CloseCurlyDoubleQuote"] = 8221,
3937 ["rdquo"] = 8221,
3938 ["rdquor"] = 8221,
3939 ["bdquo"] = 8222,
3940 ["ldquor"] = 8222,
3941 ["dagger"] = 8224,
3942 ["Dagger"] = 8225,
3943 ["ddagger"] = 8225,
3944 ["bull"] = 8226,
3945 ["bullet"] = 8226,
3946 ["nldr"] = 8229,

3947 ["hellip"] = 8230,
 3948 ["mldr"] = 8230,
 3949 ["permil"] = 8240,
 3950 ["pertenk"] = 8241,
 3951 ["prime"] = 8242,
 3952 ["Prime"] = 8243,
 3953 ["tprime"] = 8244,
 3954 ["backprime"] = 8245,
 3955 ["bprime"] = 8245,
 3956 ["lsaquo"] = 8249,
 3957 ["rsaquo"] = 8250,
 3958 ["OverBar"] = 8254,
 3959 ["oline"] = 8254,
 3960 ["caret"] = 8257,
 3961 ["hybull"] = 8259,
 3962 ["frasl"] = 8260,
 3963 ["bsemi"] = 8271,
 3964 ["qprime"] = 8279,
 3965 ["MediumSpace"] = 8287,
 3966 ["ThickSpace"] = {8287, 8202},
 3967 ["NoBreak"] = 8288,
 3968 ["ApplyFunction"] = 8289,
 3969 ["af"] = 8289,
 3970 ["InvisibleTimes"] = 8290,
 3971 ["it"] = 8290,
 3972 ["InvisibleComma"] = 8291,
 3973 ["ic"] = 8291,
 3974 ["euro"] = 8364,
 3975 ["TripleDot"] = 8411,
 3976 ["tdot"] = 8411,
 3977 ["DotDot"] = 8412,
 3978 ["Copf"] = 8450,
 3979 ["complexes"] = 8450,
 3980 ["incare"] = 8453,
 3981 ["gscr"] = 8458,
 3982 ["HilbertSpace"] = 8459,
 3983 ["Hscr"] = 8459,
 3984 ["hamilt"] = 8459,
 3985 ["Hfr"] = 8460,
 3986 ["Poincareplane"] = 8460,
 3987 ["Hopf"] = 8461,
 3988 ["quaternions"] = 8461,
 3989 ["planckh"] = 8462,
 3990 ["hbar"] = 8463,
 3991 ["hslash"] = 8463,
 3992 ["planck"] = 8463,
 3993 ["plankv"] = 8463,

3994 ["Iscr"] = 8464,
3995 ["imagline"] = 8464,
3996 ["Ifr"] = 8465,
3997 ["Im"] = 8465,
3998 ["image"] = 8465,
3999 ["imagpart"] = 8465,
4000 ["Laplacetrif"] = 8466,
4001 ["Lscr"] = 8466,
4002 ["lagran"] = 8466,
4003 ["ell"] = 8467,
4004 ["Nopf"] = 8469,
4005 ["naturals"] = 8469,
4006 ["numero"] = 8470,
4007 ["copysr"] = 8471,
4008 ["weierp"] = 8472,
4009 ["wp"] = 8472,
4010 ["Popf"] = 8473,
4011 ["primes"] = 8473,
4012 ["Qopf"] = 8474,
4013 ["rationals"] = 8474,
4014 ["Rscr"] = 8475,
4015 ["realine"] = 8475,
4016 ["Re"] = 8476,
4017 ["Rfr"] = 8476,
4018 ["real"] = 8476,
4019 ["realpart"] = 8476,
4020 ["Ropf"] = 8477,
4021 ["reals"] = 8477,
4022 ["rx"] = 8478,
4023 ["TRADE"] = 8482,
4024 ["trade"] = 8482,
4025 ["Zopf"] = 8484,
4026 ["integers"] = 8484,
4027 ["mho"] = 8487,
4028 ["Zfr"] = 8488,
4029 ["zeetrif"] = 8488,
4030 ["iiota"] = 8489,
4031 ["Bernoullis"] = 8492,
4032 ["Bscr"] = 8492,
4033 ["bernou"] = 8492,
4034 ["Cayleys"] = 8493,
4035 ["Cfr"] = 8493,
4036 ["escr"] = 8495,
4037 ["Escr"] = 8496,
4038 ["expectation"] = 8496,
4039 ["Fouriertrif"] = 8497,
4040 ["Fscr"] = 8497,

4041 ["Mellintrf"] = 8499,
 4042 ["Mscr"] = 8499,
 4043 ["phmmat"] = 8499,
 4044 ["order"] = 8500,
 4045 ["orderof"] = 8500,
 4046 ["oscr"] = 8500,
 4047 ["alefsym"] = 8501,
 4048 ["aleph"] = 8501,
 4049 ["beth"] = 8502,
 4050 ["gimel"] = 8503,
 4051 ["daleth"] = 8504,
 4052 ["CapitalDifferentialD"] = 8517,
 4053 ["DD"] = 8517,
 4054 ["DifferentialD"] = 8518,
 4055 ["dd"] = 8518,
 4056 ["ExponentialE"] = 8519,
 4057 ["ee"] = 8519,
 4058 ["exponentiale"] = 8519,
 4059 ["ImaginaryI"] = 8520,
 4060 ["ii"] = 8520,
 4061 ["frac13"] = 8531,
 4062 ["frac23"] = 8532,
 4063 ["frac15"] = 8533,
 4064 ["frac25"] = 8534,
 4065 ["frac35"] = 8535,
 4066 ["frac45"] = 8536,
 4067 ["frac16"] = 8537,
 4068 ["frac56"] = 8538,
 4069 ["frac18"] = 8539,
 4070 ["frac38"] = 8540,
 4071 ["frac58"] = 8541,
 4072 ["frac78"] = 8542,
 4073 ["LeftArrow"] = 8592,
 4074 ["ShortLeftArrow"] = 8592,
 4075 ["larr"] = 8592,
 4076 ["leftarrow"] = 8592,
 4077 ["slarr"] = 8592,
 4078 ["ShortUpArrow"] = 8593,
 4079 ["UpArrow"] = 8593,
 4080 ["uarr"] = 8593,
 4081 ["uparrow"] = 8593,
 4082 ["RightArrow"] = 8594,
 4083 ["ShortRightArrow"] = 8594,
 4084 ["rarr"] = 8594,
 4085 ["rightarrow"] = 8594,
 4086 ["srarr"] = 8594,
 4087 ["DownArrow"] = 8595,

4088 ["ShortDownArrow"] = 8595,
 4089 ["darr"] = 8595,
 4090 ["downarrow"] = 8595,
 4091 ["LeftRightArrow"] = 8596,
 4092 ["harr"] = 8596,
 4093 ["leftrightarrow"] = 8596,
 4094 ["UpDownArrow"] = 8597,
 4095 ["updownarrow"] = 8597,
 4096 ["varr"] = 8597,
 4097 ["UpperLeftArrow"] = 8598,
 4098 ["nwarr"] = 8598,
 4099 ["nwarrow"] = 8598,
 4100 ["UpperRightArrow"] = 8599,
 4101 ["nearr"] = 8599,
 4102 ["nearrow"] = 8599,
 4103 ["LowerRightArrow"] = 8600,
 4104 ["searr"] = 8600,
 4105 ["searrow"] = 8600,
 4106 ["LowerLeftArrow"] = 8601,
 4107 ["swarr"] = 8601,
 4108 ["swarrow"] = 8601,
 4109 ["nlarr"] = 8602,
 4110 ["nleftarrow"] = 8602,
 4111 ["nrarr"] = 8603,
 4112 ["nrightarrow"] = 8603,
 4113 ["nrarrw"] = {8605, 824},
 4114 ["rarrw"] = 8605,
 4115 ["rightsquigarrow"] = 8605,
 4116 ["Larr"] = 8606,
 4117 ["twoheadleftarrow"] = 8606,
 4118 ["Uarr"] = 8607,
 4119 ["Rarr"] = 8608,
 4120 ["twoheadrightarrow"] = 8608,
 4121 ["Darr"] = 8609,
 4122 ["larrtl"] = 8610,
 4123 ["leftarrowtail"] = 8610,
 4124 ["rarrtl"] = 8611,
 4125 ["rightarrowtail"] = 8611,
 4126 ["LeftTeeArrow"] = 8612,
 4127 ["mapstoleft"] = 8612,
 4128 ["UpTeeArrow"] = 8613,
 4129 ["mapstoup"] = 8613,
 4130 ["RightTeeArrow"] = 8614,
 4131 ["map"] = 8614,
 4132 ["mapsto"] = 8614,
 4133 ["DownTeeArrow"] = 8615,
 4134 ["mapstodown"] = 8615,

4135 ["hookleftarrow"] = 8617,
 4136 ["larrhk"] = 8617,
 4137 ["hookrightarrow"] = 8618,
 4138 ["rarrhk"] = 8618,
 4139 ["larrlp"] = 8619,
 4140 ["looparrowleft"] = 8619,
 4141 ["looparrowright"] = 8620,
 4142 ["rarrlp"] = 8620,
 4143 ["harrw"] = 8621,
 4144 ["leftrightsquigarrow"] = 8621,
 4145 ["nharr"] = 8622,
 4146 ["nletrightarrow"] = 8622,
 4147 ["Lsh"] = 8624,
 4148 ["lsh"] = 8624,
 4149 ["Rsh"] = 8625,
 4150 ["rsh"] = 8625,
 4151 ["ldsh"] = 8626,
 4152 ["rdsh"] = 8627,
 4153 ["crarr"] = 8629,
 4154 ["cularr"] = 8630,
 4155 ["curvearrowleft"] = 8630,
 4156 ["curarr"] = 8631,
 4157 ["curvearrowright"] = 8631,
 4158 ["circlearrowleft"] = 8634,
 4159 ["olarr"] = 8634,
 4160 ["circlearrowright"] = 8635,
 4161 ["orarr"] = 8635,
 4162 ["LeftVector"] = 8636,
 4163 ["leftharpoonup"] = 8636,
 4164 ["lharu"] = 8636,
 4165 ["DownLeftVector"] = 8637,
 4166 ["leftharpoondown"] = 8637,
 4167 ["lhard"] = 8637,
 4168 ["RightUpVector"] = 8638,
 4169 ["uharr"] = 8638,
 4170 ["upharpoonright"] = 8638,
 4171 ["LeftUpVector"] = 8639,
 4172 ["uharl"] = 8639,
 4173 ["upharpoonleft"] = 8639,
 4174 ["RightVector"] = 8640,
 4175 ["rharu"] = 8640,
 4176 ["rightharpoonup"] = 8640,
 4177 ["DownRightVector"] = 8641,
 4178 ["rhard"] = 8641,
 4179 ["rightharpoondown"] = 8641,
 4180 ["RightDownVector"] = 8642,
 4181 ["dharr"] = 8642,

4182 ["downharpoonright"] = 8642,
 4183 ["LeftDownVector"] = 8643,
 4184 ["dharl"] = 8643,
 4185 ["downharpoonleft"] = 8643,
 4186 ["RightArrowLeftArrow"] = 8644,
 4187 ["rightleftarrows"] = 8644,
 4188 ["rlarr"] = 8644,
 4189 ["UpArrowDownArrow"] = 8645,
 4190 ["udarr"] = 8645,
 4191 ["LeftArrowRightArrow"] = 8646,
 4192 ["leftrightarrows"] = 8646,
 4193 ["lrarr"] = 8646,
 4194 ["leftleftarrows"] = 8647,
 4195 ["llarr"] = 8647,
 4196 ["upuparrows"] = 8648,
 4197 ["uuarr"] = 8648,
 4198 ["rightrightarrows"] = 8649,
 4199 ["rrarr"] = 8649,
 4200 ["ddarr"] = 8650,
 4201 ["downdownarrows"] = 8650,
 4202 ["ReverseEquilibrium"] = 8651,
 4203 ["leftrightharpoons"] = 8651,
 4204 ["lrhar"] = 8651,
 4205 ["Equilibrium"] = 8652,
 4206 ["rightleftharpoons"] = 8652,
 4207 ["rlhar"] = 8652,
 4208 ["nLeftarrow"] = 8653,
 4209 ["nLArr"] = 8653,
 4210 ["nLeftrightarrow"] = 8654,
 4211 ["nhArr"] = 8654,
 4212 ["nRightarrow"] = 8655,
 4213 ["nrArr"] = 8655,
 4214 ["DoubleLeftArrow"] = 8656,
 4215 ["Leftarrow"] = 8656,
 4216 ["lArr"] = 8656,
 4217 ["DoubleUpArrow"] = 8657,
 4218 ["Uparrow"] = 8657,
 4219 ["uArr"] = 8657,
 4220 ["DoubleRightArrow"] = 8658,
 4221 ["Implies"] = 8658,
 4222 ["Rightarrow"] = 8658,
 4223 ["rArr"] = 8658,
 4224 ["DoubleDownArrow"] = 8659,
 4225 ["Downarrow"] = 8659,
 4226 ["dArr"] = 8659,
 4227 ["DoubleLeftRightArrow"] = 8660,
 4228 ["Leftrightarrow"] = 8660,

4229 ["hArr"] = 8660,
 4230 ["iff"] = 8660,
 4231 ["DoubleUpDownArrow"] = 8661,
 4232 ["Updownarrow"] = 8661,
 4233 ["vArr"] = 8661,
 4234 ["nwArr"] = 8662,
 4235 ["neArr"] = 8663,
 4236 ["seArr"] = 8664,
 4237 ["swArr"] = 8665,
 4238 ["Lleftarrow"] = 8666,
 4239 ["lAarr"] = 8666,
 4240 ["Rrightarrow"] = 8667,
 4241 ["rAarr"] = 8667,
 4242 ["zigrarr"] = 8669,
 4243 ["LeftArrowBar"] = 8676,
 4244 ["larrb"] = 8676,
 4245 ["RightArrowBar"] = 8677,
 4246 ["rarrb"] = 8677,
 4247 ["DownArrowUpArrow"] = 8693,
 4248 ["duarr"] = 8693,
 4249 ["loarr"] = 8701,
 4250 ["roarr"] = 8702,
 4251 ["hoarr"] = 8703,
 4252 ["ForAll"] = 8704,
 4253 ["forall"] = 8704,
 4254 ["comp"] = 8705,
 4255 ["complement"] = 8705,
 4256 ["PartialD"] = 8706,
 4257 ["npart"] = {8706, 824},
 4258 ["part"] = 8706,
 4259 ["Exists"] = 8707,
 4260 ["exist"] = 8707,
 4261 ["NotExists"] = 8708,
 4262 ["nexist"] = 8708,
 4263 ["nexists"] = 8708,
 4264 ["empty"] = 8709,
 4265 ["emptyset"] = 8709,
 4266 ["emptyv"] = 8709,
 4267 ["varnothing"] = 8709,
 4268 ["Del"] = 8711,
 4269 ["nabla"] = 8711,
 4270 ["Element"] = 8712,
 4271 ["in"] = 8712,
 4272 ["isin"] = 8712,
 4273 ["isinv"] = 8712,
 4274 ["NotElement"] = 8713,
 4275 ["notin"] = 8713,

4276 ["notinva"] = 8713,
 4277 ["ReverseElement"] = 8715,
 4278 ["SuchThat"] = 8715,
 4279 ["ni"] = 8715,
 4280 ["niv"] = 8715,
 4281 ["NotReverseElement"] = 8716,
 4282 ["notni"] = 8716,
 4283 ["notniva"] = 8716,
 4284 ["Product"] = 8719,
 4285 ["prod"] = 8719,
 4286 ["Coproduct"] = 8720,
 4287 ["coprod"] = 8720,
 4288 ["Sum"] = 8721,
 4289 ["sum"] = 8721,
 4290 ["minus"] = 8722,
 4291 ["MinusPlus"] = 8723,
 4292 ["mnplus"] = 8723,
 4293 ["mp"] = 8723,
 4294 ["dotplus"] = 8724,
 4295 ["plusdo"] = 8724,
 4296 ["Backslash"] = 8726,
 4297 ["setminus"] = 8726,
 4298 ["setmn"] = 8726,
 4299 ["smallsetminus"] = 8726,
 4300 ["ssetmn"] = 8726,
 4301 ["lowast"] = 8727,
 4302 ["SmallCircle"] = 8728,
 4303 ["compfn"] = 8728,
 4304 ["Sqrt"] = 8730,
 4305 ["radic"] = 8730,
 4306 ["Proportional"] = 8733,
 4307 ["prop"] = 8733,
 4308 ["propto"] = 8733,
 4309 ["varpropto"] = 8733,
 4310 ["vprop"] = 8733,
 4311 ["infin"] = 8734,
 4312 ["angrt"] = 8735,
 4313 ["ang"] = 8736,
 4314 ["angle"] = 8736,
 4315 ["nang"] = {8736, 8402},
 4316 ["angmsd"] = 8737,
 4317 ["measuredangle"] = 8737,
 4318 ["angsph"] = 8738,
 4319 ["VerticalBar"] = 8739,
 4320 ["mid"] = 8739,
 4321 ["shortmid"] = 8739,
 4322 ["smid"] = 8739,

4323 ["NotVerticalBar"] = 8740,
 4324 ["nmid"] = 8740,
 4325 ["nshortmid"] = 8740,
 4326 ["nsmid"] = 8740,
 4327 ["DoubleVerticalBar"] = 8741,
 4328 ["par"] = 8741,
 4329 ["parallel"] = 8741,
 4330 ["shortparallel"] = 8741,
 4331 ["spar"] = 8741,
 4332 ["NotDoubleVerticalBar"] = 8742,
 4333 ["npar"] = 8742,
 4334 ["nparallel"] = 8742,
 4335 ["nshortparallel"] = 8742,
 4336 ["nspar"] = 8742,
 4337 ["and"] = 8743,
 4338 ["wedge"] = 8743,
 4339 ["or"] = 8744,
 4340 ["vee"] = 8744,
 4341 ["cap"] = 8745,
 4342 ["caps"] = {8745, 65024},
 4343 ["cup"] = 8746,
 4344 ["cups"] = {8746, 65024},
 4345 ["Integral"] = 8747,
 4346 ["int"] = 8747,
 4347 ["Int"] = 8748,
 4348 ["iiint"] = 8749,
 4349 ["tint"] = 8749,
 4350 ["ContourIntegral"] = 8750,
 4351 ["conint"] = 8750,
 4352 ["oint"] = 8750,
 4353 ["Conint"] = 8751,
 4354 ["DoubleContourIntegral"] = 8751,
 4355 ["Cconint"] = 8752,
 4356 ["cwint"] = 8753,
 4357 ["ClockwiseContourIntegral"] = 8754,
 4358 ["cwconint"] = 8754,
 4359 ["CounterClockwiseContourIntegral"] = 8755,
 4360 ["awconint"] = 8755,
 4361 ["Therefore"] = 8756,
 4362 ["there4"] = 8756,
 4363 ["therefore"] = 8756,
 4364 ["Because"] = 8757,
 4365 ["because"] = 8757,
 4366 ["because"] = 8757,
 4367 ["ratio"] = 8758,
 4368 ["Colon"] = 8759,
 4369 ["Proportion"] = 8759,

4370 ["dotminus"] = 8760,
 4371 ["minusd"] = 8760,
 4372 ["mDDot"] = 8762,
 4373 ["homtht"] = 8763,
 4374 ["Tilde"] = 8764,
 4375 ["nvsim"] = {8764, 8402},
 4376 ["sim"] = 8764,
 4377 ["thicksim"] = 8764,
 4378 ["thksim"] = 8764,
 4379 ["backsim"] = 8765,
 4380 ["bsim"] = 8765,
 4381 ["race"] = {8765, 817},
 4382 ["ac"] = 8766,
 4383 ["acE"] = {8766, 819},
 4384 ["mstpos"] = 8766,
 4385 ["acd"] = 8767,
 4386 ["VerticalTilde"] = 8768,
 4387 ["wr"] = 8768,
 4388 ["wreath"] = 8768,
 4389 ["NotTilde"] = 8769,
 4390 ["nsim"] = 8769,
 4391 ["EqualTilde"] = 8770,
 4392 ["NotEqualTilde"] = {8770, 824},
 4393 ["eqsim"] = 8770,
 4394 ["esim"] = 8770,
 4395 ["nesim"] = {8770, 824},
 4396 ["TildeEqual"] = 8771,
 4397 ["sime"] = 8771,
 4398 ["simeq"] = 8771,
 4399 ["NotTildeEqual"] = 8772,
 4400 ["nsime"] = 8772,
 4401 ["nsimeq"] = 8772,
 4402 ["TildeFullEqual"] = 8773,
 4403 ["cong"] = 8773,
 4404 ["simne"] = 8774,
 4405 ["NotTildeFullEqual"] = 8775,
 4406 ["ncong"] = 8775,
 4407 ["TildeTilde"] = 8776,
 4408 ["ap"] = 8776,
 4409 ["approx"] = 8776,
 4410 ["asyp"] = 8776,
 4411 ["thickapprox"] = 8776,
 4412 ["thkap"] = 8776,
 4413 ["NotTildeTilde"] = 8777,
 4414 ["nap"] = 8777,
 4415 ["napprox"] = 8777,
 4416 ["ape"] = 8778,

4417 ["approxeq"] = 8778,
4418 ["apid"] = 8779,
4419 ["napid"] = {8779, 824},
4420 ["backcong"] = 8780,
4421 ["bcong"] = 8780,
4422 ["CupCap"] = 8781,
4423 ["asympeq"] = 8781,
4424 ["nvap"] = {8781, 8402},
4425 ["Bumpeq"] = 8782,
4426 ["HumpDownHump"] = 8782,
4427 ["NotHumpDownHump"] = {8782, 824},
4428 ["bump"] = 8782,
4429 ["nbump"] = {8782, 824},
4430 ["HumpEqual"] = 8783,
4431 ["NotHumpEqual"] = {8783, 824},
4432 ["bumpe"] = 8783,
4433 ["bumpeq"] = 8783,
4434 ["nbumpe"] = {8783, 824},
4435 ["DotEqual"] = 8784,
4436 ["doteq"] = 8784,
4437 ["esdot"] = 8784,
4438 ["nedot"] = {8784, 824},
4439 ["doteqdot"] = 8785,
4440 ["eDot"] = 8785,
4441 ["efDot"] = 8786,
4442 ["fallingdotseq"] = 8786,
4443 ["erDot"] = 8787,
4444 ["risingdotseq"] = 8787,
4445 ["Assign"] = 8788,
4446 ["colone"] = 8788,
4447 ["coloneq"] = 8788,
4448 ["ecolon"] = 8789,
4449 ["eqcolon"] = 8789,
4450 ["ecir"] = 8790,
4451 ["eqcirc"] = 8790,
4452 ["circeq"] = 8791,
4453 ["cire"] = 8791,
4454 ["wedgeq"] = 8793,
4455 ["veeeq"] = 8794,
4456 ["triangleq"] = 8796,
4457 ["trie"] = 8796,
4458 ["equest"] = 8799,
4459 ["questeq"] = 8799,
4460 ["NotEqual"] = 8800,
4461 ["ne"] = 8800,
4462 ["Congruent"] = 8801,
4463 ["bnequiv"] = {8801, 8421},

4464 ["equiv"] = 8801,
4465 ["NotCongruent"] = 8802,
4466 ["nequiv"] = 8802,
4467 ["le"] = 8804,
4468 ["leq"] = 8804,
4469 ["nvle"] = {8804, 8402},
4470 ["GreaterEqual"] = 8805,
4471 ["ge"] = 8805,
4472 ["geq"] = 8805,
4473 ["nvge"] = {8805, 8402},
4474 ["LessFullEqual"] = 8806,
4475 ["lE"] = 8806,
4476 ["leqq"] = 8806,
4477 ["nlE"] = {8806, 824},
4478 ["nleqq"] = {8806, 824},
4479 ["GreaterFullEqual"] = 8807,
4480 ["NotGreaterFullEqual"] = {8807, 824},
4481 ["gE"] = 8807,
4482 ["geqq"] = 8807,
4483 ["ngE"] = {8807, 824},
4484 ["ngeqq"] = {8807, 824},
4485 ["lnE"] = 8808,
4486 ["lneqq"] = 8808,
4487 ["lvertneqq"] = {8808, 65024},
4488 ["lvnE"] = {8808, 65024},
4489 ["gnE"] = 8809,
4490 ["gneqq"] = 8809,
4491 ["gvertneqq"] = {8809, 65024},
4492 ["gvnE"] = {8809, 65024},
4493 ["Lt"] = 8810,
4494 ["NestedLessLess"] = 8810,
4495 ["NotLessLess"] = {8810, 824},
4496 ["ll"] = 8810,
4497 ["nLt"] = {8810, 8402},
4498 ["nLtv"] = {8810, 824},
4499 ["Gt"] = 8811,
4500 ["NestedGreaterGreater"] = 8811,
4501 ["NotGreaterGreater"] = {8811, 824},
4502 ["gg"] = 8811,
4503 ["nGt"] = {8811, 8402},
4504 ["nGtv"] = {8811, 824},
4505 ["between"] = 8812,
4506 ["twixt"] = 8812,
4507 ["NotCupCap"] = 8813,
4508 ["NotLess"] = 8814,
4509 ["nless"] = 8814,
4510 ["nlt"] = 8814,

4511 ["NotGreater"] = 8815,
 4512 ["ngt"] = 8815,
 4513 ["ngtr"] = 8815,
 4514 ["NotLessEqual"] = 8816,
 4515 ["nle"] = 8816,
 4516 ["nleq"] = 8816,
 4517 ["NotGreaterEqual"] = 8817,
 4518 ["nge"] = 8817,
 4519 ["ngeq"] = 8817,
 4520 ["LessTilde"] = 8818,
 4521 ["lesssim"] = 8818,
 4522 ["lsim"] = 8818,
 4523 ["GreaterTilde"] = 8819,
 4524 ["gsim"] = 8819,
 4525 ["gtrsim"] = 8819,
 4526 ["NotLessTilde"] = 8820,
 4527 ["nlsim"] = 8820,
 4528 ["NotGreaterTilde"] = 8821,
 4529 ["ngsim"] = 8821,
 4530 ["LessGreater"] = 8822,
 4531 ["lessgtr"] = 8822,
 4532 ["lg"] = 8822,
 4533 ["GreaterLess"] = 8823,
 4534 ["gl"] = 8823,
 4535 ["gtrless"] = 8823,
 4536 ["NotLessGreater"] = 8824,
 4537 ["ntlg"] = 8824,
 4538 ["NotGreaterLess"] = 8825,
 4539 ["ntgl"] = 8825,
 4540 ["Precedes"] = 8826,
 4541 ["pr"] = 8826,
 4542 ["prec"] = 8826,
 4543 ["Succeeds"] = 8827,
 4544 ["sc"] = 8827,
 4545 ["succ"] = 8827,
 4546 ["PrecedesSlantEqual"] = 8828,
 4547 ["prcue"] = 8828,
 4548 ["preccurlyeq"] = 8828,
 4549 ["SucceedsSlantEqual"] = 8829,
 4550 ["sccue"] = 8829,
 4551 ["succcurlyeq"] = 8829,
 4552 ["PrecedesTilde"] = 8830,
 4553 ["precsim"] = 8830,
 4554 ["prsim"] = 8830,
 4555 ["NotSucceedsTilde"] = {8831, 824},
 4556 ["SucceedsTilde"] = 8831,
 4557 ["scsim"] = 8831,

```

4558 ["succsim"] = 8831,
4559 ["NotPrecedes"] = 8832,
4560 ["npr"] = 8832,
4561 ["nprec"] = 8832,
4562 ["NotSucceeds"] = 8833,
4563 ["nsc"] = 8833,
4564 ["nsucc"] = 8833,
4565 ["NotSubset"] = {8834, 8402},
4566 ["nsubset"] = {8834, 8402},
4567 ["sub"] = 8834,
4568 ["subset"] = 8834,
4569 ["vnsup"] = {8834, 8402},
4570 ["NotSuperset"] = {8835, 8402},
4571 ["Superset"] = 8835,
4572 ["nsupset"] = {8835, 8402},
4573 ["sup"] = 8835,
4574 ["supset"] = 8835,
4575 ["vnsup"] = {8835, 8402},
4576 ["nsub"] = 8836,
4577 ["nsup"] = 8837,
4578 ["SubsetEqual"] = 8838,
4579 ["sube"] = 8838,
4580 ["subseteq"] = 8838,
4581 ["SupersetEqual"] = 8839,
4582 ["supe"] = 8839,
4583 ["supseteq"] = 8839,
4584 ["NotSubsetEqual"] = 8840,
4585 ["nsube"] = 8840,
4586 ["nsubseteq"] = 8840,
4587 ["NotSupersetEqual"] = 8841,
4588 ["nsupe"] = 8841,
4589 ["nsupseteq"] = 8841,
4590 ["subne"] = 8842,
4591 ["subsetneq"] = 8842,
4592 ["varsubsetneq"] = {8842, 65024},
4593 ["vsubne"] = {8842, 65024},
4594 ["supne"] = 8843,
4595 ["supsetneq"] = 8843,
4596 ["varsupsetneq"] = {8843, 65024},
4597 ["vsupne"] = {8843, 65024},
4598 ["cupdot"] = 8845,
4599 ["UnionPlus"] = 8846,
4600 ["uplus"] = 8846,
4601 ["NotSquareSubset"] = {8847, 824},
4602 ["SquareSubset"] = 8847,
4603 ["sqsub"] = 8847,
4604 ["sqsubset"] = 8847,

```


4605 ["NotSquareSuperset"] = {8848, 824},
4606 ["SquareSuperset"] = 8848,
4607 ["sqsup"] = 8848,
4608 ["sqsupset"] = 8848,
4609 ["SquareSubsetEqual"] = 8849,
4610 ["sqsube"] = 8849,
4611 ["sqsubseteq"] = 8849,
4612 ["SquareSupersetEqual"] = 8850,
4613 ["sqsupe"] = 8850,
4614 ["sqsupseteq"] = 8850,
4615 ["SquareIntersection"] = 8851,
4616 ["sqcap"] = 8851,
4617 ["sqcaps"] = {8851, 65024},
4618 ["SquareUnion"] = 8852,
4619 ["sqcup"] = 8852,
4620 ["sqcups"] = {8852, 65024},
4621 ["CirclePlus"] = 8853,
4622 ["oplus"] = 8853,
4623 ["CircleMinus"] = 8854,
4624 ["ominus"] = 8854,
4625 ["CircleTimes"] = 8855,
4626 ["otimes"] = 8855,
4627 ["osol"] = 8856,
4628 ["CircleDot"] = 8857,
4629 ["odot"] = 8857,
4630 ["circledcirc"] = 8858,
4631 ["ocir"] = 8858,
4632 ["circledast"] = 8859,
4633 ["oast"] = 8859,
4634 ["circleddash"] = 8861,
4635 ["odash"] = 8861,
4636 ["boxplus"] = 8862,
4637 ["plusb"] = 8862,
4638 ["boxminus"] = 8863,
4639 ["minusb"] = 8863,
4640 ["boxtimes"] = 8864,
4641 ["timesb"] = 8864,
4642 ["dotsquare"] = 8865,
4643 ["sdotb"] = 8865,
4644 ["RightTee"] = 8866,
4645 ["vdash"] = 8866,
4646 ["LeftTee"] = 8867,
4647 ["dashv"] = 8867,
4648 ["DownTee"] = 8868,
4649 ["top"] = 8868,
4650 ["UpTee"] = 8869,
4651 ["bot"] = 8869,

4652 ["bottom"] = 8869,
 4653 ["perp"] = 8869,
 4654 ["models"] = 8871,
 4655 ["DoubleRightTee"] = 8872,
 4656 ["vDash"] = 8872,
 4657 ["Vdash"] = 8873,
 4658 ["Vvdash"] = 8874,
 4659 ["VDash"] = 8875,
 4660 ["nvdash"] = 8876,
 4661 ["nvDash"] = 8877,
 4662 ["nVdash"] = 8878,
 4663 ["nVDash"] = 8879,
 4664 ["prurel"] = 8880,
 4665 ["LeftTriangle"] = 8882,
 4666 ["vartriangleleft"] = 8882,
 4667 ["vltri"] = 8882,
 4668 ["RightTriangle"] = 8883,
 4669 ["vartriangleright"] = 8883,
 4670 ["vrtri"] = 8883,
 4671 ["LeftTriangleEqual"] = 8884,
 4672 ["ltrie"] = 8884,
 4673 ["nvltrie"] = {8884, 8402},
 4674 ["trianglelefteq"] = 8884,
 4675 ["RightTriangleEqual"] = 8885,
 4676 ["nvrtrie"] = {8885, 8402},
 4677 ["rtrie"] = 8885,
 4678 ["trianglerighteq"] = 8885,
 4679 ["origof"] = 8886,
 4680 ["imof"] = 8887,
 4681 ["multimap"] = 8888,
 4682 ["mumap"] = 8888,
 4683 ["hercon"] = 8889,
 4684 ["intcal"] = 8890,
 4685 ["intercal"] = 8890,
 4686 ["veebar"] = 8891,
 4687 ["barvee"] = 8893,
 4688 ["angrtvb"] = 8894,
 4689 ["ltri"] = 8895,
 4690 ["Wedge"] = 8896,
 4691 ["bigwedge"] = 8896,
 4692 ["xwedge"] = 8896,
 4693 ["Vee"] = 8897,
 4694 ["bigvee"] = 8897,
 4695 ["xvee"] = 8897,
 4696 ["Intersection"] = 8898,
 4697 ["bigcap"] = 8898,
 4698 ["xcap"] = 8898,

4699 ["Union"] = 8899,
 4700 ["bigcup"] = 8899,
 4701 ["xcup"] = 8899,
 4702 ["Diamond"] = 8900,
 4703 ["diam"] = 8900,
 4704 ["diamond"] = 8900,
 4705 ["sdot"] = 8901,
 4706 ["Star"] = 8902,
 4707 ["sstarf"] = 8902,
 4708 ["divideontimes"] = 8903,
 4709 ["divonx"] = 8903,
 4710 ["bowtie"] = 8904,
 4711 ["ltimes"] = 8905,
 4712 ["rtimes"] = 8906,
 4713 ["leftthreetimes"] = 8907,
 4714 ["lthree"] = 8907,
 4715 ["rightthreetimes"] = 8908,
 4716 ["rthree"] = 8908,
 4717 ["backsimeq"] = 8909,
 4718 ["bsime"] = 8909,
 4719 ["curlyvee"] = 8910,
 4720 ["cuvee"] = 8910,
 4721 ["curlywedge"] = 8911,
 4722 ["cuwed"] = 8911,
 4723 ["Sub"] = 8912,
 4724 ["Subset"] = 8912,
 4725 ["Sup"] = 8913,
 4726 ["Supset"] = 8913,
 4727 ["Cap"] = 8914,
 4728 ["Cup"] = 8915,
 4729 ["fork"] = 8916,
 4730 ["pitchfork"] = 8916,
 4731 ["epar"] = 8917,
 4732 ["lessdot"] = 8918,
 4733 ["ltdot"] = 8918,
 4734 ["gtdot"] = 8919,
 4735 ["gtrdot"] = 8919,
 4736 ["L1"] = 8920,
 4737 ["nL1"] = {8920, 824},
 4738 ["Gg"] = 8921,
 4739 ["ggg"] = 8921,
 4740 ["nGg"] = {8921, 824},
 4741 ["LessEqualGreater"] = 8922,
 4742 ["leg"] = 8922,
 4743 ["lesg"] = {8922, 65024},
 4744 ["lesseqgtr"] = 8922,
 4745 ["GreaterEqualLess"] = 8923,

4746 ["gel"] = 8923,
 4747 ["gesl"] = {8923, 65024},
 4748 ["gtreqless"] = 8923,
 4749 ["cuepr"] = 8926,
 4750 ["curlyeqprec"] = 8926,
 4751 ["cuesc"] = 8927,
 4752 ["curlyeqsucc"] = 8927,
 4753 ["NotPrecedesSlantEqual"] = 8928,
 4754 ["nprcue"] = 8928,
 4755 ["NotSucceedsSlantEqual"] = 8929,
 4756 ["nsccue"] = 8929,
 4757 ["NotSquareSubsetEqual"] = 8930,
 4758 ["nsqsube"] = 8930,
 4759 ["NotSquareSupersetEqual"] = 8931,
 4760 ["nsqsupe"] = 8931,
 4761 ["lnsim"] = 8934,
 4762 ["gnsim"] = 8935,
 4763 ["precnsim"] = 8936,
 4764 ["prnsim"] = 8936,
 4765 ["scnsim"] = 8937,
 4766 ["succnsim"] = 8937,
 4767 ["NotLeftTriangle"] = 8938,
 4768 ["nltri"] = 8938,
 4769 ["ntriangleleft"] = 8938,
 4770 ["NotRightTriangle"] = 8939,
 4771 ["nrtri"] = 8939,
 4772 ["ntriangleright"] = 8939,
 4773 ["NotLeftTriangleEqual"] = 8940,
 4774 ["nltrie"] = 8940,
 4775 ["ntrianglelefteq"] = 8940,
 4776 ["NotRightTriangleEqual"] = 8941,
 4777 ["nrtrie"] = 8941,
 4778 ["ntrianglerighteq"] = 8941,
 4779 ["vellip"] = 8942,
 4780 ["ctdot"] = 8943,
 4781 ["utdot"] = 8944,
 4782 ["dtdot"] = 8945,
 4783 ["disin"] = 8946,
 4784 ["isinsv"] = 8947,
 4785 ["isins"] = 8948,
 4786 ["isindot"] = 8949,
 4787 ["notindot"] = {8949, 824},
 4788 ["notinvc"] = 8950,
 4789 ["notinvb"] = 8951,
 4790 ["isinE"] = 8953,
 4791 ["notinE"] = {8953, 824},
 4792 ["nisd"] = 8954,

4793 ["xnis"] = 8955,
4794 ["nis"] = 8956,
4795 ["notnivc"] = 8957,
4796 ["notnivb"] = 8958,
4797 ["barwed"] = 8965,
4798 ["barwedge"] = 8965,
4799 ["Barwed"] = 8966,
4800 ["doublebarwedge"] = 8966,
4801 ["LeftCeiling"] = 8968,
4802 ["lceil"] = 8968,
4803 ["RightCeiling"] = 8969,
4804 ["rceil"] = 8969,
4805 ["LeftFloor"] = 8970,
4806 ["lfloor"] = 8970,
4807 ["RightFloor"] = 8971,
4808 ["rfloor"] = 8971,
4809 ["drcrop"] = 8972,
4810 ["dlcrop"] = 8973,
4811 ["urcrop"] = 8974,
4812 ["ulcrop"] = 8975,
4813 ["bnot"] = 8976,
4814 ["proflines"] = 8978,
4815 ["profsurf"] = 8979,
4816 ["telrec"] = 8981,
4817 ["target"] = 8982,
4818 ["ulcorn"] = 8988,
4819 ["ulcorner"] = 8988,
4820 ["urcorn"] = 8989,
4821 ["urcorner"] = 8989,
4822 ["dlcorn"] = 8990,
4823 ["llcorner"] = 8990,
4824 ["drcorn"] = 8991,
4825 ["lrcorn"] = 8991,
4826 ["frown"] = 8994,
4827 ["sfrown"] = 8994,
4828 ["smile"] = 8995,
4829 ["ssmile"] = 8995,
4830 ["cylcty"] = 9005,
4831 ["profalar"] = 9006,
4832 ["topbot"] = 9014,
4833 ["ovbar"] = 9021,
4834 ["solbar"] = 9023,
4835 ["angzarr"] = 9084,
4836 ["lmoust"] = 9136,
4837 ["lmoustache"] = 9136,
4838 ["rmoust"] = 9137,
4839 ["rmoustache"] = 9137,

4840 ["OverBracket"] = 9140,
4841 ["tbrk"] = 9140,
4842 ["UnderBracket"] = 9141,
4843 ["bbrk"] = 9141,
4844 ["bbrktbrk"] = 9142,
4845 ["OverParenthesis"] = 9180,
4846 ["UnderParenthesis"] = 9181,
4847 ["OverBrace"] = 9182,
4848 ["UnderBrace"] = 9183,
4849 ["trpezium"] = 9186,
4850 ["elinters"] = 9191,
4851 ["blank"] = 9251,
4852 ["circledS"] = 9416,
4853 ["oS"] = 9416,
4854 ["HorizontalLine"] = 9472,
4855 ["boxh"] = 9472,
4856 ["boxv"] = 9474,
4857 ["boxdr"] = 9484,
4858 ["boxdl"] = 9488,
4859 ["boxur"] = 9492,
4860 ["boxul"] = 9496,
4861 ["boxvr"] = 9500,
4862 ["boxvl"] = 9508,
4863 ["boxhd"] = 9516,
4864 ["boxhu"] = 9524,
4865 ["boxvh"] = 9532,
4866 ["boxH"] = 9552,
4867 ["boxV"] = 9553,
4868 ["boxdR"] = 9554,
4869 ["boxDr"] = 9555,
4870 ["boxDR"] = 9556,
4871 ["boxdL"] = 9557,
4872 ["boxDL"] = 9558,
4873 ["boxDL"] = 9559,
4874 ["boxuR"] = 9560,
4875 ["boxUr"] = 9561,
4876 ["boxUR"] = 9562,
4877 ["boxuL"] = 9563,
4878 ["boxUL"] = 9564,
4879 ["boxUL"] = 9565,
4880 ["boxvR"] = 9566,
4881 ["boxVr"] = 9567,
4882 ["boxVR"] = 9568,
4883 ["boxvL"] = 9569,
4884 ["boxVL"] = 9570,
4885 ["boxVL"] = 9571,
4886 ["boxHd"] = 9572,

4887 ["boxhD"] = 9573,
4888 ["boxHD"] = 9574,
4889 ["boxHu"] = 9575,
4890 ["boxhU"] = 9576,
4891 ["boxHU"] = 9577,
4892 ["boxvH"] = 9578,
4893 ["boxVh"] = 9579,
4894 ["boxVH"] = 9580,
4895 ["uhblk"] = 9600,
4896 ["lhblk"] = 9604,
4897 ["block"] = 9608,
4898 ["blk14"] = 9617,
4899 ["blk12"] = 9618,
4900 ["blk34"] = 9619,
4901 ["Square"] = 9633,
4902 ["squ"] = 9633,
4903 ["square"] = 9633,
4904 ["FilledVerySmallSquare"] = 9642,
4905 ["blacksquare"] = 9642,
4906 ["squarf"] = 9642,
4907 ["squf"] = 9642,
4908 ["EmptyVerySmallSquare"] = 9643,
4909 ["rect"] = 9645,
4910 ["marker"] = 9646,
4911 ["fltns"] = 9649,
4912 ["bigtriangleup"] = 9651,
4913 ["xutri"] = 9651,
4914 ["blacktriangle"] = 9652,
4915 ["utrif"] = 9652,
4916 ["triangle"] = 9653,
4917 ["utri"] = 9653,
4918 ["blacktriangleright"] = 9656,
4919 ["rtrif"] = 9656,
4920 ["rtri"] = 9657,
4921 ["triangleright"] = 9657,
4922 ["bigtriangledown"] = 9661,
4923 ["xdtri"] = 9661,
4924 ["blacktriangledown"] = 9662,
4925 ["dtrif"] = 9662,
4926 ["dtri"] = 9663,
4927 ["triangledown"] = 9663,
4928 ["blacktriangleleft"] = 9666,
4929 ["ltrif"] = 9666,
4930 ["ltri"] = 9667,
4931 ["triangleleft"] = 9667,
4932 ["loz"] = 9674,
4933 ["lozenge"] = 9674,

4934 ["cir"] = 9675,
4935 ["tridot"] = 9708,
4936 ["bigcirc"] = 9711,
4937 ["xcirc"] = 9711,
4938 ["ultri"] = 9720,
4939 ["urtri"] = 9721,
4940 ["lltri"] = 9722,
4941 ["EmptySmallSquare"] = 9723,
4942 ["FilledSmallSquare"] = 9724,
4943 ["bigstar"] = 9733,
4944 ["starf"] = 9733,
4945 ["star"] = 9734,
4946 ["phone"] = 9742,
4947 ["female"] = 9792,
4948 ["male"] = 9794,
4949 ["spades"] = 9824,
4950 ["spadesuit"] = 9824,
4951 ["clubs"] = 9827,
4952 ["clubsuit"] = 9827,
4953 ["hearts"] = 9829,
4954 ["heartsuit"] = 9829,
4955 ["diamondsuit"] = 9830,
4956 ["diams"] = 9830,
4957 ["sung"] = 9834,
4958 ["flat"] = 9837,
4959 ["natur"] = 9838,
4960 ["natural"] = 9838,
4961 ["sharp"] = 9839,
4962 ["check"] = 10003,
4963 ["checkmark"] = 10003,
4964 ["cross"] = 10007,
4965 ["malt"] = 10016,
4966 ["maltese"] = 10016,
4967 ["sext"] = 10038,
4968 ["VerticalSeparator"] = 10072,
4969 ["lbrk"] = 10098,
4970 ["rbrk"] = 10099,
4971 ["bsolhsub"] = 10184,
4972 ["suphsol"] = 10185,
4973 ["LeftDoubleBracket"] = 10214,
4974 ["lbrk"] = 10214,
4975 ["RightDoubleBracket"] = 10215,
4976 ["robrk"] = 10215,
4977 ["LeftAngleBracket"] = 10216,
4978 ["lang"] = 10216,
4979 ["langle"] = 10216,
4980 ["RightAngleBracket"] = 10217,

4981 ["rang"] = 10217,
 4982 ["rangle"] = 10217,
 4983 ["Lang"] = 10218,
 4984 ["Rang"] = 10219,
 4985 ["loang"] = 10220,
 4986 ["roang"] = 10221,
 4987 ["LongLeftArrow"] = 10229,
 4988 ["longleftarrow"] = 10229,
 4989 ["xlarr"] = 10229,
 4990 ["LongRightArrow"] = 10230,
 4991 ["longrightarrow"] = 10230,
 4992 ["xrarr"] = 10230,
 4993 ["LongLeftRightArrow"] = 10231,
 4994 ["longleftrightarrow"] = 10231,
 4995 ["xharr"] = 10231,
 4996 ["DoubleLongLeftArrow"] = 10232,
 4997 ["Longleftarrow"] = 10232,
 4998 ["xlArr"] = 10232,
 4999 ["DoubleLongRightArrow"] = 10233,
 5000 ["Longrightarrow"] = 10233,
 5001 ["xrArr"] = 10233,
 5002 ["DoubleLongLeftRightArrow"] = 10234,
 5003 ["Longleftrightarrow"] = 10234,
 5004 ["xhArr"] = 10234,
 5005 ["longmapsto"] = 10236,
 5006 ["xmap"] = 10236,
 5007 ["dzigrarr"] = 10239,
 5008 ["nvlArr"] = 10498,
 5009 ["nvrArr"] = 10499,
 5010 ["nvHarr"] = 10500,
 5011 ["Map"] = 10501,
 5012 ["lbarr"] = 10508,
 5013 ["bkarow"] = 10509,
 5014 ["rbarr"] = 10509,
 5015 ["lBarr"] = 10510,
 5016 ["dbkarow"] = 10511,
 5017 ["rBarr"] = 10511,
 5018 ["RBarr"] = 10512,
 5019 ["drbkarow"] = 10512,
 5020 ["DDottrahd"] = 10513,
 5021 ["UpArrowBar"] = 10514,
 5022 ["DownArrowBar"] = 10515,
 5023 ["Rarrtl"] = 10518,
 5024 ["latail"] = 10521,
 5025 ["ratail"] = 10522,
 5026 ["lAtail"] = 10523,
 5027 ["rAtail"] = 10524,

5028 ["larrfs"] = 10525,
5029 ["rarrfs"] = 10526,
5030 ["larrbfs"] = 10527,
5031 ["rarrbfs"] = 10528,
5032 ["nwarhk"] = 10531,
5033 ["nearhk"] = 10532,
5034 ["hksearow"] = 10533,
5035 ["searhk"] = 10533,
5036 ["hkswarow"] = 10534,
5037 ["swarhk"] = 10534,
5038 ["nwnear"] = 10535,
5039 ["nesear"] = 10536,
5040 ["toea"] = 10536,
5041 ["seswar"] = 10537,
5042 ["tosa"] = 10537,
5043 ["swnwar"] = 10538,
5044 ["nrarrc"] = {10547, 824},
5045 ["rarrc"] = 10547,
5046 ["cudarr"] = 10549,
5047 ["ldca"] = 10550,
5048 ["rdca"] = 10551,
5049 ["cudarrl"] = 10552,
5050 ["larrpl"] = 10553,
5051 ["curarrm"] = 10556,
5052 ["cularrp"] = 10557,
5053 ["rarrpl"] = 10565,
5054 ["harrcir"] = 10568,
5055 ["Uarrocir"] = 10569,
5056 ["lurdshar"] = 10570,
5057 ["ldrushar"] = 10571,
5058 ["LeftRightVector"] = 10574,
5059 ["RightUpDownVector"] = 10575,
5060 ["DownLeftRightVector"] = 10576,
5061 ["LeftUpDownVector"] = 10577,
5062 ["LeftVectorBar"] = 10578,
5063 ["RightVectorBar"] = 10579,
5064 ["RightUpVectorBar"] = 10580,
5065 ["RightDownVectorBar"] = 10581,
5066 ["DownLeftVectorBar"] = 10582,
5067 ["DownRightVectorBar"] = 10583,
5068 ["LeftUpVectorBar"] = 10584,
5069 ["LeftDownVectorBar"] = 10585,
5070 ["LeftTeeVector"] = 10586,
5071 ["RightTeeVector"] = 10587,
5072 ["RightUpTeeVector"] = 10588,
5073 ["RightDownTeeVector"] = 10589,
5074 ["DownLeftTeeVector"] = 10590,

5075 ["DownRightTeeVector"] = 10591,
5076 ["LeftUpTeeVector"] = 10592,
5077 ["LeftDownTeeVector"] = 10593,
5078 ["lHar"] = 10594,
5079 ["uHar"] = 10595,
5080 ["rHar"] = 10596,
5081 ["dHar"] = 10597,
5082 ["luruhar"] = 10598,
5083 ["ldrdhar"] = 10599,
5084 ["ruluhar"] = 10600,
5085 ["rdldhar"] = 10601,
5086 ["lharul"] = 10602,
5087 ["llhard"] = 10603,
5088 ["rharul"] = 10604,
5089 ["lrhard"] = 10605,
5090 ["UpEquilibrium"] = 10606,
5091 ["udhar"] = 10606,
5092 ["ReverseUpEquilibrium"] = 10607,
5093 ["duhar"] = 10607,
5094 ["RoundImplies"] = 10608,
5095 ["erarr"] = 10609,
5096 ["simrarr"] = 10610,
5097 ["larrsim"] = 10611,
5098 ["rarrsim"] = 10612,
5099 ["rarrap"] = 10613,
5100 ["ltlarr"] = 10614,
5101 ["gtrarr"] = 10616,
5102 ["subrarr"] = 10617,
5103 ["suplarr"] = 10619,
5104 ["lfisht"] = 10620,
5105 ["rfisht"] = 10621,
5106 ["ufisht"] = 10622,
5107 ["dfisht"] = 10623,
5108 ["lopar"] = 10629,
5109 ["ropar"] = 10630,
5110 ["lbrke"] = 10635,
5111 ["rbrke"] = 10636,
5112 ["lbrkslu"] = 10637,
5113 ["rbrksld"] = 10638,
5114 ["lbrksld"] = 10639,
5115 ["rbrkslu"] = 10640,
5116 ["langd"] = 10641,
5117 ["rangd"] = 10642,
5118 ["lparlt"] = 10643,
5119 ["rpargt"] = 10644,
5120 ["gtlPar"] = 10645,
5121 ["ltrPar"] = 10646,

5122 ["vzigzag"] = 10650,
5123 ["vangrt"] = 10652,
5124 ["angrtvbd"] = 10653,
5125 ["ange"] = 10660,
5126 ["range"] = 10661,
5127 ["dwangle"] = 10662,
5128 ["uwangle"] = 10663,
5129 ["angmsdaa"] = 10664,
5130 ["angmsdab"] = 10665,
5131 ["angmsdac"] = 10666,
5132 ["angmsdad"] = 10667,
5133 ["angmsdae"] = 10668,
5134 ["angmsdaf"] = 10669,
5135 ["angmsdag"] = 10670,
5136 ["angmsdah"] = 10671,
5137 ["bemptyv"] = 10672,
5138 ["demptyv"] = 10673,
5139 ["cemptyv"] = 10674,
5140 ["raemptyv"] = 10675,
5141 ["laemptyv"] = 10676,
5142 ["ohbar"] = 10677,
5143 ["omid"] = 10678,
5144 ["opar"] = 10679,
5145 ["operp"] = 10681,
5146 ["olcross"] = 10683,
5147 ["odsold"] = 10684,
5148 ["olcir"] = 10686,
5149 ["ofcir"] = 10687,
5150 ["olt"] = 10688,
5151 ["ogt"] = 10689,
5152 ["cirscir"] = 10690,
5153 ["cirE"] = 10691,
5154 ["solb"] = 10692,
5155 ["bsolb"] = 10693,
5156 ["boxbox"] = 10697,
5157 ["trisb"] = 10701,
5158 ["rtriltri"] = 10702,
5159 ["LeftTriangleBar"] = 10703,
5160 ["NotLeftTriangleBar"] = {10703, 824},
5161 ["NotRightTriangleBar"] = {10704, 824},
5162 ["RightTriangleBar"] = 10704,
5163 ["iinfin"] = 10716,
5164 ["infintie"] = 10717,
5165 ["nvinfin"] = 10718,
5166 ["eparsl"] = 10723,
5167 ["smeparsl"] = 10724,
5168 ["eqvparsl"] = 10725,

5169 ["blacklozenge"] = 10731,
5170 ["lozf"] = 10731,
5171 ["RuleDelayed"] = 10740,
5172 ["dsol"] = 10742,
5173 ["bigodot"] = 10752,
5174 ["xodot"] = 10752,
5175 ["bigoplus"] = 10753,
5176 ["xoplus"] = 10753,
5177 ["bigotimes"] = 10754,
5178 ["xotime"] = 10754,
5179 ["biguplus"] = 10756,
5180 ["xuplus"] = 10756,
5181 ["bigsqcup"] = 10758,
5182 ["xsqlcup"] = 10758,
5183 ["iiiint"] = 10764,
5184 ["qint"] = 10764,
5185 ["fpartint"] = 10765,
5186 ["cirfnint"] = 10768,
5187 ["awint"] = 10769,
5188 ["rppolint"] = 10770,
5189 ["scpolint"] = 10771,
5190 ["npolint"] = 10772,
5191 ["pointint"] = 10773,
5192 ["quatint"] = 10774,
5193 ["intlarhk"] = 10775,
5194 ["pluscir"] = 10786,
5195 ["plusacir"] = 10787,
5196 ["simplus"] = 10788,
5197 ["plusdu"] = 10789,
5198 ["plussim"] = 10790,
5199 ["plustwo"] = 10791,
5200 ["mcomma"] = 10793,
5201 ["minusdu"] = 10794,
5202 ["loplus"] = 10797,
5203 ["roplus"] = 10798,
5204 ["Cross"] = 10799,
5205 ["timesd"] = 10800,
5206 ["timesbar"] = 10801,
5207 ["smashp"] = 10803,
5208 ["lotimes"] = 10804,
5209 ["rotimes"] = 10805,
5210 ["otimesas"] = 10806,
5211 ["Otimes"] = 10807,
5212 ["odiv"] = 10808,
5213 ["triplus"] = 10809,
5214 ["triminus"] = 10810,
5215 ["tritime"] = 10811,

5216 ["intprod"] = 10812,
5217 ["iproduct"] = 10812,
5218 ["amalg"] = 10815,
5219 ["capdot"] = 10816,
5220 ["ncup"] = 10818,
5221 ["ncap"] = 10819,
5222 ["capand"] = 10820,
5223 ["cupor"] = 10821,
5224 ["cupcap"] = 10822,
5225 ["capcup"] = 10823,
5226 ["cupbrcap"] = 10824,
5227 ["capbrcup"] = 10825,
5228 ["cupcup"] = 10826,
5229 ["capcap"] = 10827,
5230 ["ccups"] = 10828,
5231 ["ccaps"] = 10829,
5232 ["ccupssm"] = 10832,
5233 ["And"] = 10835,
5234 ["Or"] = 10836,
5235 ["andand"] = 10837,
5236 ["oror"] = 10838,
5237 ["orslope"] = 10839,
5238 ["andslope"] = 10840,
5239 ["andv"] = 10842,
5240 ["orv"] = 10843,
5241 ["andd"] = 10844,
5242 ["ord"] = 10845,
5243 ["wedbar"] = 10847,
5244 ["sdote"] = 10854,
5245 ["simdot"] = 10858,
5246 ["congdote"] = 10861,
5247 ["ncongdote"] = {10861, 824},
5248 ["easter"] = 10862,
5249 ["apacir"] = 10863,
5250 ["apE"] = 10864,
5251 ["napE"] = {10864, 824},
5252 ["eplus"] = 10865,
5253 ["pluse"] = 10866,
5254 ["Esim"] = 10867,
5255 ["Colone"] = 10868,
5256 ["Equal"] = 10869,
5257 ["ddotseq"] = 10871,
5258 ["eDDot"] = 10871,
5259 ["equivDD"] = 10872,
5260 ["ltcir"] = 10873,
5261 ["gtcir"] = 10874,
5262 ["ltquest"] = 10875,

5263 ["gtquest"] = 10876,
5264 ["LessSlantEqual"] = 10877,
5265 ["NotLessSlantEqual"] = {10877, 824},
5266 ["leqslant"] = 10877,
5267 ["les"] = 10877,
5268 ["nleqslant"] = {10877, 824},
5269 ["nles"] = {10877, 824},
5270 ["GreaterSlantEqual"] = 10878,
5271 ["NotGreaterSlantEqual"] = {10878, 824},
5272 ["geqslant"] = 10878,
5273 ["ges"] = 10878,
5274 ["ngeqslant"] = {10878, 824},
5275 ["nges"] = {10878, 824},
5276 ["lesdot"] = 10879,
5277 ["gesdot"] = 10880,
5278 ["lesdoto"] = 10881,
5279 ["gesdoto"] = 10882,
5280 ["lesdotor"] = 10883,
5281 ["gesdoto1"] = 10884,
5282 ["lap"] = 10885,
5283 ["lessapprox"] = 10885,
5284 ["gap"] = 10886,
5285 ["gtrapprox"] = 10886,
5286 ["lne"] = 10887,
5287 ["lneq"] = 10887,
5288 ["gne"] = 10888,
5289 ["gneq"] = 10888,
5290 ["lnap"] = 10889,
5291 ["lnapprox"] = 10889,
5292 ["gnap"] = 10890,
5293 ["gnapprox"] = 10890,
5294 ["lEg"] = 10891,
5295 ["lesseqqgtr"] = 10891,
5296 ["gEl"] = 10892,
5297 ["gtreqqless"] = 10892,
5298 ["lsime"] = 10893,
5299 ["gsime"] = 10894,
5300 ["lsimg"] = 10895,
5301 ["gsiml"] = 10896,
5302 ["lgE"] = 10897,
5303 ["glE"] = 10898,
5304 ["lesges"] = 10899,
5305 ["gesles"] = 10900,
5306 ["els"] = 10901,
5307 ["eqslantless"] = 10901,
5308 ["egs"] = 10902,
5309 ["eqslantgtr"] = 10902,

```

5310 ["elsdot"] = 10903,
5311 ["egsdot"] = 10904,
5312 ["el"] = 10905,
5313 ["eg"] = 10906,
5314 ["siml"] = 10909,
5315 ["simg"] = 10910,
5316 ["simlE"] = 10911,
5317 ["simgE"] = 10912,
5318 ["LessLess"] = 10913,
5319 ["NotNestedLessLess"] = {10913, 824},
5320 ["GreaterGreater"] = 10914,
5321 ["NotNestedGreaterGreater"] = {10914, 824},
5322 ["glj"] = 10916,
5323 ["gla"] = 10917,
5324 ["ltcc"] = 10918,
5325 ["gtcc"] = 10919,
5326 ["lescc"] = 10920,
5327 ["gescc"] = 10921,
5328 ["smt"] = 10922,
5329 ["lat"] = 10923,
5330 ["smtE"] = 10924,
5331 ["smtes"] = {10924, 65024},
5332 ["late"] = 10925,
5333 ["lates"] = {10925, 65024},
5334 ["bumpE"] = 10926,
5335 ["NotPrecedesEqual"] = {10927, 824},
5336 ["PrecedesEqual"] = 10927,
5337 ["npre"] = {10927, 824},
5338 ["npreceq"] = {10927, 824},
5339 ["pre"] = 10927,
5340 ["preceq"] = 10927,
5341 ["NotSucceedsEqual"] = {10928, 824},
5342 ["SucceedsEqual"] = 10928,
5343 ["nsce"] = {10928, 824},
5344 ["nsucceq"] = {10928, 824},
5345 ["sce"] = 10928,
5346 ["succeq"] = 10928,
5347 ["prE"] = 10931,
5348 ["scE"] = 10932,
5349 ["precneqq"] = 10933,
5350 ["prnE"] = 10933,
5351 ["scnE"] = 10934,
5352 ["succneqq"] = 10934,
5353 ["prap"] = 10935,
5354 ["precapprox"] = 10935,
5355 ["scap"] = 10936,
5356 ["succapprox"] = 10936,

```


5357 ["precnapprox"] = 10937,
5358 ["prnap"] = 10937,
5359 ["scnap"] = 10938,
5360 ["succnapprox"] = 10938,
5361 ["Pr"] = 10939,
5362 ["Sc"] = 10940,
5363 ["subdot"] = 10941,
5364 ["supdot"] = 10942,
5365 ["subplus"] = 10943,
5366 ["supplus"] = 10944,
5367 ["submult"] = 10945,
5368 ["supmult"] = 10946,
5369 ["subedot"] = 10947,
5370 ["supedot"] = 10948,
5371 ["nsubE"] = {10949, 824},
5372 ["nsubseteqq"] = {10949, 824},
5373 ["subE"] = 10949,
5374 ["subseteqq"] = 10949,
5375 ["nsupE"] = {10950, 824},
5376 ["nsupseteqq"] = {10950, 824},
5377 ["supE"] = 10950,
5378 ["supseteqq"] = 10950,
5379 ["subsim"] = 10951,
5380 ["supsim"] = 10952,
5381 ["subnE"] = 10955,
5382 ["subsetneqq"] = 10955,
5383 ["varsubsetneqq"] = {10955, 65024},
5384 ["vsubnE"] = {10955, 65024},
5385 ["supnE"] = 10956,
5386 ["supsetneqq"] = 10956,
5387 ["varsupsetneqq"] = {10956, 65024},
5388 ["vsupnE"] = {10956, 65024},
5389 ["csub"] = 10959,
5390 ["csup"] = 10960,
5391 ["csube"] = 10961,
5392 ["csupe"] = 10962,
5393 ["subsup"] = 10963,
5394 ["supsub"] = 10964,
5395 ["subsub"] = 10965,
5396 ["supsup"] = 10966,
5397 ["suphsub"] = 10967,
5398 ["supdsub"] = 10968,
5399 ["forkv"] = 10969,
5400 ["topfork"] = 10970,
5401 ["mlcp"] = 10971,
5402 ["Dashv"] = 10980,
5403 ["DoubleLeftTee"] = 10980,

```

5404 ["Vdashl"] = 10982,
5405 ["Barv"] = 10983,
5406 ["vBar"] = 10984,
5407 ["vBarv"] = 10985,
5408 ["Vbar"] = 10987,
5409 ["Not"] = 10988,
5410 ["bNot"] = 10989,
5411 ["rnmid"] = 10990,
5412 ["cirmid"] = 10991,
5413 ["midcir"] = 10992,
5414 ["topcir"] = 10993,
5415 ["nhpar"] = 10994,
5416 ["parsim"] = 10995,
5417 ["nparsl"] = {11005, 8421},
5418 ["parsl"] = 11005,
5419 ["fflig"] = 64256,
5420 ["filig"] = 64257,
5421 ["fllig"] = 64258,
5422 ["ffilig"] = 64259,
5423 ["ffllig"] = 64260,
5424 ["Ascr"] = 119964,
5425 ["Cscr"] = 119966,
5426 ["Dscr"] = 119967,
5427 ["Gscr"] = 119970,
5428 ["Jscr"] = 119973,
5429 ["Kscr"] = 119974,
5430 ["Nscr"] = 119977,
5431 ["Oscr"] = 119978,
5432 ["Pscr"] = 119979,
5433 ["Qscr"] = 119980,
5434 ["Sscr"] = 119982,
5435 ["Tscr"] = 119983,
5436 ["Uscr"] = 119984,
5437 ["Vscr"] = 119985,
5438 ["Wscr"] = 119986,
5439 ["Xscr"] = 119987,
5440 ["Yscr"] = 119988,
5441 ["Zscr"] = 119989,
5442 ["ascr"] = 119990,
5443 ["bscr"] = 119991,
5444 ["cscr"] = 119992,
5445 ["dscr"] = 119993,
5446 ["fscr"] = 119995,
5447 ["hscr"] = 119997,
5448 ["iscr"] = 119998,
5449 ["jscr"] = 119999,
5450 ["kscr"] = 120000,

```

5451 ["lscr"] = 120001,
5452 ["mscr"] = 120002,
5453 ["nscr"] = 120003,
5454 ["pscr"] = 120005,
5455 ["qscr"] = 120006,
5456 ["rscr"] = 120007,
5457 ["sscr"] = 120008,
5458 ["tscr"] = 120009,
5459 ["uscr"] = 120010,
5460 ["vscr"] = 120011,
5461 ["wscr"] = 120012,
5462 ["xscr"] = 120013,
5463 ["yscr"] = 120014,
5464 ["zscr"] = 120015,
5465 ["Afr"] = 120068,
5466 ["Bfr"] = 120069,
5467 ["Dfr"] = 120071,
5468 ["Efr"] = 120072,
5469 ["Ffr"] = 120073,
5470 ["Gfr"] = 120074,
5471 ["Jfr"] = 120077,
5472 ["Kfr"] = 120078,
5473 ["Lfr"] = 120079,
5474 ["Mfr"] = 120080,
5475 ["Nfr"] = 120081,
5476 ["Ofr"] = 120082,
5477 ["Pfr"] = 120083,
5478 ["Qfr"] = 120084,
5479 ["Sfr"] = 120086,
5480 ["Tfr"] = 120087,
5481 ["Ufr"] = 120088,
5482 ["Vfr"] = 120089,
5483 ["Wfr"] = 120090,
5484 ["Xfr"] = 120091,
5485 ["Yfr"] = 120092,
5486 ["afr"] = 120094,
5487 ["bfr"] = 120095,
5488 ["cfr"] = 120096,
5489 ["dfr"] = 120097,
5490 ["efr"] = 120098,
5491 ["ffr"] = 120099,
5492 ["gfr"] = 120100,
5493 ["hfr"] = 120101,
5494 ["ifr"] = 120102,
5495 ["jfr"] = 120103,
5496 ["kfr"] = 120104,
5497 ["lfr"] = 120105,

5498 ["mfr"] = 120106,
5499 ["nfr"] = 120107,
5500 ["ofr"] = 120108,
5501 ["pfr"] = 120109,
5502 ["qfr"] = 120110,
5503 ["rfr"] = 120111,
5504 ["sfr"] = 120112,
5505 ["tfr"] = 120113,
5506 ["ufr"] = 120114,
5507 ["vfr"] = 120115,
5508 ["wfr"] = 120116,
5509 ["xfr"] = 120117,
5510 ["yfr"] = 120118,
5511 ["zfr"] = 120119,
5512 ["Aopf"] = 120120,
5513 ["Bopf"] = 120121,
5514 ["Dopf"] = 120123,
5515 ["Eopf"] = 120124,
5516 ["Fopf"] = 120125,
5517 ["Gopf"] = 120126,
5518 ["Iopf"] = 120128,
5519 ["Jopf"] = 120129,
5520 ["Kopf"] = 120130,
5521 ["Lopf"] = 120131,
5522 ["Mopf"] = 120132,
5523 ["Oopf"] = 120134,
5524 ["Sopf"] = 120138,
5525 ["Topf"] = 120139,
5526 ["Uopf"] = 120140,
5527 ["Vopf"] = 120141,
5528 ["Wopf"] = 120142,
5529 ["Xopf"] = 120143,
5530 ["Yopf"] = 120144,
5531 ["aopf"] = 120146,
5532 ["bopf"] = 120147,
5533 ["copf"] = 120148,
5534 ["dopf"] = 120149,
5535 ["eopf"] = 120150,
5536 ["fopf"] = 120151,
5537 ["gopf"] = 120152,
5538 ["hopf"] = 120153,
5539 ["iopf"] = 120154,
5540 ["jopf"] = 120155,
5541 ["kopf"] = 120156,
5542 ["lopf"] = 120157,
5543 ["mopf"] = 120158,
5544 ["nopf"] = 120159,

```

5545 ["oopf"] = 120160,
5546 ["popf"] = 120161,
5547 ["qopf"] = 120162,
5548 ["ropf"] = 120163,
5549 ["sopf"] = 120164,
5550 ["topf"] = 120165,
5551 ["uopf"] = 120166,
5552 ["vopf"] = 120167,
5553 ["wopf"] = 120168,
5554 ["xopf"] = 120169,
5555 ["yopf"] = 120170,
5556 ["zopf"] = 120171,
5557 }

```

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

```

5558 function entities.dec_entity(s)
5559   local n = tonumber(s)
5560   if n == nil then
5561     return "&#" .. s .. ";" -- fallback for unknown entities
5562   end
5563   return unicode.utf8.char(n)
5564 end

```

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

```

5565 function entities.hex_entity(s)
5566   local n = tonumber("0x"..s)
5567   if n == nil then
5568     return "&#x" .. s .. ";" -- fallback for unknown entities
5569   end
5570   return unicode.utf8.char(n)
5571 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.

```

5572 function entities.hex_entity_with_x_char(x, s)
5573   local n = tonumber("0x"..s)
5574   if n == nil then
5575     return "&#" .. x .. s .. ";" -- fallback for unknown entities
5576   end
5577   return unicode.utf8.char(n)
5578 end

```

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

```

5579 function entities.char_entity(s)

```

```

5580 local code_points = character_entities[s]
5581 if code_points == nil then
5582     return "&" .. s .. ";"
5583 end
5584 if type(code_points) ~= 'table' then
5585     code_points = {code_points}
5586 end
5587 local char_table = {}
5588 for _, code_point in ipairs(code_points) do
5589     table.insert(char_table, unicode.utf8.char(code_point))
5590 end
5591 return table.concat(char_table)
5592 end

```

3.1.3 Plain T_EX Writer

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

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

```
5593 M.writer = {}
```

The `writer.new` method creates and returns a new T_EX 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.

```
5594 function M.writer.new(options)
5595     local self = {}

```

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

```
5596     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.

```
5597     self.flatten_inlines = false

```

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

```
5598 local slice_specifiers = {}
5599 for specifier in options.slice:gmatch("[~%s]+") do
5600     table.insert(slice_specifiers, specifier)
5601 end
5602
5603 if #slice_specifiers == 2 then
5604     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5605     local slice_begin_type = self.slice_begin:sub(1, 1)
5606     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5607         self.slice_begin = "^" .. self.slice_begin
5608     end
5609     local slice_end_type = self.slice_end:sub(1, 1)
5610     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5611         self.slice_end = "$" .. self.slice_end
5612     end
5613 elseif #slice_specifiers == 1 then
5614     self.slice_begin = "^" .. slice_specifiers[1]
5615     self.slice_end = "$" .. slice_specifiers[1]
5616 end
5617
5618 self.slice_begin_type = self.slice_begin:sub(1, 1)
5619 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5620 self.slice_end_type = self.slice_end:sub(1, 1)
5621 self.slice_end_identifier = self.slice_end:sub(2) or ""
5622
5623 if self.slice_begin == "^" and self.slice_end ~= "^" then
5624     self.is_writing = true
5625 else
5626     self.is_writing = false
5627 end
```

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

```
5628 self.suffix = ".tex"
```

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

```
5629 self.space = " "
```

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

```
5630 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5631 function self.plain(s)
5632     return s
5633 end
```

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

```
5634 function self.paragraph(s)
5635   if not self.is_writing then return "" end
5636   return s
5637 end
```

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

```
5638 function self.pack(name)
5639   return [[\input{]} .. name .. [{}\relax]]
5640 end
```

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

```
5641 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
5642 function self.interblocksep()
5643   if not self.is_writing then return "" end
5644   return self.interblocksep_text
5645 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.

```
5646 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
5647 function self.paragraphsep()
5648   if not self.is_writing then return "" end
5649   return self.paragraphsep_text
5650 end
```

Define `writer->undosep` as a function that will remove the output produced by an immediately preceding block element / paragraph separator.

```
5651 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
5652 function self.undosep()
5653   if not self.is_writing then return "" end
5654   return self.undosep_text
5655 end
```

Define `writer->soft_line_break` as the output format of a soft line break.

```
5656 self.soft_line_break = function()
5657   if self.flatten_inlines then return "\n" end
5658   return "\\markdownRendererSoftLineBreak\n{"
5659 end
```

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

```
5660 self.hard_line_break = function()
5661   if self.flatten_inlines then return "\n" end
5662   return "\\markdownRendererHardLineBreak\n{"
5663 end
```


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

```
5664 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
5665 function self.thematic_break()
5666   if not self.is_writing then return "" end
5667   return "\\markdownRendererThematicBreak{"
5668 end
```

Define tables `writer->escaped_uri_chars` and `writer->escaped_minimal_strings` containing the mapping from special plain characters and character strings that always need to be escaped.

```
5669 self.escaped_uri_chars = {
5670   [{""] = "\\markdownRendererLeftBrace{"},
5671   ["}"] = "\\markdownRendererRightBrace{"},
5672   [{"\\"}] = "\\markdownRendererBackslash{"},
5673 }
5674 self.escaped_minimal_strings = {
5675   [{"^"}] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
5676   [{"☒"}] = "\\markdownRendererTickedBox{"},
5677   [{"◻"}] = "\\markdownRendererHalfTickedBox{"},
5678   [{"□"}] = "\\markdownRendererUntickedBox{"},
5679   [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{"},
5680 }
```

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

```
5681 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5682 self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

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

```
5683 self.escaped_chars = {
5684   [{"{"] = "\\markdownRendererLeftBrace{"},
5685   [{"}"] = "\\markdownRendererRightBrace{"},
5686   [{"%"}] = "\\markdownRendererPercentSign{"},
5687   [{"\\"}] = "\\markdownRendererBackslash{"},
5688   [{"#"}] = "\\markdownRendererHash{"},
5689   [{"$"}] = "\\markdownRendererDollarSign{"},
5690   [{"&"}] = "\\markdownRendererAmpersand{"},
5691   [{"_"}] = "\\markdownRendererUnderscore{"},
5692   [{"^"}] = "\\markdownRendererCircumflex{"},
5693   [{"~"}] = "\\markdownRendererTilde{"},
5694   [{"|"}] = "\\markdownRendererPipe{"},
5695   [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{"},
5696 }
```

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

```
5697 local function create_escaper(char_escapes, string_escapes)
5698     local escape = util.escaper(char_escapes, string_escapes)
5699     return function(s)
5700         if self.flatten_inlines then return s end
5701         return escape(s)
5702     end
5703 end
5704 local escape_typographic_text = create_escaper(
5705     self.escaped_chars, self.escaped_strings)
5706 local escape_programmatic_text = create_escaper(
5707     self.escaped_uri_chars, self.escaped_minimal_strings)
5708 local escape_minimal = create_escaper(
5709     {}, 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.

```
5710 self.escape = escape_typographic_text
5711 self.math = escape_minimal
5712 if options.hybrid then
5713     self.identifier = escape_minimal
5714     self.string = escape_minimal
5715     self.uri = escape_minimal
5716     self.infostring = escape_minimal
5717 else
5718     self.identifier = escape_programmatic_text
5719     self.string = escape_typographic_text
5720     self.uri = escape_programmatic_text
5721     self.infostring = escape_programmatic_text
5722 end
```

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

```
5723 function self.code(s, attributes)
5724     if self.flatten_inlines then return s end
```

```

5725     local buf = {}
5726     if attributes ~= nil then
5727         table.insert(buf,
5728             "\\markdownRendererCodeSpanAttributeContextBegin\n")
5729         table.insert(buf, self.attributes(attributes))
5730     end
5731     table.insert(buf,
5732         {"\\markdownRendererCodeSpan{" , self.escape(s), "}"})
5733     if attributes ~= nil then
5734         table.insert(buf,
5735             "\\markdownRendererCodeSpanAttributeContextEnd{")
5736     end
5737     return buf
5738 end

```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, `tit` to the title of the link, and `attributes` to optional attributes.

```

5739     function self.link(lab, src, tit, attributes)
5740         if self.flatten_inlines then return lab end
5741         local buf = {}
5742         if attributes ~= nil then
5743             table.insert(buf,
5744                 "\\markdownRendererLinkAttributeContextBegin\n")
5745             table.insert(buf, self.attributes(attributes))
5746         end
5747         table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ,
5748             {"",self.escape(src),"} ,
5749             {"",self.uri(src),"} ,
5750             {"",self.string(tit or ""),"}"}))
5751         if attributes ~= nil then
5752             table.insert(buf,
5753                 "\\markdownRendererLinkAttributeContextEnd{")
5754         end
5755         return buf
5756     end

```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, `tit` to the title of the image, and `attributes` to optional attributes.

```

5757     function self.image(lab, src, tit, attributes)
5758         if self.flatten_inlines then return lab end
5759         local buf = {}
5760         if attributes ~= nil then
5761             table.insert(buf,
5762                 "\\markdownRendererImageAttributeContextBegin\n")
5763             table.insert(buf, self.attributes(attributes))

```

```

5764     end
5765     table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
5766                     "{" ,self.string(src),"} ",
5767                     "{" ,self.uri(src),"} ",
5768                     "{" ,self.string(tit or ""),"}"} )
5769     if attributes ~= nil then
5770         table.insert(buf,
5771                     "\\markdownRendererImageAttributeContextEnd{ }")
5772     end
5773     return buf
5774 end

```

Define `writer->bulletlist` as a function that will transform an input bulleted list to the output format, where `items` is an array of the list items and `tight` specifies, whether the list is tight or not.

```

5775 function self.bulletlist(items,tight)
5776     if not self.is_writing then return "" end
5777     local buffer = {}
5778     for _,item in ipairs(items) do
5779         if item ~= "" then
5780             buffer[#buffer + 1] = self.bulletitem(item)
5781         end
5782     end
5783     local contents = util.intersperse(buffer,"\n")
5784     if tight and options.tightLists then
5785         return {"\\markdownRendererULBeginTight\n",contents,
5786                 "\n\\markdownRendererULEndTight "}
5787     else
5788         return {"\\markdownRendererULBegin\n",contents,
5789                 "\n\\markdownRendererULEnd "}
5790     end
5791 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.

```

5792 function self.bulletitem(s)
5793     return {"\\markdownRendererULItem ",s,
5794             "\\markdownRendererULItemEnd "}
5795 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.

```

5796 function self.orderedlist(items,tight,startnum)
5797     if not self.is_writing then return "" end
5798     local buffer = {}
5799     local num = startnum

```

```

5800     for _,item in ipairs(items) do
5801         if item ~= "" then
5802             buffer[#buffer + 1] = self.ordereditem(item,num)
5803         end
5804         if num ~= nil and item ~= "" then
5805             num = num + 1
5806         end
5807     end
5808     local contents = util.intersperse(buffer,"\n")
5809     if tight and options.tightLists then
5810         return {"\\markdownRenderer01BeginTight\n",contents,
5811             "\n\\markdownRenderer01EndTight "}
5812     else
5813         return {"\\markdownRenderer01Begin\n",contents,
5814             "\n\\markdownRenderer01End "}
5815     end
5816 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.

```

5817 function self.ordereditem(s,num)
5818     if num ~= nil then
5819         return {"\\markdownRenderer01ItemWithNumber{" ,num,"} ",s,
5820             "\\markdownRenderer01ItemEnd "}
5821     else
5822         return {"\\markdownRenderer01Item ",s,
5823             "\\markdownRenderer01ItemEnd "}
5824     end
5825 end

```

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

```

5826 function self.inline_html_comment(contents)
5827     if self.flatten_inlines then return contents end
5828     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5829 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.

```

5830 function self.inline_html_tag(contents)
5831     if self.flatten_inlines then return contents end
5832     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5833 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.

```
5834 function self.block_html_element(s)
5835   if not self.is_writing then return "" end
5836   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5837   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5838 end
```

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

```
5839 function self.emphasis(s)
5840   if self.flatten_inlines then return s end
5841   return {"\\markdownRendererEmphasis{" ,s,"}"}
5842 end
```

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

```
5843 function self.checkbox(f)
5844   if f == 1.0 then
5845     return "☒ "
5846   elseif f == 0.0 then
5847     return "☐ "
5848   else
5849     return "◻ "
5850   end
5851 end
```

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

```
5852 function self.strong(s)
5853   if self.flatten_inlines then return s end
5854   return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
5855 end
```

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

```
5856 function self.blockquote(s)
5857   if not self.is_writing then return "" end
5858   return {"\\markdownRendererBlockQuoteBegin\n" ,s,
5859     "\\markdownRendererBlockQuoteEnd "}
5860 end
```

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

```
5861 function self.verbatim(s)
5862   if not self.is_writing then return "" end
5863   s = s:gsub("\\n$", "")
```

```

5864     local name = util.cache_verbatim(options.cacheDir, s)
5865     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
5866 end

```

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

```

5867 function self.document(d)
5868     local buf = {"\\markdownRendererDocumentBegin\n", d}
5869
5870     -- pop all attributes
5871     table.insert(buf, self.pop_attributes())
5872
5873     table.insert(buf, "\\markdownRendererDocumentEnd")
5874
5875     return buf
5876 end

```

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

```

5877 local seen_identifiers = {}
5878 local key_value_regex = "([~= ]+)%s*=%s*(.*)"
5879 local function normalize_attributes(attributes, auto_identifiers)
5880     -- normalize attributes
5881     local normalized_attributes = {}
5882     local has_explicit_identifiers = false
5883     local key, value
5884     for _, attribute in ipairs(attributes or {}) do
5885         if attribute:sub(1, 1) == "#" then
5886             table.insert(normalized_attributes, attribute)
5887             has_explicit_identifiers = true
5888             seen_identifiers[attribute:sub(2)] = true
5889         elseif attribute:sub(1, 1) == "." then
5890             table.insert(normalized_attributes, attribute)
5891         else
5892             key, value = attribute:match(key_value_regex)
5893             if key:lower() == "id" then
5894                 table.insert(normalized_attributes, "#" .. value)
5895             elseif key:lower() == "class" then
5896                 local classes = {}
5897                 for class in value:gmatch("%S+") do
5898                     table.insert(classes, class)
5899                 end
5900                 table.sort(classes)
5901                 for _, class in ipairs(classes) do
5902                     table.insert(normalized_attributes, "." .. class)
5903                 end
5904             else
5905                 table.insert(normalized_attributes, attribute)

```

```

5906         end
5907     end
5908 end
5909
5910 -- if no explicit identifiers exist, add auto identifiers
5911 if not has_explicit_identifiers and auto_identifiers ~= nil then
5912     local seen_auto_identifiers = {}
5913     for _, auto_identifier in ipairs(auto_identifiers) do
5914         if seen_auto_identifiers[auto_identifier] == nil then
5915             seen_auto_identifiers[auto_identifier] = true
5916             if seen_identifiers[auto_identifier] == nil then
5917                 seen_identifiers[auto_identifier] = true
5918                 table.insert(normalized_attributes,
5919                     "#" .. auto_identifier)
5920             else
5921                 local auto_identifier_number = 1
5922                 while true do
5923                     local numbered_auto_identifier = auto_identifier .. "-"
5924                                             .. auto_identifier_number
5925                     if seen_identifiers[numbered_auto_identifier] == nil then
5926                         seen_identifiers[numbered_auto_identifier] = true
5927                         table.insert(normalized_attributes,
5928                             "#" .. numbered_auto_identifier)
5929                     break
5930                 end
5931                 auto_identifier_number = auto_identifier_number + 1
5932             end
5933         end
5934     end
5935 end
5936 end
5937
5938 -- sort and deduplicate normalized attributes
5939 table.sort(normalized_attributes)
5940 local seen_normalized_attributes = {}
5941 local deduplicated_normalized_attributes = {}
5942 for _, attribute in ipairs(normalized_attributes) do
5943     if seen_normalized_attributes[attribute] == nil then
5944         seen_normalized_attributes[attribute] = true
5945         table.insert(deduplicated_normalized_attributes, attribute)
5946     end
5947 end
5948
5949 return deduplicated_normalized_attributes
5950 end
5951
5952 function self.attributes(attributes, should_normalize_attributes)

```



```

5953     local normalized_attributes
5954     if should_normalize_attributes == false then
5955         normalized_attributes = attributes
5956     else
5957         normalized_attributes = normalize_attributes(attributes)
5958     end
5959
5960     local buf = {}
5961     local key, value
5962     for _, attribute in ipairs(normalized_attributes) do
5963         if attribute:sub(1, 1) == "#" then
5964             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
5965                                 attribute:sub(2), "}"}))
5966         elseif attribute:sub(1, 1) == "." then
5967             table.insert(buf, {"\\markdownRendererAttributeName{" ,
5968                                 attribute:sub(2), "}"}))
5969         else
5970             key, value = attribute:match(key_value_regex)
5971             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
5972                                 key, "}{" , value, "}"}))
5973         end
5974     end
5975
5976     return buf
5977 end

```

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

```

5978     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`.

```

5979     self.attribute_type_levels = {}
5980     setmetatable(self.attribute_type_levels,
5981                 { __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`.

```

5982     local function apply_attributes()
5983         local buf = {}
5984         for i = 1, #self.active_attributes do
5985             local start_output = self.active_attributes[i][3]
5986             if start_output ~= nil then
5987                 table.insert(buf, start_output)
5988             end
5989         end
5990         return buf

```

```

5991 end
5992
5993 local function tear_down_attributes()
5994     local buf = {}
5995     for i = #self.active_attributes, 1, -1 do
5996         local end_output = self.active_attributes[i][4]
5997         if end_output ~= nil then
5998             table.insert(buf, end_output)
5999         end
6000     end
6001     return buf
6002 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.

```

6003 function self.push_attributes(attribute_type, attributes,
6004                             start_output, end_output)
6005     local attribute_type_level = self.attribute_type_levels[attribute_type]
6006     self.attribute_type_levels[attribute_type] = attribute_type_level + 1
6007
6008     -- index attributes in a hash table for easy lookup
6009     attributes = attributes or {}
6010     for i = 1, #attributes do
6011         attributes[attributes[i]] = true
6012     end
6013
6014     local buf = {}
6015     -- handle slicing
6016     if attributes["#" .. self.slice_end_identifier] ~= nil and
6017        self.slice_end_type == "^" then
6018         if self.is_writing then
6019             table.insert(buf, self.undosep())
6020             table.insert(buf, tear_down_attributes())
6021         end
6022         self.is_writing = false
6023     end
6024     if attributes["#" .. self.slice_begin_identifier] ~= nil and
6025        self.slice_begin_type == "^" then
6026         table.insert(buf, apply_attributes())
6027         self.is_writing = true
6028     end
6029     if self.is_writing and start_output ~= nil then
6030         table.insert(buf, start_output)
6031     end

```

```

6032     table.insert(self.active_attributes,
6033                   {attribute_type, attributes,
6034                     start_output, end_output})
6035     return buf
6036 end
6037

```

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`).

```

6038 function self.pop_attributes(attribute_type)
6039     local buf = {}
6040     -- pop attributes until we find attributes of correct type
6041     -- or until no attributes remain
6042     local current_attribute_type = false
6043     while current_attribute_type ~= attribute_type and
6044           #self.active_attributes > 0 do
6045         local attributes, _, end_output
6046         current_attribute_type, attributes, _, end_output = table.unpack(
6047             self.active_attributes[#self.active_attributes])
6048         local attribute_type_level = self.attribute_type_levels[current_attribute_type]
6049         self.attribute_type_levels[current_attribute_type] = attribute_type_level - 1
6050         if self.is_writing and end_output ~= nil then
6051             table.insert(buf, end_output)
6052         end
6053         table.remove(self.active_attributes, #self.active_attributes)
6054         -- handle slicing
6055         if attributes["#" .. self.slice_end_identifier] ~= nil
6056            and self.slice_end_type == "$" then
6057             if self.is_writing then
6058                 table.insert(buf, self.undosep())
6059                 table.insert(buf, tear_down_attributes())
6060             end
6061             self.is_writing = false
6062         end
6063         if attributes["#" .. self.slice_begin_identifier] ~= nil and
6064            self.slice_begin_type == "$" then
6065             self.is_writing = true
6066             table.insert(buf, apply_attributes())
6067         end
6068     end
6069     return buf
6070 end

```

Create an auto identifier string by stripping and converting characters from string `s`.

```
6071 local function create_auto_identifier(s)
6072   local buffer = {}
6073   local prev_space = false
6074   local letter_found = false
6075
6076   for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
6077     local char = utf8.char(code)
6078
6079     -- Remove everything up to the first letter.
6080     if not letter_found then
6081       local is_letter = unicode.utf8.match(char, "%a")
6082       if is_letter then
6083         letter_found = true
6084       else
6085         goto continue
6086       end
6087     end
6088
6089     -- Remove all non-alphanumeric characters, except underscores, hyphens, and per
6090     if not unicode.utf8.match(char, "[%w_%-%.%s]") then
6091       goto continue
6092     end
6093
6094     -- Replace all spaces and newlines with hyphens.
6095     if unicode.utf8.match(char, "[%s\n]") then
6096       char = "-"
6097       if prev_space then
6098         goto continue
6099       else
6100         prev_space = true
6101       end
6102     else
6103       -- Convert all alphabetic characters to lowercase.
6104       char = unicode.utf8.lower(char)
6105       prev_space = false
6106     end
6107
6108     table.insert(buffer, char)
6109
6110     ::continue::
6111   end
6112
6113   if prev_space then
6114     table.remove(buffer)
6115   end
6116
```

```

6117     local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6118     return identifier
6119 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

6120 local function create_gfm_auto_identifier(s)
6121     local buffer = {}
6122     local prev_space = false
6123     local letter_found = false
6124
6125     for _, code in utf8.codes(uni_algos.normalize.NFC(s)) do
6126         local char = utf8.char(code)
6127
6128         -- Remove everything up to the first non-space.
6129         if not letter_found then
6130             local is_letter = unicode.utf8.match(char, "%S")
6131             if is_letter then
6132                 letter_found = true
6133             else
6134                 goto continue
6135             end
6136         end
6137
6138         -- Remove all non-alphanumeric characters, except underscores and hyphens.
6139         if not unicode.utf8.match(char, "[%w_-%s]") then
6140             prev_space = false
6141             goto continue
6142         end
6143
6144         -- Replace all spaces and newlines with hyphens.
6145         if unicode.utf8.match(char, "[%s\n]") then
6146             char = "-"
6147             if prev_space then
6148                 goto continue
6149             else
6150                 prev_space = true
6151             end
6152         else
6153             -- Convert all alphabetic characters to lowercase.
6154             char = unicode.utf8.lower(char)
6155             prev_space = false
6156         end
6157
6158         table.insert(buffer, char)
6159
6160     ::continue::

```

```

6161     end
6162
6163     if prev_space then
6164         table.remove(buffer)
6165     end
6166
6167     local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6168     return identifier
6169 end

```

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

```

6170 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6171 self.secend_text = "\n\\markdownRendererSectionEnd "
6172 function self.heading(s, level, attributes)
6173     local buf = {}
6174     local flat_text, inlines = table.unpack(s)
6175
6176     -- push empty attributes for implied sections
6177     while self.attribute_type_levels["heading"] < level - 1 do
6178         table.insert(buf,
6179             self.push_attributes("heading",
6180                 nil,
6181                 self.secbegin_text,
6182                 self.secend_text))
6183     end
6184
6185     -- pop attributes for sections that have ended
6186     while self.attribute_type_levels["heading"] >= level do
6187         table.insert(buf, self.pop_attributes("heading"))
6188     end
6189
6190     -- construct attributes for the new section
6191     local auto_identifiers = {}
6192     if self.options.autoIdentifiers then
6193         table.insert(auto_identifiers, create_auto_identifier(flat_text))
6194     end
6195     if self.options.gfmAutoIdentifiers then
6196         table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))
6197     end
6198     local normalized_attributes = normalize_attributes(attributes, auto_identifiers)
6199
6200     -- push attributes for the new section
6201     local start_output = {}
6202     local end_output = {}
6203     table.insert(start_output, self.secbegin_text)
6204     table.insert(end_output, self.secend_text)

```

```

6205
6206     table.insert(buf, self.push_attributes("heading",
6207                                         normalized_attributes,
6208                                         start_output,
6209                                         end_output))
6210     assert(self.attribute_type_levels["heading"] == level)
6211
6212     -- render the heading and its attributes
6213     if self.is_writing and #normalized_attributes > 0 then
6214         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\n")
6215         table.insert(buf, self.attributes(normalized_attributes, false))
6216     end
6217
6218     local cmd
6219     level = level + options.shiftHeadings
6220     if level <= 1 then
6221         cmd = "\\markdownRendererHeadingOne"
6222     elseif level == 2 then
6223         cmd = "\\markdownRendererHeadingTwo"
6224     elseif level == 3 then
6225         cmd = "\\markdownRendererHeadingThree"
6226     elseif level == 4 then
6227         cmd = "\\markdownRendererHeadingFour"
6228     elseif level == 5 then
6229         cmd = "\\markdownRendererHeadingFive"
6230     elseif level >= 6 then
6231         cmd = "\\markdownRendererHeadingSix"
6232     else
6233         cmd = ""
6234     end
6235     if self.is_writing then
6236         table.insert(buf, {cmd, "{", inlines, "}"})
6237     end
6238
6239     if self.is_writing and #normalized_attributes > 0 then
6240         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6241     end
6242
6243     return buf
6244 end

```

Define `writer->get_state` as a function that returns the current state of the writer, where the state of a writer are its mutable member variables.

```

6245 function self.get_state()
6246     return {
6247         is_writing=self.is_writing,
6248         flatten_inlines=self.flatten_inlines,

```

```

6249     active_attributes={table.unpack(self.active_attributes)},
6250   }
6251 end

```

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

```

6252 function self.set_state(s)
6253   local previous_state = self.get_state()
6254   for key, value in pairs(s) do
6255     self[key] = value
6256   end
6257   return previous_state
6258 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`.

```

6259 function self.defer_call(f)
6260   local previous_state = self.get_state()
6261   return function(...)
6262     local state = self.set_state(previous_state)
6263     local return_value = f(...)
6264     self.set_state(state)
6265     return return_value
6266   end
6267 end
6268
6269 return self
6270 end

```

3.1.4 Parsers

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

```

6271 local parsers = {}

```

3.1.4.1 Basic Parsers

```

6272 parsers.percent = P("%")
6273 parsers.at = P("@")
6274 parsers.comma = P(",")
6275 parsers.asterisk = P("*")
6276 parsers.dash = P("-")
6277 parsers.plus = P("+")
6278 parsers.underscore = P("_")
6279 parsers.period = P(".")
6280 parsers.hash = P("#")

```



```

6281 parsers.dollar           = P("$")
6282 parsers.ampersand        = P("&")
6283 parsers.backtick         = P("`")
6284 parsers.less              = P("<")
6285 parsers.more              = P(">")
6286 parsers.space            = P(" ")
6287 parsers.squote           = P("'")
6288 parsers.dquote           = P('"')
6289 parsers.lparent          = P("(")
6290 parsers.rparent          = P(")")
6291 parsers.lbracket         = P("[")
6292 parsers.rbracket         = P("]")
6293 parsers.lbrace           = P("{")
6294 parsers.rbrace           = P("}")
6295 parsers.circumflex       = P("^")
6296 parsers.slash            = P("/")
6297 parsers.equal            = P("=")
6298 parsers.colon            = P(":")
6299 parsers.semicolon       = P(";")
6300 parsers.exclamation      = P("!")
6301 parsers.pipe             = P("|")
6302 parsers.tilde            = P("~")
6303 parsers.backslash        = P("\\")
6304 parsers.tab              = P("\t")
6305 parsers.newline          = P("\n")
6306
6307 parsers.digit            = R("09")
6308 parsers.hexdigit         = R("09", "af", "AF")
6309 parsers.letter          = R("AZ", "az")
6310 parsers.alphanumeric     = R("AZ", "az", "09")
6311 parsers.keyword         = parsers.letter
6312                         * (parsers.alphanumeric + parsers.dash)^0
6313
6314 parsers.doubleasterisks   = P("**")
6315 parsers.doubleunderscores = P("__")
6316 parsers.doubletildes     = P("~~")
6317 parsers.fourspace       = P("    ")
6318
6319 parsers.any              = P(1)
6320 parsers.succeed         = P(true)
6321 parsers.fail            = P(false)
6322
6323 parsers.internal_punctuation = S(";, .?")
6324 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")

```

3.1.5 Unicode punctuation

This section documents the Unicode punctuation³² recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

(CommonMark Spec, Version 0.31.2 (2024-01-28))
--

```
6325 parsers.punctuation          = {}
6326 (function()
6327   local pathname = kpse.lookup("UnicodeData.txt")
6328   local file = assert(io.open(pathname, "r"),
6329     [[Could not open file "UnicodeData.txt"]])
6330   for line in file:lines() do
6331     local codepoint, major_category = line:match("^(%x+);[^\;]*;(%a)")
6332     if major_category == "P" or major_category == "S" then
6333       local code = unicode.utf8.char(tonumber(codepoint, 16))
6334       if parsers.punctuation[#code] == nil then
6335         parsers.punctuation[#code] = parsers.fail
6336       end
6337       local code_parser = parsers.succeed
6338       for i = 1, #code do
6339         local byte = code:sub(i, i)
6340         local byte_parser = S(byte)
6341         code_parser = code_parser
6342           * byte_parser
6343       end
6344       parsers.punctuation[#code] = parsers.punctuation[#code]
6345         + code_parser
6346     end
6347   end
6348   assert(file:close())
6349 end)()
6350
6351 parsers.escapable                = parsers.ascii_punctuation
6352 parsers.anyescaped               = parsers.backslash / " " * parsers.escapable
6353                                 + parsers.any
6354
6355 parsers.spacechar                = S("\t ")
6356 parsers.spacing                  = S(" \n\r\t")
6357 parsers.nonspacechar             = parsers.any - parsers.spacing
6358 parsers.optionalspace            = parsers.spacechar^0
6359
6360 parsers.normalchar               = parsers.any - (V("SpecialChar")
6361                                           + parsers.spacing)
```

³²See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

6362 parsers.eof                = -parsers.any
6363 parsers.nonindentspace      = parsers.space^-3 * - parsers.spacechar
6364 parsers.indent              = parsers.space^-3 * parsers.tab
6365                             + parsers.fourspace / ""
6366 parsers.linechar             = P(1 - parsers.newline)
6367
6368 parsers.blankline            = parsers.optionalspace
6369                             * parsers.newline / "\n"
6370 parsers.blanklines           = parsers.blankline^0
6371 parsers.skipblanklines        = (parsers.optionalspace * parsers.newline)^0
6372 parsers.indentedline         = parsers.indent / ""
6373                             * C(parsers.linechar^1 * parsers.newline^-
1)
6374 parsers.optionallyindentedline = parsers.indent^-1 / ""
6375                             * C(parsers.linechar^1 * parsers.newline^-
1)
6376 parsers.sp                    = parsers.spacing^0
6377 parsers.spnl                  = parsers.optionalspace
6378                             * (parsers.newline * parsers.optionalspace)^-
1
6379 parsers.line                  = parsers.linechar^0 * parsers.newline
6380 parsers.nonemptyline          = parsers.line - parsers.blankline

```

3.1.5.1 Parsers Used for Indentation

```

6381
6382 parsers.leader                = parsers.space^-3
6383

```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```

6384 local function has_trail(indent_table)
6385   return indent_table ~= nil and
6386     indent_table.trail ~= nil and
6387     next(indent_table.trail) ~= nil
6388 end
6389

```

Check if indent table `indent_table` has any indents.

```

6390 local function has_indents(indent_table)
6391   return indent_table ~= nil and
6392     indent_table.indents ~= nil and
6393     next(indent_table.indents) ~= nil
6394 end
6395

```

Add a trail `trail_info` to the indent table `indent_table`.

```

6396 local function add_trail(indent_table, trail_info)
6397   indent_table.trail = trail_info
6398   return indent_table

```

```
6399 end
6400
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
6401 local function remove_trail(indent_table)
6402     indent_table.trail = nil
6403     return indent_table
6404 end
6405
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
6406 local function update_indent_table(indent_table, new_indent, add)
6407     indent_table = remove_trail(indent_table)
6408
6409     if not has_indents(indent_table) then
6410         indent_table.indents = {}
6411     end
6412
6413
6414     if add then
6415         indent_table.indents[#indent_table.indents + 1] = new_indent
6416     else
6417         if indent_table.indents[#indent_table.indents].name == new_indent.name then
6418             indent_table.indents[#indent_table.indents] = nil
6419         end
6420     end
6421
6422     return indent_table
6423 end
6424
```

Remove an indent by its name `name`.

```
6425 local function remove_indent(name)
6426     local function remove_indent_level(s, i, indent_table) -- luacheck: ignore s i
6427         indent_table = update_indent_table(indent_table, {name=name}, false)
6428         return true, indent_table
6429     end
6430
6431     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6432 end
6433
```

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`.

```
6434 local function process_starter_spacing(indent, spacing, minimum, left_strip_length)
```

```

6435 left_strip_length = left_strip_length or 0
6436
6437 local count = 0
6438 local tab_value = 4 - (indent) % 4
6439
6440 local code_started, minimum_found = false, false
6441 local code_start, minimum_remainder = "", ""
6442
6443 local left_total_stripped = 0
6444 local full_remainder = ""
6445
6446 if spacing ~= nil then
6447     for i = 1, #spacing do
6448         local character = spacing:sub(i, i)
6449
6450         if character == "\t" then
6451             count = count + tab_value
6452             tab_value = 4
6453         elseif character == " " then
6454             count = count + 1
6455             tab_value = 4 - (1 - tab_value) % 4
6456         end
6457
6458         if (left_strip_length ~= 0) then
6459             local possible_to_strip = math.min(count, left_strip_length)
6460             count = count - possible_to_strip
6461             left_strip_length = left_strip_length - possible_to_strip
6462             left_total_stripped = left_total_stripped + possible_to_strip
6463         else
6464             full_remainder = full_remainder .. character
6465         end
6466
6467         if (minimum_found) then
6468             minimum_remainder = minimum_remainder .. character
6469         elseif (count >= minimum) then
6470             minimum_found = true
6471             minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6472         end
6473
6474         if (code_started) then
6475             code_start = code_start .. character
6476         elseif (count >= minimum + 4) then
6477             code_started = true
6478             code_start = code_start .. string.rep(" ", count - (minimum + 4))
6479         end
6480     end
6481 end

```

```

6482
6483   local remainder
6484   if (code_started) then
6485     remainder = code_start
6486   else
6487     remainder = string.rep(" ", count - minimum)
6488   end
6489
6490   local is_minimum = count >= minimum
6491   return {
6492     is_code = code_started,
6493     remainder = remainder,
6494     left_total_stripped = left_total_stripped,
6495     is_minimum = is_minimum,
6496     minimum_remainder = minimum_remainder,
6497     total_length = count,
6498     full_remainder = full_remainder
6499   }
6500 end
6501

```

Count the total width of all indents in the indent table `indent_table`.

```

6502 local function count_indent_tab_level(indent_table)
6503   local count = 0
6504   if not has_indents(indent_table) then
6505     return count
6506   end
6507
6508   for i=1, #indent_table.indents do
6509     count = count + indent_table.indents[i].length
6510   end
6511   return count
6512 end
6513

```

Count the total width of a delimiter `delimiter`.

```

6514 local function total_delimiter_length(delimiter)
6515   local count = 0
6516   if type(delimiter) == "string" then return #delimiter end
6517   for _, value in pairs(delimiter) do
6518     count = count + total_delimiter_length(value)
6519   end
6520   return count
6521 end
6522

```

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`.

```

6523 local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
6524     local last_trail = starter[1]
6525     local delimiter = starter[2]
6526     local raw_new_trail = starter[3]
6527
6528     if indent_type == "bq" and not breakable then
6529         indent_table.ignore_blockquote_blank = true
6530     end
6531
6532     if has_trail(indent_table) then
6533         local trail = indent_table.trail
6534         if trail.is_code then
6535             return false
6536         end
6537         last_trail = trail.remainder
6538     else
6539         local sp = process_starter_spacing(0, last_trail, 0, 0)
6540
6541         if sp.is_code then
6542             return false
6543         end
6544         last_trail = sp.remainder
6545     end
6546
6547     local preceding_indentation = count_indent_tab_level(indent_table) % 4
6548     local last_trail_length = #last_trail
6549     local delimiter_length = total_delimiter_length(delimiter)
6550
6551     local total_indent_level = preceding_indentation + last_trail_length + delimiter_le
6552
6553     local sp = {}
6554     if not is_blank then
6555         sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6556     end
6557
6558     local del_trail_length = sp.left_total_stripped
6559     if is_blank then
6560         del_trail_length = 1
6561     elseif not sp.is_code then
6562         del_trail_length = del_trail_length + #sp.remainder
6563     end
6564
6565     local indent_length = last_trail_length + delimiter_length + del_trail_length
6566     local new_indent_info = {name=indent_type, length=indent_length}
6567
6568     indent_table = update_indent_table(indent_table, new_indent_info, true)
6569     indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,

```

```

6570                                     full_remainder=sp.full_remainder})
6571
6572     return true, indent_table
6573 end
6574

```

Return the pattern corresponding with the indent name `name`.

```

6575 local function decode_pattern(name)
6576     local delimiter = parsers.succeed
6577     if name == "bq" then
6578         delimiter = parsers.more
6579     end
6580
6581     return C(parsers.optionalspace) * C(delimiter) * C(parsers.optionalspace) * Cp()
6582 end
6583

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

6584 local function left_blank_starter(indent_table)
6585     local blank_starter_index
6586
6587     if not has_indents(indent_table) then
6588         return
6589     end
6590
6591     for i = #indent_table.indents,1,-1 do
6592         local value = indent_table.indents[i]
6593         if value.name == "li" then
6594             blank_starter_index = i
6595         else
6596             break
6597         end
6598     end
6599
6600     return blank_starter_index
6601 end
6602

```

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.

```

6603 local function traverse_indent(s, i, indent_table, is_optional, is_blank, current_line)
6604     local new_index = i
6605
6606     local preceding_indentation = 0

```



```

6607 local current_trail = {}
6608
6609 local blank_starter = left_blank_starter(indent_table)
6610
6611 if current_line_indents == nil then
6612     current_line_indents = {}
6613 end
6614
6615 for index = 1,#indent_table.indents do
6616     local value = indent_table.indents[index]
6617     local pattern = decode_pattern(value.name)
6618
6619     -- match decoded pattern
6620     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6621     if new_indent_info == nil then
6622         local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
6623         if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end then
6624             return is_optional, new_index, current_trail, current_line_indents
6625         end
6626
6627         return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optional)
6628     end
6629
6630     local raw_last_trail = new_indent_info[1]
6631     local delimiter = new_indent_info[2]
6632     local raw_new_trail = new_indent_info[3]
6633     local next_index = new_indent_info[4]
6634
6635     local space_only = delimiter == ""
6636
6637     -- check previous trail
6638     if not space_only and next(current_trail) == nil then
6639         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6640         current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.total_length,
6641             full_remainder=sp.full_remainder}
6642     end
6643
6644     if next(current_trail) ~= nil then
6645         if not space_only and current_trail.is_code then
6646             return is_optional, new_index, current_trail, current_line_indents
6647         end
6648         if current_trail.internal_remainder ~= nil then
6649             raw_last_trail = current_trail.internal_remainder
6650         end
6651     end
6652
6653     local raw_last_trail_length = 0

```

```

6654     local delimiter_length = 0
6655
6656     if not space_only then
6657         delimiter_length = #delimiter
6658         raw_last_trail_length = #raw_last_trail
6659     end
6660
6661     local total_indent_level = preceding_indentation + raw_last_trail_length + delimi
6662
6663     local spacing_to_process
6664     local minimum = 0
6665     local left_strip_length = 0
6666
6667     if not space_only then
6668         spacing_to_process = raw_new_trail
6669         left_strip_length = 1
6670     else
6671         spacing_to_process = raw_last_trail
6672         minimum = value.length
6673     end
6674
6675     local sp = process_starter_spacing(total_indent_level, spacing_to_process, minimu
6676
6677     if space_only and not sp.is_minimum then
6678         return is_optional or (is_blank and blank_starter <= index), new_index, current
6679     end
6680
6681     local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_st
6682
6683     -- update info for the next pattern
6684     if not space_only then
6685         preceding_indentation = preceding_indentation + indent_length
6686     else
6687         preceding_indentation = preceding_indentation + value.length
6688     end
6689
6690     current_trail = {is_code=sp.is_code, remainder=sp.remainder, internal_remainder=s
6691         total_length=sp.total_length, full_remainder=sp.full_remainder}
6692
6693     current_line_indents[#current_line_indents + 1] = new_indent_info
6694     new_index = next_index
6695 end
6696
6697 return true, new_index, current_trail, current_line_indents
6698 end
6699

```

Check if a code trail is expected.

```
6700 local function check_trail(expect_code, is_code)
6701   return (expect_code and is_code) or (not expect_code and not is_code)
6702 end
6703
```

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`.

```
6704 local function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_rema
6705   local is_code
6706   local remainder
6707
6708   if has_trail(indent_table) then
6709     local trail = indent_table.trail
6710     is_code = trail.is_code
6711     if is_code then
6712       remainder = trail.remainder
6713     else
6714       remainder = trail.full_remainder
6715     end
6716   else
6717     local sp = process_starter_spacing(0, spacing, 0, 0)
6718     is_code = sp.is_code
6719     if is_code then
6720       remainder = sp.remainder
6721     else
6722       remainder = sp.full_remainder
6723     end
6724   end
6725
6726   local result = check_trail(expect_code, is_code)
6727   if omit_remainder then
6728     return result
6729   end
6730   return result, remainder
6731 end
6732
```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
6733 local function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:
6734   local trail
6735
6736   if has_trail(indent_table) then
6737     trail = indent_table.trail
6738   else
6739     trail = process_starter_spacing(0, spacing, 0, 0)
```

```

6740 end
6741
6742 local total_length = trail.total_length
6743 if total_length == nil then
6744     return false
6745 end
6746
6747 return min <= total_length and total_length <= max
6748 end
6749

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

6750 local function check_continuation_indentation(s, i, indent_table, is_optional, is_bla
6751     if not has_indents(indent_table) then
6752         return true
6753     end
6754
6755     local passes, new_index, current_trail, current_line_indents =
6756         traverse_indent(s, i, indent_table, is_optional, is_blank)
6757
6758     if passes then
6759         indent_table.current_line_indents = current_line_indents
6760         indent_table = add_trail(indent_table, current_trail)
6761         return new_index, indent_table
6762     end
6763     return false
6764 end
6765

```

Get name of the last indent from the `indent_table`.

```

6766 local function get_last_indent_name(indent_table)
6767     if has_indents(indent_table) then
6768         return indent_table.indents[#indent_table.indents].name
6769     end
6770 end
6771

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```

6772 local function remove_remainder_if_blank(indent_table, remainder)
6773     if get_last_indent_name(indent_table) == "li" then
6774         return ""
6775     end
6776     return remainder
6777 end
6778

```

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.

```
6779 local function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore
6780   if trail == nil then
6781     trail = process_starter_spacing(0, spacing, 0, 0)
6782   end
6783
6784   if trail_type == "non-code" then
6785     return check_trail(false, trail.is_code)
6786   end
6787   if trail_type == "code" then
6788     return check_trail(true, trail.is_code)
6789   end
6790   if trail_type == "full-code" then
6791     if (trail.is_code) then
6792       return i, trail.remainder
6793     end
6794     return i, ""
6795   end
6796   if trail_type == "full-any" then
6797     return i, trail.internal_remainder
6798   end
6799 end
6800
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
6801 local function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
6802   if is_freezing then
6803     if indent_table.is_trail_frozen then
6804       indent_table.trail = indent_table.frozen_trail
6805     else
6806       indent_table.frozen_trail = indent_table.trail
6807       indent_table.is_trail_frozen = true
6808     end
6809   else
6810     indent_table.frozen_trail = nil
6811     indent_table.is_trail_frozen = false
6812   end
6813   return true, indent_table
6814 end
6815
```

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`.

```
6816 local function check_continuation_indentation_and_trail(s, i, indent_table, is_option
6817                                                         reset_rem, omit_remainder)
```

```

6818 if not has_indents(indent_table) then
6819     local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6820     local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, tra
6821     if remainder == nil then
6822         if result then
6823             return new_index
6824         end
6825         return false
6826     end
6827     if result then
6828         return new_index, remainder
6829     end
6830     return false
6831 end
6832
6833 local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt
6834
6835 if passes then
6836     local spacing
6837     if current_trail == nil then
6838         local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s,
6839         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6840         new_index = newer_index
6841         spacing = newer_spacing
6842     else
6843         spacing = current_trail.remainder
6844     end
6845     local result, remainder = check_trail_type(s, new_index, current_trail, spacing,
6846     if remainder == nil or omit_remainder then
6847         if result then
6848             return new_index
6849         end
6850         return false
6851     end
6852
6853     if is_blank and reset_rem then
6854         remainder = remove_remainder_if_blank(indent_table, remainder)
6855     end
6856     if result then
6857         return new_index, remainder
6858     end
6859     return false
6860 end
6861 return false
6862 end
6863

```

The following patterns check whitespace indentation at the start of a block.

```
6864 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(false), che
6865
6866 parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals
6867
6868 parsers.check_code_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)
6869
6870 parsers.check_trail_length_range = function(min, max)
6871   return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
6872 end
6873
6874 parsers.check_trail_length = function(n)
6875   return parsers.check_trail_length_range(n, n)
6876 end
6877
```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```
6878 parsers.freeze_trail = Cg(Cmt(Cb("indent_info") * Cc(true), trail_freezing), "indent_
6879
6880 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde
6881
```

The following patterns check indentation in continuation lines as defined by the container start.

```
6882 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false), check_continuation_
6883
6884 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_
6885
6886 parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch
6887
```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```
6888
6889 parsers.check_minimal_indent_and_trail = Cmt( Cb("indent_info")
6890           * Cc(false) * Cc(false) * Cc("non-
        code") * Cc(true),
6891           check_continuation_indentation_and_trail)
6892
6893 parsers.check_minimal_indent_and_code_trail = Cmt( Cb("indent_info")
6894           * Cc(false) * Cc(false) * Cc("code")
6895           check_continuation_indentation_and_t
6896
6897 parsers.check_minimal_blank_indent_and_full_code_trail = Cmt( Cb("indent_info")
6898           * Cc(false) * Cc(true) *
        code") * Cc(true),
6899           check_continuation_indent
```

```

6900
6901 parsers.check_minimal_indent_and_any_trail = Cmt( Cb("indent_info")
6902                                     * Cc(false) * Cc(false) * Cc("full-
any") * Cc(true) * Cc(false),
6903                                     check_continuation_indentation_and_tr
6904
6905 parsers.check_minimal_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6906                                     * Cc(false) * Cc(true) * Cc("fu
any") * Cc(true) * Cc(false),
6907                                     check_continuation_indentation
6908
6909 parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt( Cb("indent_info")
6910                                     * Cc(false) * Cc(true) * Cc("
any") * Cc(true) * Cc(true),
6911                                     check_continuation_indentation
6912
6913 parsers.check_optional_indent_and_any_trail = Cmt( Cb("indent_info")
6914                                     * Cc(true) * Cc(false) * Cc("full-
any") * Cc(true) * Cc(false),
6915                                     check_continuation_indentation_and_tr
6916
6917 parsers.check_optional_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6918                                     * Cc(true) * Cc(true) * Cc("ful
any") * Cc(true) * Cc(false),
6919                                     check_continuation_indentation
6920

```

The following patterns specify behaviour around newlines.

```

6921
6922 parsers.spnlc_noexc = parsers.optionalspace
6923                                     * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
1
6924
6925 parsers.spnlc = parsers.optionalspace
6926                                     * (V("EndlineNoSub"))^-1
6927
6928 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
6929                                     + parsers.spacechar^1
6930
6931 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6932
6933 % \end{macrocode}
6934 % \begin{figure}
6935 % \hspace*{-0.1\textwidth}
6936 % \begin{minipage}{1.2\textwidth}
6937 % \centering
6938 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6939 % \node[state, initial by diamond, accepting] (noop) {initial};

```



```

6940 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
6941 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
6942 % \node[state] (comment) [below=of noop] {comment};
6943 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
6944 % \node[state] (blank_line) [below right=of comment] {blank line};
6945 % \path[->]
6946 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [^\
6947 %     edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
6948 %     edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
6949 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [^\wedge$$\drsh
6950 %     edge [bend left=10] node {match $\drsh$} (leading_spaces)
6951 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$
6952 %     edge [bend right=90] node [right] {match \textbackslash} (odd_back
6953 %     edge [bend left=10] node {match \%} (comment)
6954 %     edge [bend right=10] node {$\epsilon$} (blank_line)
6955 %     edge [bend left=10] node [align=center, right=0.3cm] {match [^\we
6956 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
6957 %     edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\
6958 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
6959 %     edge [bend right=10] node [align=center, above left=-
        0.3cm, xshift=0.1cm] {match [^\wedge$\textbackslash}\\for \%, capture \textbackslash
6960 % (even_backslash) edge [bend left=10] node {$\epsilon$} (noop);
6961 % \end{tikzpicture}
6962 % \caption{A pushdown automaton that recognizes \TeX{} comments}
6963 % \label{fig:commented_line}
6964 % \end{minipage}
6965 % \end{figure}
6966 % \begin{markdown}
6967 %
6968 % The \luamdef{parsers.commented_line}^1 parser recognizes the regular
6969 % language of \TeX{} comments, see an equivalent finite automaton in Figure
6970 % <#fig:commented_line>.
6971 %
6972 % \end{markdown}
6973 % \begin{macrocode}
6974 parsers.comment_line_letter = parsers.linechar
6975                               + parsers.newline
6976                               - parsers.backslash
6977                               - parsers.percent
6978 parsers.comment_line         = Cg(Cc(""), "backslashes")
6979                               * ((#(parsers.comment_line_letter
6980                                   - parsers.newline)
6981                                   * Cb("backslashes")
6982                                   * Cs(parsers.comment_line_letter
6983                                       - parsers.newline)^1 -- initial
6984                                       * Cg(Cc(""), "backslashes"))
6985                               + #(parsers.backslash * parsers.backslash)

```

```

6986 * Cg((parsers.backslash -- even backslash
6987 * parsers.backslash)^1, "backslashes")
6988 + (parsers.backslash
6989 * (#parsers.percent
6990 * Cb("backslashes")
6991 / function(backslashes)
6992 return string.rep("\\", #backslashes / 2)
6993 end
6994 * C(parsers.percent)
6995 + #parsers.commented_line_letter
6996 * Cb("backslashes")
6997 * Cc("\\")
6998 * C(parsers.commented_line_letter))
6999 * Cg(Cc(""), "backslashes"))^0
7000 * (#parsers.percent
7001 * Cb("backslashes")
7002 / function(backslashes)
7003 return string.rep("\\", #backslashes / 2)
7004 end
7005 * ((parsers.percent -- comment
7006 * parsers.line
7007 * #parsers.blankline) -- blank line
7008 / "\n"
7009 + parsers.percent -- comment
7010 * parsers.line
7011 * parsers.optionalspace) -- leading tabs and space
7012 + (#parsers.newline)
7013 * Cb("backslashes")
7014 * C(parsers.newline))
7015
7016 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7017 - parsers.blankline)^0
7018
7019 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7020 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7021 parsers.attribute_key = (parsers.attribute_key_char
7022 - parsers.dash - parsers.digit)
7023 * parsers.attribute_key_char^0
7024 parsers.attribute_value = ( (parsers.dquote / "\"")
7025 * (parsers.anyescaped - parsers.dquote)^0
7026 * (parsers.dquote / "\""))
7027 + ( (parsers.squote / "\"")
7028 * (parsers.anyescaped - parsers.squote)^0
7029 * (parsers.squote / "\""))
7030 + ( parsers.anyescaped - parsers.dquote - parsers.rbra
7031 - parsers.space)^0
7032 parsers.attribute_identifier = parsers.attribute_key_char^1

```

```

7033 parsers.attribute_classname = parsers.letter
7034                               * parsers.attribute_key_char^0
7035 parsers.attribute_raw         = parsers.attribute_raw_char^1
7036
7037 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7038                   + C( parsers.hash
7039                       * parsers.attribute_identifier)
7040                   + C( parsers.period
7041                       * parsers.attribute_classname)
7042                   + Cs( parsers.attribute_key
7043                       * parsers.optionalspace * parsers.equal * parsers.optionalspace
7044                       * parsers.attribute_value)
7045 parsers.attributes = parsers.lbrace
7046                   * parsers.optionalspace
7047                   * parsers.attribute
7048                   * (parsers.spacechar^1
7049                     * parsers.attribute)^0
7050                   * parsers.optionalspace
7051                   * parsers.rbrace
7052
7053
7054 parsers.raw_attribute = parsers.lbrace
7055                       * parsers.optionalspace
7056                       * parsers.equal
7057                       * C(parsers.attribute_raw)
7058                       * parsers.optionalspace
7059                       * parsers.rbrace
7060
7061 -- block followed by 0 or more optionally
7062 -- indented blocks with first line indented.
7063 parsers.indented_blocks = function(bl)
7064   return Cs( bl
7065             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
7066             * (parsers.blankline^1 + parsers.eof) )
7067 end

```

3.1.5.2 Parsers Used for HTML Entities

```

7068 local function repeat_between(pattern, min, max)
7069   return -pattern^(max + 1) * pattern^min
7070 end
7071
7072 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7073                   * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
7074 parsers.decentity = parsers.ampersand * parsers.hash
7075                   * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
7076 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)

```

```

7077             * parsers.semicolon
7078
7079 parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
7080                       + parsers.decentity / entities.dec_entity
7081                       + parsers.tagentity / entities.char_entity

```

3.1.5.3 Parsers Used for Markdown Lists

```

7082 parsers.bullet = function(bullet_char, interrupting)
7083   local allowed_end
7084   if interrupting then
7085     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7086   else
7087     allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7088   end
7089   return parsers.check_trail
7090     * Ct(C(bullet_char) * Cc(""))
7091     * allowed_end
7092 end
7093
7094 local function tickbox(interior)
7095   return parsers.optionalspace * parsers.lbracket
7096     * interior * parsers.rbracket * parsers.spacechar^1
7097 end
7098
7099 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7100 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7101 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7102

```

3.1.5.4 Parsers Used for Markdown Code Spans

```

7103 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7104
7105 local function captures_equal_length(_,i,a,b)
7106   return #a == #b and i
7107 end
7108
7109 parsers.closeticks = Cmt(C(parsers.backtick^1)
7110                        * Cb("ticks"), captures_equal_length)
7111
7112 parsers.intickschar = (parsers.any - S("\n\r`"))
7113                      + V("NoSoftLineBreakEndline")
7114                      + (parsers.backtick^1 - parsers.closeticks)
7115
7116 local function process_inticks(s)
7117   s = s:gsub("\n", " ")
7118   s = s:gsub("^ (.*) $", "%1")

```

```

7119     return s
7120 end
7121
7122 parsers.inticks = parsers.openticks
7123             * C(parsers.space^0)
7124             * parsers.closeticks
7125             + parsers.openticks
7126             * Cs(Cs(parsers.intickschar^0) / process_inticks)
7127             * parsers.closeticks
7128

```

3.1.5.5 Parsers Used for HTML

```

7129 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
7130 parsers.keyword_exact = function(s)
7131     local parser = P(0)
7132     for i=1,#s do
7133         local c = s:sub(i,i)
7134         local m = c .. upper(c)
7135         parser = parser * S(m)
7136     end
7137     return parser
7138 end
7139
7140 parsers.special_block_keyword =
7141     parsers.keyword_exact("pre") +
7142     parsers.keyword_exact("script") +
7143     parsers.keyword_exact("style") +
7144     parsers.keyword_exact("textarea")
7145
7146 parsers.block_keyword =
7147     parsers.keyword_exact("address") +
7148     parsers.keyword_exact("article") +
7149     parsers.keyword_exact("aside") +
7150     parsers.keyword_exact("base") +
7151     parsers.keyword_exact("basefont") +
7152     parsers.keyword_exact("blockquote") +
7153     parsers.keyword_exact("body") +
7154     parsers.keyword_exact("caption") +
7155     parsers.keyword_exact("center") +
7156     parsers.keyword_exact("col") +
7157     parsers.keyword_exact("colgroup") +
7158     parsers.keyword_exact("dd") +
7159     parsers.keyword_exact("details") +
7160     parsers.keyword_exact("dialog") +
7161     parsers.keyword_exact("dir") +
7162     parsers.keyword_exact("div") +

```

7163 parsers.keyword_exact("dl") +
7164 parsers.keyword_exact("dt") +
7165 parsers.keyword_exact("fieldset") +
7166 parsers.keyword_exact("figcaption") +
7167 parsers.keyword_exact("figure") +
7168 parsers.keyword_exact("footer") +
7169 parsers.keyword_exact("form") +
7170 parsers.keyword_exact("frame") +
7171 parsers.keyword_exact("frameset") +
7172 parsers.keyword_exact("h1") +
7173 parsers.keyword_exact("h2") +
7174 parsers.keyword_exact("h3") +
7175 parsers.keyword_exact("h4") +
7176 parsers.keyword_exact("h5") +
7177 parsers.keyword_exact("h6") +
7178 parsers.keyword_exact("head") +
7179 parsers.keyword_exact("header") +
7180 parsers.keyword_exact("hr") +
7181 parsers.keyword_exact("html") +
7182 parsers.keyword_exact("iframe") +
7183 parsers.keyword_exact("legend") +
7184 parsers.keyword_exact("li") +
7185 parsers.keyword_exact("link") +
7186 parsers.keyword_exact("main") +
7187 parsers.keyword_exact("menu") +
7188 parsers.keyword_exact("menuitem") +
7189 parsers.keyword_exact("nav") +
7190 parsers.keyword_exact("noframes") +
7191 parsers.keyword_exact("ol") +
7192 parsers.keyword_exact("optgroup") +
7193 parsers.keyword_exact("option") +
7194 parsers.keyword_exact("p") +
7195 parsers.keyword_exact("param") +
7196 parsers.keyword_exact("section") +
7197 parsers.keyword_exact("source") +
7198 parsers.keyword_exact("summary") +
7199 parsers.keyword_exact("table") +
7200 parsers.keyword_exact("tbody") +
7201 parsers.keyword_exact("td") +
7202 parsers.keyword_exact("tfoot") +
7203 parsers.keyword_exact("th") +
7204 parsers.keyword_exact("thead") +
7205 parsers.keyword_exact("title") +
7206 parsers.keyword_exact("tr") +
7207 parsers.keyword_exact("track") +
7208 parsers.keyword_exact("ul")
7209

```

7210 -- end conditions
7211 parsers.html_blankline_end_condition = parsers.linechar^0
7212         * ( parsers.newline
7213         * (parsers.check_minimal_blank_indent_and_any
7214         * #parsers.blankline
7215         + parsers.check_minimal_indent_and_any_trai
7216         * parsers.linechar^1)^0
7217         * (parsers.newline^-1 / "")
7218
7219 local function remove_trailing_blank_lines(s)
7220     return s:gsub("[\n\r]+%s*$", "")
7221 end
7222
7223 parsers.html_until_end = function(end_marker)
7224     return Cs(Cs((parsers.newline
7225         * (parsers.check_minimal_blank_indent_and_any_trail
7226         * #parsers.blankline
7227         + parsers.check_minimal_indent_and_any_trail)
7228         + parsers.linechar - end_marker)^0
7229         * parsers.linechar^0 * parsers.newline^-1)
7230         / remove_trailing_blank_lines)
7231 end
7232
7233 -- attributes
7234 parsers.html_attribute_spacing = parsers.optionalspace
7235         * V("NoSoftLineBreakEndline")
7236         * parsers.optionalspace
7237         + parsers.spacechar^1
7238
7239 parsers.html_attribute_name = (parsers.letter + parsers.colon + parsers.underscore)
7240         * (parsers.alphanumeric + parsers.colon + parsers.undersco
7241         + parsers.period + parsers.dash)^0
7242
7243 parsers.html_attribute_value = parsers.squote
7244         * (parsers.linechar - parsers.squote)^0
7245         * parsers.squote
7246         + parsers.dquote
7247         * (parsers.linechar - parsers.dquote)^0
7248         * parsers.dquote
7249         + ( parsers.any - parsers.spacechar - parsers.newline
7250         - parsers.dquote - parsers.squote - parsers.backtick
7251         - parsers.equal - parsers.less - parsers.more)^1
7252
7253 parsers.html_inline_attribute_value = parsers.squote
7254         * (V("NoSoftLineBreakEndline")
7255         + parsers.any
7256         - parsers.blankline^2

```

```

7257         - parsers.squote)^0
7258     * parsers.squote
7259     + parsers.dquote
7260     * (V("NoSoftLineBreakEndline")
7261       + parsers.any
7262       - parsers.blankline^2
7263       - parsers.dquote)^0
7264     * parsers.dquote
7265     + (parsers.any - parsers.spacechar - parsers.newline
7266       - parsers.dquote - parsers.squote - parsers.backslash
7267       - parsers.equal - parsers.less - parsers.more)^0
7268
7269 parsers.html_attribute_value_specification = parsers.optionalspace
7270                                           * parsers.equal
7271                                           * parsers.optionalspace
7272                                           * parsers.html_attribute_value
7273
7274 parsers.html_spnl = parsers.optionalspace
7275                   * (V("NoSoftLineBreakEndline") * parsers.optionalspace)^-
7276                   1
7277
7277 parsers.html_inline_attribute_value_specification = parsers.html_spnl
7278                                                   * parsers.equal
7279                                                   * parsers.html_spnl
7280                                                   * parsers.html_inline_attribute_value_specification
7281
7282 parsers.html_attribute = parsers.html_attribute_spacing
7283                       * parsers.html_attribute_name
7284                       * parsers.html_inline_attribute_value_specification^-
7285                       1
7286
7286 parsers.html_non_newline_attribute = parsers.spacechar^1
7287                                   * parsers.html_attribute_name
7288                                   * parsers.html_attribute_value_specification^-
7289                                   1
7289
7290 parsers.nested_breaking_blank = parsers.newline
7291                               * parsers.check_minimal_blank_indent
7292                               * parsers.blankline
7293
7294 parsers.html_comment_start = P("<!--")
7295
7296 parsers.html_comment_end = P("-->")
7297
7298 parsers.html_comment = Cs( parsers.html_comment_start
7299                          * parsers.html_until_end(parsers.html_comment_end))
7300

```



```

7301 parsers.html_inline_comment = (parsers.html_comment_start / "")
7302                               * -P(">") * -P("->")
7303                               * Cs((V("NoSoftLineBreakEndline") + parsers.any
7304                                   - parsers.nested_breaking_blank - parsers.html_commen
7305                               * (parsers.html_comment_end / ""))
7306
7307 parsers.html_cdatasection_start = P("<![CDATA[(")
7308
7309 parsers.html_cdatasection_end = P("]]>")
7310
7311 parsers.html_cdatasection = Cs( parsers.html_cdatasection_start
7312                               * parsers.html_until_end(parsers.html_cdatasection_end))
7313
7314 parsers.html_inline_cdatasection = parsers.html_cdatasection_start
7315                               * Cs(V("NoSoftLineBreakEndline") + parsers.any
7316                                   - parsers.nested_breaking_blank - parsers.html_
7317                               * parsers.html_cdatasection_end)
7318
7319 parsers.html_declaration_start = P("<!") * parsers.letter
7320
7321 parsers.html_declaration_end = P(">")
7322
7323 parsers.html_declaration = Cs( parsers.html_declaration_start
7324                               * parsers.html_until_end(parsers.html_declaration_end))
7325
7326 parsers.html_inline_declaration = parsers.html_declaration_start
7327                               * Cs(V("NoSoftLineBreakEndline") + parsers.any
7328                                   - parsers.nested_breaking_blank - parsers.html_de
7329                               * parsers.html_declaration_end)
7330
7331 parsers.html_instruction_start = P("<?")
7332
7333 parsers.html_instruction_end = P("?>")
7334
7335 parsers.html_instruction = Cs( parsers.html_instruction_start
7336                               * parsers.html_until_end(parsers.html_instruction_end))
7337
7338 parsers.html_inline_instruction = parsers.html_instruction_start
7339                               * Cs(V("NoSoftLineBreakEndline") + parsers.any
7340                                   - parsers.nested_breaking_blank - parsers.html_in
7341                               * parsers.html_instruction_end)
7342
7343 parsers.html_blankline = parsers.newline
7344                       * parsers.optionalspace
7345                       * parsers.newline
7346
7347 parsers.html_tag_start = parsers.less

```

```

7348
7349 parsers.html_tag_closing_start = parsers.less
7350                                 * parsers.slash
7351
7352 parsers.html_tag_end = parsers.html_spnl
7353                        * parsers.more
7354
7355 parsers.html_empty_tag_end = parsers.html_spnl
7356                            * parsers.slash
7357                            * parsers.more
7358
7359 -- opening tags
7360 parsers.html_any_open_inline_tag = parsers.html_tag_start
7361                                 * parsers.keyword
7362                                 * parsers.html_attribute^0
7363                                 * parsers.html_tag_end
7364
7365 parsers.html_any_open_tag = parsers.html_tag_start
7366                            * parsers.keyword
7367                            * parsers.html_non_newline_attribute^0
7368                            * parsers.html_tag_end
7369
7370 parsers.html_open_tag = parsers.html_tag_start
7371                       * parsers.block_keyword
7372                       * parsers.html_attribute^0
7373                       * parsers.html_tag_end
7374
7375 parsers.html_open_special_tag = parsers.html_tag_start
7376                               * parsers.special_block_keyword
7377                               * parsers.html_attribute^0
7378                               * parsers.html_tag_end
7379
7380 -- incomplete tags
7381 parsers.incomplete_tag_following = parsers.spacechar
7382                                 + parsers.more
7383                                 + parsers.slash * parsers.more
7384                                 + #(parsers.newline + parsers.eof)
7385
7386 parsers.incomplete_special_tag_following = parsers.spacechar
7387                                           + parsers.more
7388                                           + #(parsers.newline + parsers.eof)
7389
7390 parsers.html_incomplete_open_tag = parsers.html_tag_start
7391                                   * parsers.block_keyword
7392                                   * parsers.incomplete_tag_following
7393
7394 parsers.html_incomplete_open_special_tag = parsers.html_tag_start

```

```

7395             * parsers.special_block_keyword
7396             * parsers.incomplete_special_tag_following
7397
7398 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7399             * parsers.block_keyword
7400             * parsers.incomplete_tag_following
7401
7402 parsers.html_incomplete_close_special_tag = parsers.html_tag_closing_start
7403             * parsers.special_block_keyword
7404             * parsers.incomplete_tag_following
7405
7406 -- closing tags
7407 parsers.html_close_tag = parsers.html_tag_closing_start
7408             * parsers.block_keyword
7409             * parsers.html_tag_end
7410
7411 parsers.html_any_close_tag = parsers.html_tag_closing_start
7412             * parsers.keyword
7413             * parsers.html_tag_end
7414
7415 parsers.html_close_special_tag = parsers.html_tag_closing_start
7416             * parsers.special_block_keyword
7417             * parsers.html_tag_end
7418
7419 -- empty tags
7420 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7421             * parsers.keyword
7422             * parsers.html_attribute^0
7423             * parsers.html_empty_tag_end
7424
7425 parsers.html_any_empty_tag = parsers.html_tag_start
7426             * parsers.keyword
7427             * parsers.html_non_newline_attribute^0
7428             * parsers.optionalspace
7429             * parsers.slash
7430             * parsers.more
7431
7432 parsers.html_empty_tag = parsers.html_tag_start
7433             * parsers.block_keyword
7434             * parsers.html_attribute^0
7435             * parsers.html_empty_tag_end
7436
7437 parsers.html_empty_special_tag = parsers.html_tag_start
7438             * parsers.special_block_keyword
7439             * parsers.html_attribute^0
7440             * parsers.html_empty_tag_end
7441

```

```

7442 parsers.html_incomplete_blocks = parsers.html_incomplete_open_tag
7443                                     + parsers.html_incomplete_open_special_tag
7444                                     + parsers.html_incomplete_close_tag
7445
7446 -- parse special html blocks
7447 parsers.html_blankline_ending_special_block_opening = (parsers.html_close_special_tag
7448                                                         + parsers.html_empty_special_tag
7449                                                         * #(parsers.optionalspace
7450                                                         * (parsers.newline + parsers.e
7451
7452 parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_b
7453                                               * parsers.html_blankline_end_condition
7454
7455 parsers.html_special_block_opening = parsers.html_incomplete_open_special_tag
7456                                     - parsers.html_empty_special_tag
7457
7458 parsers.html_closing_special_block = parsers.html_special_block_opening
7459                                     * parsers.html_until_end(parsers.html_close_speci
7460
7461 parsers.html_special_block = parsers.html_blankline_ending_special_block
7462                             + parsers.html_closing_special_block
7463
7464 -- parse html blocks
7465 parsers.html_block_opening = parsers.html_incomplete_open_tag
7466                             + parsers.html_incomplete_close_tag
7467
7468 parsers.html_block = parsers.html_block_opening
7469                     * parsers.html_blankline_end_condition
7470
7471 -- parse any html blocks
7472 parsers.html_any_block_opening = (parsers.html_any_open_tag
7473                                   + parsers.html_any_close_tag
7474                                   + parsers.html_any_empty_tag)
7475                                   * #(parsers.optionalspace * (parsers.newline + parser
7476
7477 parsers.html_any_block = parsers.html_any_block_opening
7478                         * parsers.html_blankline_end_condition
7479
7480 parsers.html_inline_comment_full = parsers.html_comment_start
7481                                   * -P(">") * -P("->")
7482                                   * Cs((V("NoSoftLineBreakEndline") + parsers.any - P
7483                                     ")
7484                                     - parsers.nested_breaking_blank - parsers.html_
7485                                   * parsers.html_comment_end
7486
7487 parsers.html_inline_tags = parsers.html_inline_comment_full
7488                           + parsers.html_any_empty_inline_tag

```

```

7488         + parsers.html_inline_instruction
7489         + parsers.html_inline_cdatasection
7490         + parsers.html_inline_declaration
7491         + parsers.html_any_open_inline_tag
7492         + parsers.html_any_close_tag
7493

```

3.1.5.6 Parsers Used for Markdown Tags and Links

```

7494 parsers.urlchar = parsers.anyescaped
7495                 - parsers.newline
7496                 - parsers.more
7497
7498 parsers.auto_link_scheme_part = parsers.alphanumeric
7499                               + parsers.plus
7500                               + parsers.period
7501                               + parsers.dash
7502
7503 parsers.auto_link_scheme = parsers.letter
7504                           * parsers.auto_link_scheme_part
7505                           * parsers.auto_link_scheme_part^-30
7506
7507 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7508                       * (parsers.any - parsers.spacing - parsers.less - parsers.more)
7509
7510 parsers.printable_characters = S(" !#$%&'*/=?^_`{|}~-")
7511
7512 parsers.email_address_local_part_char = parsers.alphanumeric
7513                                       + parsers.printable_characters
7514
7515 parsers.email_address_local_part = parsers.email_address_local_part_char^1
7516
7517 parsers.email_address_dns_label = parsers.alphanumeric
7518                                 * (parsers.alphanumeric + parsers.dash)^-
7519                                 * B(parsers.alphanumeric)
7520
7521 parsers.email_address_domain = parsers.email_address_dns_label
7522                               * (parsers.period * parsers.email_address_dns_label)^0
7523
7524 parsers.email_address = parsers.email_address_local_part
7525                         * parsers.at
7526                         * parsers.email_address_domain
7527
7528 parsers.auto_link_url = parsers.less
7529                       * C(parsers.absolute_uri)
7530                       * parsers.more

```

```

7531
7532 parsers.auto_link_email = parsers.less
7533         * C(parsers.email_address)
7534         * parsers.more
7535
7536 parsers.auto_link_relative_reference = parsers.less
7537         * C(parsers.urlchar^1)
7538         * parsers.more
7539
7540 parsers.autolink = parsers.auto_link_url
7541         + parsers.auto_link_email
7542
7543 -- content in balanced brackets, parentheses, or quotes:
7544 parsers.bracketed = P{ parsers.lbracket
7545         * (( parsers.backslash / "\"" * parsers.rbracket
7546         + parsers.any - (parsers.lbracket
7547         + parsers.rbracket
7548         + parsers.blankline^2)
7549         ) + V(1))^0
7550         * parsers.rbracket }
7551
7552 parsers.inparens = P{ parsers.lparent
7553         * ((parsers.anyescaped - (parsers.lparent
7554         + parsers.rparent
7555         + parsers.blankline^2)
7556         ) + V(1))^0
7557         * parsers.rparent }
7558
7559 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
7560         * ((parsers.anyescaped - (parsers.squote
7561         + parsers.blankline^2)
7562         ) + V(1))^0
7563         * parsers.squote }
7564
7565 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
7566         * ((parsers.anyescaped - (parsers.dquote
7567         + parsers.blankline^2)
7568         ) + V(1))^0
7569         * parsers.dquote }
7570
7571 parsers.link_text = parsers.lbracket
7572         * Cs((parsers.alphanumeric^1
7573         + parsers.bracketed
7574         + parsers.inticks
7575         + parsers.autolink
7576         + V("InlineHtml"))
7577         + ( parsers.backslash * parsers.backslash)

```

```

7578         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7579         + V("NoSoftLineBreakSpace")
7580         + V("NoSoftLineBreakEndline")
7581         + (parsers.any
7582         - (parsers.newline + parsers.lbracket + parsers.rbracket)
7583         * parsers.rbracket
7584
7585 parsers.link_label = parsers.lbracket
7586         * -(parsers.sp * parsers.rbracket)
7587         * #((parsers.any - parsers.rbracket)^-999 * parsers.rbracket)
7588         * Cs((parsers.alphanumeric^1
7589         + parsers.inticks
7590         + parsers.autolink
7591         + V("InlineHtml")
7592         + ( parsers.backslash * parsers.backslash)
7593         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7594         + V("NoSoftLineBreakSpace")
7595         + V("NoSoftLineBreakEndline")
7596         + (parsers.any
7597         - (parsers.newline + parsers.lbracket + parsers.rbracket)
7598         * parsers.rbracket
7599
7600 parsers.inparens_url = P{ parsers.lparent
7601         * ((parsers.anyescaped - (parsers.lparent
7602         + parsers.rparent
7603         + parsers.spacing)
7604         ) + V(1))^0
7605         * parsers.rparent }
7606
7607 -- url for markdown links, allowing nested brackets:
7608 parsers.url = parsers.less * Cs((parsers.anyescaped
7609         - parsers.newline
7610         - parsers.less
7611         - parsers.more)^0)
7612         * parsers.more
7613         + -parsers.less
7614         * Cs((parsers.inparens_url + (parsers.anyescaped
7615         - parsers.spacing
7616         - parsers.lparent
7617         - parsers.rparent))^1)
7618
7619 -- quoted text:
7620 parsers.title_s = parsers.squote
7621         * Cs((parsers.html_entities
7622         + V("NoSoftLineBreakSpace")
7623         + V("NoSoftLineBreakEndline")
7624         + (parsers.anyescaped - parsers.newline - parsers.squote - p

```

```

7625         * parsers.squote
7626
7627 parsers.title_d   = parsers.dquote
7628         * Cs((parsers.html_entities
7629             + V("NoSoftLineBreakSpace")
7630             + V("NoSoftLineBreakEndline")
7631             + (parsers.anyescaped - parsers.newline - parsers.dquote - p
7632         * parsers.dquote
7633
7634 parsers.title_p   = parsers.lparent
7635         * Cs((parsers.html_entities
7636             + V("NoSoftLineBreakSpace")
7637             + V("NoSoftLineBreakEndline")
7638             + (parsers.anyescaped - parsers.newline - parsers.lparent -
7639             - parsers.blankline^2))^0)
7640         * parsers.rparent
7641
7642 parsers.title     = parsers.title_d + parsers.title_s + parsers.title_p
7643
7644 parsers.optionaltitle
7645         = parsers.spnlc * parsers.title * parsers.spacechar^0
7646         + Cc("")
7647

```

3.1.5.7 Helpers for Links and Link Reference Definitions

```

7648 -- parse a reference definition: [foo]: /bar "title"
7649 parsers.define_reference_parser = (parsers.check_trail / "") * parsers.link_label * p
7650         * parsers.spnlc * parsers.url
7651         * ( parsers.spnlc_sep * parsers.title * parsers.only_
7652         + Cc("") * parsers.only_blank)

```

3.1.5.8 Inline Elements

```

7653 parsers.Inline      = V("Inline")
7654
7655 -- parse many p between starter and ender
7656 parsers.between = function(p, starter, ender)
7657     local ender2 = B(parsers.nonspacechar) * ender
7658     return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
7659 end
7660

```

3.1.5.9 Block Elements

```

7661 parsers.lineof = function(c)
7662     return (parsers.check_trail_no_rem * (P(c) * parsers.optionalspace)^3
7663         * (parsers.newline + parsers.eof))

```



```

7664 end
7665
7666 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
7667                               + parsers.lineof(parsers.dash)
7668                               + parsers.lineof(parsers.underscore)

```

3.1.5.10 Headings

```

7669 -- parse Atx heading start and return level
7670 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
7671                       * -parsers.hash / length
7672
7673 -- parse setext header ending and return level
7674 parsers.heading_level = parsers.nonindentspace * parsers.equal^1 * parsers.optionalspace
7675                       + parsers.nonindentspace * parsers.dash^1 * parsers.optionalspace
7676
7677 local function strip_atx_end(s)
7678   return s:gsub("%s+#+%s*\n$", "")
7679 end
7680
7681 parsers.atx_heading = parsers.check_trail_no_rem
7682                    * Cg(parsers.heading_start, "level")
7683                    * (C( parsers.optionalspace
7684                        * parsers.hash^0
7685                        * parsers.optionalspace
7686                        * parsers.newline)
7687                      + parsers.spacechar^1
7688                      * 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 $\text{T}_{\text{E}}\text{X}$ 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 $\langle member \rangle$ s as `reader->` $\langle member \rangle$.

```

7689 M.reader = {}
7690 function M.reader.new(writer, options)
7691   local self = {}

```

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

```
7692 self.writer = writer
7693 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.

```
7694 self.parsers = {}
7695 (function(parsers)
7696     setmetatable(self.parsers, {
7697         __index = function (_, key)
7698             return parsers[key]
7699         end
7700     })
7701 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.

```
7702 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.

```
7703 function self.normalize_tag(tag)
7704     tag = util.ropetostring(tag)
7705     tag = tag:gsub("[ \\n\\r\\t]+", " ")
7706     tag = tag:gsub("^ ", ""):gsub(" $", "")
7707     tag = uni_algos.case.casefold(tag, true, false)
7708     return tag
7709 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.

```
7710 local function iterlines(s, f)
7711     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7712     return util.ropetostring(rope)
7713 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.

```
7714 if options.preserveTabs then
7715     self.expandtabs = function(s) return s end
7716 else
7717     self.expandtabs = function(s)
7718         if s:find("\\t") then
```

```

7719             return iterlines(s, util.expand_tabs_in_line)
7720         else
7721             return s
7722         end
7723     end
7724 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.

```

7725 self.parser_functions = {}
7726 self.create_parser = function(name, grammar, topLevel)
7727     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.

```

7728     if topLevel and options.stripIndent then
7729         local min_prefix_length, min_prefix = nil, ''
7730         str = iterlines(str, function(line)
7731             if lpeg.match(parsers.nonemptyline, line) == nil then
7732                 return line
7733             end
7734             line = util.expand_tabs_in_line(line)
7735             local prefix = lpeg.match(C(parsers.optionalspace), line)
7736             local prefix_length = #prefix
7737             local is_shorter = min_prefix_length == nil
7738             is_shorter = is_shorter or prefix_length < min_prefix_length
7739             if is_shorter then
7740                 min_prefix_length, min_prefix = prefix_length, prefix
7741             end
7742             return line
7743         end)
7744         str = str:gsub('^' .. min_prefix, '')
7745     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.

```

7746     if topLevel and (options.texComments or options.hybrid) then
7747         str = lpeg.match(Ct(parsers.commented_line^1), str)
7748         str = util.rope_to_string(str)

```

```

7749     end
7750     local res = lpeg.match(grammar(), str)
7751     if res == nil then
7752         error(format("%s failed on:\n%s", name, str:sub(1,20)))
7753     else
7754         return res
7755     end
7756 end
7757 end
7758
7759 self.create_parser("parse_blocks",
7760                   function()
7761                       return parsers.blocks
7762                   end, true)
7763
7764 self.create_parser("parse_blocks_nested",
7765                   function()
7766                       return parsers.blocks_nested
7767                   end, false)
7768
7769 self.create_parser("parse_inlines",
7770                   function()
7771                       return parsers.inlines
7772                   end, false)
7773
7774 self.create_parser("parse_inlines_no_inline_note",
7775                   function()
7776                       return parsers.inlines_no_inline_note
7777                   end, false)
7778
7779 self.create_parser("parse_inlines_no_html",
7780                   function()
7781                       return parsers.inlines_no_html
7782                   end, false)
7783
7784 self.create_parser("parse_inlines_nbsp",
7785                   function()
7786                       return parsers.inlines_nbsp
7787                   end, false)
7788 self.create_parser("parse_inlines_no_link_or_emphasis",
7789                   function()
7790                       return parsers.inlines_no_link_or_emphasis
7791                   end, false)

```

3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

7792 parsers.minimally_indented_blankline = parsers.check_minimal_indent * (parsers.blan
7793
7794 parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")
7795
7796 parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("E
7797
7798 parsers.minimally_indented_paragraph = parsers.check_minimal_indent * V("Paragraph"
7799
7800 parsers.minimally_indented_plain = parsers.check_minimal_indent * V("Plain")
7801
7802 parsers.minimally_indented_par_or_plain = parsers.minimally_indented_paragraph
7803         + parsers.minimally_indented_plain
7804
7805 parsers.minimally_indented_par_or_plain_no_blank = parsers.minimally_indented_par_
7806         - parsers.minimally_indented_blan
7807
7808 parsers.minimally_indented_ref = parsers.check_minimal_indent * V("Reference")
7809
7810 parsers.minimally_indented_blank = parsers.check_minimal_indent * V("Blank")
7811
7812 parsers.conditionally_indented_blankline = parsers.check_minimal_blank_indent * (pa
7813
7814 parsers.minimally_indented_ref_or_block = parsers.minimally_indented_ref
7815         + parsers.minimally_indented_block
7816         - parsers.minimally_indented_blankline
7817
7818 parsers.minimally_indented_ref_or_block_or_par = parsers.minimally_indented_ref
7819         + parsers.minimally_indented_block_
7820         - parsers.minimally_indented_blan
7821

```

The following pattern parses the properly indented content that follows the initial container start.

```

7822
7823 parsers.separator_loop = function(separated_block, paragraph, block_separator, para
7824     return separated_block
7825         + block_separator
7826         * paragraph
7827         * separated_block
7828         + paragraph_separator
7829         * paragraph
7830 end
7831
7832 parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
7833     return {
7834         block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
7835         par = parsers.separator_loop(separated_block, paragraph, block_separator, para

```

```

7836     }
7837 end
7838
7839 parsers.block_sep_group = function(blank)
7840     return blank^0 * parsers.eof
7841         + ( blank^2 / writer.paragraphsep
7842             + blank^0 / writer.interblocksep
7843             )
7844 end
7845
7846 parsers.par_sep_group = function(blank)
7847     return blank^0 * parsers.eof
7848         + blank^0 / writer.paragraphsep
7849 end
7850
7851 parsers.sep_group_no_output = function(blank)
7852     return blank^0 * parsers.eof
7853         + blank^0
7854 end
7855
7856 parsers.content_blank = parsers.minimally_indented_blankline
7857
7858 parsers.ref_or_block_separated = parsers.sep_group_no_output(parsers.content_blank
7859     * ( parsers.minimally_indented_ref
7860         - parsers.content_blank)
7861     + parsers.block_sep_group(parsers.content_blank)
7862     * ( parsers.minimally_indented_block
7863         - parsers.content_blank)
7864
7865 parsers.loop_body_pair =
7866     parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7867     parsers.minimally_indented_par_or_plain_no_blank,
7868     parsers.block_sep_group(parsers.content_blank),
7869     parsers.par_sep_group(parsers.content_blank))
7870
7871 parsers.content_loop = ( V("Block")
7872     * parsers.loop_body_pair.block^0
7873     + (V("Paragraph") + V("Plain"))
7874     * parsers.ref_or_block_separated
7875     * parsers.loop_body_pair.block^0
7876     + (V("Paragraph") + V("Plain"))
7877     * parsers.loop_body_pair.par^0)
7878     * parsers.content_blank^0
7879
7880 parsers.indented_content = function()
7881     return Ct( (V("Reference") + (parsers.blankline / ""))
7882         * parsers.content_blank^0

```

```

7883         * parsers.check_minimal_indent
7884         * parsers.content_loop
7885         + (V("Reference") + (parsers.blankline / ""))
7886         * parsers.content_blank^0
7887         + parsers.content_loop)
7888     end
7889
7890     parsers.add_indent = function(pattern, name, breakable)
7891         return Cg(Cmt( Cb("indent_info")
7892             * Ct(pattern)
7893             * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
7894             * Cc(name)
7895             * Cc(breakable),
7896             process_starter_indent), "indent_info")
7897     end
7898

```

3.1.6.4 Parsers Used for Markdown Lists (local)

```

7899     if options.hashEnumerators then
7900         parsers.dig = parsers.digit + parsers.hash
7901     else
7902         parsers.dig = parsers.digit
7903     end
7904
7905     parsers.enumerator = function(delimiter_type, interrupting)
7906         local delimiter_range
7907         local allowed_end
7908         if interrupting then
7909             delimiter_range = P("1")
7910             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7911         else
7912             delimiter_range = parsers.dig * parsers.dig^-8
7913             allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7914         end
7915
7916         return parsers.check_trail
7917             * Ct(C(delimiter_range) * C(delimiter_type))
7918             * allowed_end
7919     end
7920
7921     parsers.starter = parsers.bullet(parsers.dash)
7922         + parsers.bullet(parsers.asterisk)
7923         + parsers.bullet(parsers.plus)
7924         + parsers.enumerator(parsers.period)
7925         + parsers.enumerator(parsers.rparent)
7926

```

3.1.6.5 Parsers Used for Blockquotes (local)

```
7927 parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.space)
7928
7929 parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", true)
7930     * parsers.indented_content()
7931     * remove_indent("bq")
7932
7933 if not options.breakableBlockquotes then
7934     parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", false)
7935     * parsers.indented_content()
7936     * remove_indent("bq")
7937 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.

```
7938 local function parse_content_part(content_part)
7939     local rope = util.rope_to_string(content_part)
7940     local parsed = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
7941     parsed.indent_info = nil
7942     return parsed
7943 end
7944
```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
7945 local function collect_emphasis_content(t, opening_index, closing_index)
7946     local content = {}
7947
7948     local content_part = {}
7949     for i = opening_index, closing_index do
7950         local value = t[i]
7951
7952         if value.rendered ~= nil then
7953             content[#content + 1] = parse_content_part(content_part)
7954             content_part = {}
7955             content[#content + 1] = value.rendered
7956             value.rendered = nil
7957         else
7958             if value.type == "delimiter" and value.element == "emphasis" then
7959                 if value.is_active then
7960                     content_part[#content_part + 1] = string.rep(value.character, value.current)
7961                 end
7962             else
7963                 content_part[#content_part + 1] = value.content
7964             end
7965         end
7966     end
7967 end
```



```

7965     value.content = ''
7966     value.is_active = false
7967   end
7968 end
7969
7970 if next(content_part) ~= nil then
7971   content[#content + 1] = parse_content_part(content_part)
7972 end
7973
7974 return content
7975 end
7976

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```

7977 local function fill_emph(t, opening_index, closing_index)
7978   local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7979   t[opening_index + 1].is_active = true
7980   t[opening_index + 1].rendered = writer.emphasis(content)
7981 end
7982

```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```

7983 local function fill_strong(t, opening_index, closing_index)
7984   local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
7985   t[opening_index + 1].is_active = true
7986   t[opening_index + 1].rendered = writer.strong(content)
7987 end
7988

```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```

7989 local function breaks_three_rule(opening_delimiter, closing_delimiter)
7990   return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
7991     ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0) and
7992     (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count % 3 ~= 0)
7993 end
7994

```

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`.

```

7995 local function find_emphasis_opener(t, bottom_index, latest_index, character, closing_delimiter)
7996   for i = latest_index, bottom_index, -1 do
7997     local value = t[i]
7998     if value.is_active and
7999       value.is_opening and

```

```

8000         value.type == "delimiter" and
8001         value.element == "emphasis" and
8002         (value.character == character) and
8003         (value.current_count > 0) then
8004         if not breaks_three_rule(value, closing_delimiter) then
8005             return i
8006         end
8007     end
8008 end
8009 end
8010

```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

8011 local function process_emphasis(t, opening_index, closing_index)
8012     for i = opening_index, closing_index do
8013         local value = t[i]
8014         if value.type == "delimiter" and value.element == "emphasis" then
8015             local delimiter_length = string.len(value.content)
8016             value.character = string.sub(value.content, 1, 1)
8017             value.current_count = delimiter_length
8018             value.original_count = delimiter_length
8019         end
8020     end
8021
8022     local openers_bottom = {
8023         ['*'] = {
8024             [true] = {opening_index, opening_index, opening_index},
8025             [false] = {opening_index, opening_index, opening_index}
8026         },
8027         ['_'] = {
8028             [true] = {opening_index, opening_index, opening_index},
8029             [false] = {opening_index, opening_index, opening_index}
8030         }
8031     }
8032
8033     local current_position = opening_index
8034     local max_position = closing_index
8035
8036     while current_position <= max_position do
8037         local value = t[current_position]
8038
8039         if value.type ~= "delimiter" or
8040            value.element ~= "emphasis" or
8041            not value.is_active or
8042            not value.is_closing or
8043            (value.current_count <= 0) then

```

```

8044     current_position = current_position + 1
8045     goto continue
8046 end
8047
8048     local character = value.character
8049     local is_opening = value.is_opening
8050     local closing_length_modulo_three = value.original_count % 3
8051
8052     local current_openers_bottom = openers_bottom[character][is_opening][closing_le
8053
8054     local opener_position = find_emphasis_opener(t, current_openers_bottom, current
8055
8056     if (opener_position == nil) then
8057         openers_bottom[character][is_opening][closing_length_modulo_three + 1] = curr
8058         current_position = current_position + 1
8059         goto continue
8060     end
8061
8062     local opening_delimiter = t[opener_position]
8063
8064     local current_opening_count = opening_delimiter.current_count
8065     local current_closing_count = t[current_position].current_count
8066
8067     if (current_opening_count >= 2) and (current_closing_count >= 2) then
8068         opening_delimiter.current_count = current_opening_count - 2
8069         t[current_position].current_count = current_closing_count - 2
8070         fill_strong(t, opener_position, current_position)
8071     else
8072         opening_delimiter.current_count = current_opening_count - 1
8073         t[current_position].current_count = current_closing_count - 1
8074         fill_emph(t, opener_position, current_position)
8075     end
8076
8077     ::continue::
8078 end
8079 end
8080
8081 local cont = lpeg.R("\128\191") -- continuation byte
8082

```

Match a UTF-8 character of byte length *n*.

```

8083 local function utf8_by_byte_count(n)
8084     if (n == 1) then
8085         return lpeg.R("\0\127")
8086     end
8087     if (n == 2) then
8088         return lpeg.R("\194\223") * cont
8089     end

```

```

8090     if (n == 3) then
8091         return lpeg.R("\224\239") * cont * cont
8092     end
8093     if (n == 4) then
8094         return lpeg.R("\240\244") * cont * cont * cont
8095     end
8096 end

```

Check if there is a character of a type `chartype` between the start position `start_pos` and end position `end_pos` in a string `s` relative to current index `i`.

```

8097 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
8098     local c
8099     local char_length
8100     for pos = start_pos, end_pos, 1 do
8101         if (start_pos < 0) then
8102             char_length = -pos
8103         else
8104             char_length = pos + 1
8105         end
8106
8107         if (chartype == "punctuation") then
8108             if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8109                 return i
8110             end
8111         else
8112             c = lpeg.match({ C(utf8_by_byte_count(char_length)) }, s, i+pos)
8113             if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8114                 return i
8115             end
8116         end
8117     end
8118 end
8119
8120 local function check_preceding_unicode_punctuation(s, i)
8121     return check_unicode_type(s, i, -4, -1, "punctuation")
8122 end
8123
8124 local function check_preceding_unicode_whitespace(s, i)
8125     return check_unicode_type(s, i, -4, -1, "%s")
8126 end
8127
8128 local function check_following_unicode_punctuation(s, i)
8129     return check_unicode_type(s, i, 0, 3, "punctuation")
8130 end
8131
8132 local function check_following_unicode_whitespace(s, i)
8133     return check_unicode_type(s, i, 0, 3, "%s")

```

```

8134 end
8135
8136 parsers.unicode_preceding_punctuation = B(parsers.escapable)
8137         + Cmt(parsers.succeed, check_preceding_uniCOD
8138
8139 parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_uniCOD
8140
8141 parsers.unicode_following_punctuation = #parsers.escapable
8142         + Cmt(parsers.succeed, check_following_uniCOD
8143
8144 parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_uniCOD
8145
8146 parsers.delimiter_run = function(character)
8147     return (B(parsers.backslash * character) + -B(character))
8148             * character^1
8149             * -#character
8150 end
8151
8152 parsers.left_flanking_delimiter_run = function(character)
8153     return (B( parsers.any)
8154             * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_wh
8155             + -B(parsers.any))
8156             * parsers.delimiter_run(character)
8157             * parsers.unicode_following_punctuation
8158             + parsers.delimiter_run(character)
8159             * -(parsers.unicode_following_punctuation + parsers.unicode_following_wh
8160             + parsers.eof)
8161 end
8162
8163 parsers.right_flanking_delimiter_run = function(character)
8164     return parsers.unicode_preceding_punctuation
8165             * parsers.delimiter_run(character)
8166             * (parsers.unicode_following_punctuation + parsers.unicode_following_whites
8167             + parsers.eof)
8168             + (B(parsers.any)
8169             * -(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whi
8170             * parsers.delimiter_run(character)
8171 end
8172
8173 if options.underscores then
8174     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8175                         + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
8176                         + (parsers.unicode_preceding_punctuation
8177                         * #parsers.right_flanking_delimiter_run(parsers.underscor
8178                         * parsers.left_flanking_delimiter_run(parsers.underscore)
8179
8180     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)

```

```

8181             + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
8182             + #(parsers.left_flanking_delimiter_run(parsers.underscore)
8183             * parsers.unicode_following_punctuation))
8184             * parsers.right_flanking_delimiter_run(parsers.underscore)
8185     else
8186         parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8187
8188         parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8189     end
8190
8191     parsers.emph_capturing_open_and_close = #parsers.emph_start * #parsers.emph_end
8192             * Ct( Cg(Cc("delimiter"), "type")
8193             * Cg(Cc("emphasis"), "element")
8194             * Cg(C(parsers.emph_start), "content")
8195             * Cg(Cc(true), "is_opening")
8196             * Cg(Cc(true), "is_closing"))
8197
8198     parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
8199             * Cg(Cc("emphasis"), "element")
8200             * Cg(C(parsers.emph_start), "content")
8201             * Cg(Cc(true), "is_opening")
8202             * Cg(Cc(false), "is_closing"))
8203
8204     parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
8205             * Cg(Cc("emphasis"), "element")
8206             * Cg(C(parsers.emph_end), "content")
8207             * Cg(Cc(false), "is_opening")
8208             * Cg(Cc(true), "is_closing"))
8209
8210     parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8211             + parsers.emph_capturing_open
8212             + parsers.emph_capturing_close
8213
8214     parsers.emph_open = parsers.emph_capturing_open_and_close
8215             + parsers.emph_capturing_open
8216
8217     parsers.emph_close = parsers.emph_capturing_open_and_close
8218             + parsers.emph_capturing_close
8219

```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

8220 -- List of references defined in the document
8221 local references
8222

```

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.

```
8223 function self.register_link(_, tag, url, title,
8224                             attributes)
8225     local normalized_tag = self.normalize_tag(tag)
8226     if references[normalized_tag] == nil then
8227         references[normalized_tag] = {
8228             url = url,
8229             title = title,
8230             attributes = attributes
8231         }
8232     end
8233     return ""
8234 end
8235
```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```
8236 function self.lookup_reference(tag)
8237     return references[self.normalize_tag(tag)]
8238 end
8239
8240 parsers.title_s_direct_ref = parsers.squote
8241                             * Cs((parsers.html_entities
8242                                 + (parsers.anyescaped - parsers.squote - parsers.bl
8243                                 * parsers.squote
8244
8245 parsers.title_d_direct_ref = parsers.dquote
8246                             * Cs((parsers.html_entities
8247                                 + (parsers.anyescaped - parsers.dquote - parsers.bl
8248                                 * parsers.dquote
8249
8250 parsers.title_p_direct_ref = parsers.lparent
8251                             * Cs((parsers.html_entities
8252                                 + (parsers.anyescaped - parsers.lparent - parsers.r
8253                                 * parsers.rparent
8254
8255 parsers.title_direct_ref = parsers.title_s_direct_ref
8256                          + parsers.title_d_direct_ref
8257                          + parsers.title_p_direct_ref
8258
8259 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8260                                  * Cg(parsers.url + Cc(""), "url")
8261                                  * parsers.spnl
8262                                  * Cg(parsers.title_direct_ref + Cc(""), "title")
8263                                  * parsers.spnl * parsers.rparent
8264
8265 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
```

```

8266             * Cg(parsers.url + Cc(""), "url")
8267             * parsers.spnlc
8268             * Cg(parsers.title + Cc(""), "title")
8269             * parsers.spnlc * parsers.rparent
8270
8271 parsers.empty_link = parsers.lbracket
8272                 * parsers.rbracket
8273
8274 parsers.inline_link = parsers.link_text
8275                 * parsers.inline_direct_ref
8276
8277 parsers.full_link = parsers.link_text
8278                 * parsers.link_label
8279
8280 parsers.shortcut_link = parsers.link_label
8281                 * -(parsers.empty_link + parsers.link_label)
8282
8283 parsers.collapsed_link = parsers.link_label
8284                 * parsers.empty_link
8285
8286 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8287                 * Cg(Cc("inline"), "link_type")
8288                 + #(parsers.exclamation * parsers.full_link)
8289                 * Cg(Cc("full"), "link_type")
8290                 + #(parsers.exclamation * parsers.collapsed_link)
8291                 * Cg(Cc("collapsed"), "link_type")
8292                 + #(parsers.exclamation * parsers.shortcut_link)
8293                 * Cg(Cc("shortcut"), "link_type")
8294                 + #(parsers.exclamation * parsers.empty_link)
8295                 * Cg(Cc("empty"), "link_type")
8296
8297 parsers.link_opening = #parsers.inline_link
8298                 * Cg(Cc("inline"), "link_type")
8299                 + #parsers.full_link
8300                 * Cg(Cc("full"), "link_type")
8301                 + #parsers.collapsed_link
8302                 * Cg(Cc("collapsed"), "link_type")
8303                 + #parsers.shortcut_link
8304                 * Cg(Cc("shortcut"), "link_type")
8305                 + #parsers.empty_link
8306                 * Cg(Cc("empty_link"), "link_type")
8307                 + #parsers.link_text
8308                 * Cg(Cc("link_text"), "link_type")
8309
8310 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
8311                 * Cg(Cc(true), "is_opening")
8312                 * Cg(Cc(false), "is_closing")

```



```

8313             * ( Cg(Cc("image"), "element")
8314             * parsers.image_opening
8315             * Cg(parsers.exclamation * parsers.lbracket, "con
8316             + Cg(Cc("link"), "element")
8317             * parsers.link_opening
8318             * Cg(parsers.lbracket, "content")))
8319
8320 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
8321             * Cg(Cc("link"), "element")
8322             * Cg(Cc(false), "is_opening")
8323             * Cg(Cc(true), "is_closing")
8324             * ( Cg(Cc(true), "is_direct")
8325             * Cg(parsers.rbracket * #parsers.inline_direct_re
8326             + Cg(Cc(false), "is_direct")
8327             * Cg(parsers.rbracket, "content")))
8328
8329 parsers.link_image_open_or_close = parsers.link_image_opening
8330             + parsers.link_image_closing
8331
8332 if options.html then
8333     parsers.link_emph_precedence = parsers.inticks
8334             + parsers.autolink
8335             + parsers.html_inline_tags
8336 else
8337     parsers.link_emph_precedence = parsers.inticks
8338             + parsers.autolink
8339 end
8340
8341 parsers.link_and_emph_endline = parsers.newline
8342             * ((parsers.check_minimal_indent
8343             * -V("EndlineExceptions")
8344             + parsers.check_optional_indent
8345             * -V("EndlineExceptions")
8346             * -parsers.starter) / "")
8347             * parsers.spacechar^0 / "\n"
8348
8349 parsers.link_and_emph_content = Ct( Cg(Cc("content"), "type")
8350             * Cg(Cs(( parsers.link_emph_precedence
8351             + parsers.backslash * parsers.any
8352             + parsers.link_and_emph_endline
8353             + (parsers.linechar
8354             - parsers.blankline^2
8355             - parsers.link_image_open_or_close
8356             - parsers.emph_open_or_close))^0), "con
8357
8358 parsers.link_and_emph_table = (parsers.link_image_opening + parsers.emph_open)
8359             * parsers.link_and_emph_content

```

```

8360             * ((parsers.link_image_open_or_close + parsers.emph_ope
8361             * parsers.link_and_emph_content)^1
8362

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

8363 local function collect_link_content(t, opening_index, closing_index)
8364     local content = {}
8365     for i = opening_index, closing_index do
8366         content[#content + 1] = t[i].content
8367     end
8368     return util.rope_to_string(content)
8369 end
8370

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

8371 local function find_link_opener(t, bottom_index, latest_index)
8372     for i = latest_index, bottom_index, -1 do
8373         local value = t[i]
8374         if value.type == "delimiter" and
8375            value.is_opening and
8376            (value.element == "link" or value.element == "image")
8377            and not value.removed then
8378             if value.is_active then
8379                 return i
8380             end
8381             value.removed = true
8382             return nil
8383         end
8384     end
8385 end
8386

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

8387 local function find_next_link_closing_index(t, latest_index)
8388     for i = latest_index, #t do
8389         local value = t[i]
8390         if value.is_closing and
8391            value.element == "link" and
8392            not value.removed then
8393             return i
8394         end
8395     end
8396 end
8397

```

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.

```
8398 local function disable_previous_link_openers(t, opening_index)
8399   if t[opening_index].element == "image" then
8400     return
8401   end
8402
8403   for i = opening_index, 1, -1 do
8404     local value = t[i]
8405     if value.is_active and
8406        value.type == "delimiter" and
8407        value.is_opening and
8408        value.element == "link" then
8409       value.is_active = false
8410     end
8411   end
8412 end
8413
```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```
8414 local function disable_range(t, opening_index, closing_index)
8415   for i = opening_index, closing_index do
8416     local value = t[i]
8417     if value.is_active then
8418       value.is_active = false
8419       if value.type == "delimiter" then
8420         value.removed = true
8421       end
8422     end
8423   end
8424 end
8425
```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8426 local function delete_parsed_content_in_range(t, opening_index, closing_index)
8427   for i = opening_index, closing_index do
8428     t[i].rendered = nil
8429   end
8430 end
8431
```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8432 local function empty_content_in_range(t, opening_index, closing_index)
8433   for i = opening_index, closing_index do
8434     t[i].content = ''

```

```

8435     end
8436   end
8437

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

8438   local function join_attributes(reference_attributes, own_attributes)
8439     local merged_attributes = {}
8440     for _, attribute in ipairs(reference_attributes or {}) do
8441       table.insert(merged_attributes, attribute)
8442     end
8443     for _, attribute in ipairs(own_attributes or {}) do
8444       table.insert(merged_attributes, attribute)
8445     end
8446     if next(merged_attributes) == nil then
8447       merged_attributes = nil
8448     end
8449     return merged_attributes
8450   end
8451

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

8452   local function render_link_or_image(t, opening_index, closing_index, content_end_index)
8453     process_emphasis(t, opening_index, content_end_index)
8454     local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index - 1)
8455
8456     local rendered = {}
8457     if (t[opening_index].element == "link") then
8458       rendered = writer.link(mapped, reference.url, reference.title, reference.attributes)
8459     end
8460
8461     if (t[opening_index].element == "image") then
8462       rendered = writer.image(mapped, reference.url, reference.title, reference.attributes)
8463     end
8464
8465     t[opening_index].rendered = rendered
8466     delete_parsed_content_in_range(t, opening_index + 1, closing_index)
8467     empty_content_in_range(t, opening_index, closing_index)
8468     disable_previous_link_openers(t, opening_index)
8469     disable_range(t, opening_index, closing_index)
8470   end
8471

```

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.

```

8472   local function resolve_inline_following_content(t, closing_index, match_reference,

```

```

8473     local content = ""
8474     for i = closing_index + 1, #t do
8475         content = content .. t[i].content
8476     end
8477
8478     local matching_content = parsers.succeed
8479
8480     if match_reference then
8481         matching_content = matching_content * parsers.inline_direct_ref_inside
8482     end
8483
8484     if match_link_attributes then
8485         matching_content = matching_content * Cg(Ct(parsers.attributes^-
251), "attributes")
8486     end
8487
8488     local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), content)
8489
8490     local matched_count = matched.end_position - 1
8491     for i = closing_index + 1, #t do
8492         local value = t[i]
8493
8494         local chars_left = matched_count
8495         matched_count = matched_count - #value.content
8496
8497         if matched_count <= 0 then
8498             value.content = value.content:sub(chars_left + 1)
8499             break
8500         end
8501
8502         value.content = ''
8503         value.is_active = false
8504     end
8505
8506     local attributes = matched.attributes
8507     if attributes == nil or next(attributes) == nil then
8508         attributes = nil
8509     end
8510
8511     return {
8512         url = matched.url or "",
8513         title = matched.title or "",
8514         attributes = attributes
8515     }
8516 end
8517

```

Resolve an inline link [a](#)³³ 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.

```
8518 local function resolve_inline_link(t, opening_index, closing_index)
8519     local inline_content = resolve_inline_following_content(t, closing_index, true, t
8520     render_link_or_image(t, opening_index, closing_index, closing_index, inline_content)
8521 end
8522
```

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.

```
8523 local function resolve_shortcut_link(t, opening_index, closing_index)
8524     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8525     local r = self.lookup_reference(content)
8526
8527     if r then
8528         local inline_content = resolve_inline_following_content(t, closing_index, false, t
8529         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8530         render_link_or_image(t, opening_index, closing_index, closing_index, r)
8531     end
8532 end
8533
```

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.

```
8534 local function resolve_full_link(t, opening_index, closing_index)
8535     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4
8536     local next_link_content = collect_link_content(t, closing_index + 3, next_link_closing_index)
8537     local r = self.lookup_reference(next_link_content)
8538
8539     if r then
8540         local inline_content = resolve_inline_following_content(t, next_link_closing_index,
8541         t.match_link_attributes
8542         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8543         render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8544     end
8545 end
8546
```

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.

```
8547 local function resolve_collapsed_link(t, opening_index, closing_index)
8548     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4
8549     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8550     local r = self.lookup_reference(content)
```

³³See [b](#).

```

8551
8552     if r then
8553         local inline_content = resolve_inline_following_content(t, closing_index, false)
8554         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8555         render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8556     end
8557 end
8558

```

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.

```

8559 local function process_links_and_emphasis(t)
8560     for _,value in ipairs(t) do
8561         value.is_active = true
8562     end
8563
8564     for i,value in ipairs(t) do
8565         if not value.is_closing or
8566            value.type ~= "delimiter" or
8567            not (value.element == "link" or value.element == "image") then
8568             goto continue
8569         end
8570
8571         local opener_position = find_link_opener(t, 1, i - 1)
8572         if (opener_position == nil) then
8573             goto continue
8574         end
8575
8576         local opening_delimiter = t[opener_position]
8577         opening_delimiter.removed = true
8578
8579         local link_type = opening_delimiter.link_type
8580
8581         if (link_type == "inline") then
8582             resolve_inline_link(t, opener_position, i)
8583         end
8584         if (link_type == "shortcut") then
8585             resolve_shortcut_link(t, opener_position, i)
8586         end
8587         if (link_type == "full") then
8588             resolve_full_link(t, opener_position, i)
8589         end
8590         if (link_type == "collapsed") then
8591             resolve_collapsed_link(t, opener_position, i)
8592         end

```

```

8593
8594     ::continue::
8595 end
8596
8597 t[#t].content = t[#t].content.gsub("%s*$", "")
8598
8599 process_emphasis(t, 1, #t)
8600 local final_result = collect_emphasis_content(t, 1, #t)
8601 return final_result
8602 end
8603
8604 function self.defer_link_and_emphasis_processing(delimiter_table)
8605     return writer.defer_call(function()
8606         return process_links_and_emphasis(delimiter_table)
8607     end)
8608 end
8609

```

3.1.6.8 Inline Elements (local)

```

8610 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
8611                  / writer.string
8612
8613 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
8614                  / writer.string
8615
8616 parsers.Ellipsis = P("...") / writer.ellipsis
8617
8618 parsers.Smart    = parsers.Ellipsis
8619
8620 parsers.Code     = parsers.inticks / writer.code
8621
8622 if options.blankBeforeBlockquote then
8623     parsers.bqstart = parsers.fail
8624 else
8625     parsers.bqstart = parsers.blockquote_start
8626 end
8627
8628 if options.blankBeforeHeading then
8629     parsers.headerstart = parsers.fail
8630 else
8631     parsers.headerstart = parsers.atx_heading
8632 end
8633
8634 if options.blankBeforeList then
8635     parsers.interrupting_bullets = parsers.fail
8636     parsers.interrupting_enumerators = parsers.fail

```



```

8637 else
8638     parsers.interrupting_bullets = parsers.bullet(parsers.dash, true)
8639                                 + parsers.bullet(parsers.asterisk, true)
8640                                 + parsers.bullet(parsers.plus, true)
8641
8642     parsers.interrupting_enumerators = parsers.enumerator(parsers.period, true)
8643                                     + parsers.enumerator(parsers.rparent, true)
8644 end
8645
8646 if options.html then
8647     parsers.html_interrupting = parsers.check_trail
8648                               * ( parsers.html_incomplete_open_tag
8649                                   + parsers.html_incomplete_close_tag
8650                                   + parsers.html_incomplete_open_special_tag
8651                                   + parsers.html_comment_start
8652                                   + parsers.html_cdatasection_start
8653                                   + parsers.html_declaration_start
8654                                   + parsers.html_instruction_start
8655                                   - parsers.html_close_special_tag
8656                                   - parsers.html_empty_special_tag)
8657 else
8658     parsers.html_interrupting = parsers.fail
8659 end
8660
8661 parsers.EndlineExceptions
8662     = parsers.blankline -- paragraph break
8663     + parsers.eof      -- end of document
8664     + parsers.bqstart
8665     + parsers.thematic_break_lines
8666     + parsers.interrupting_bullets
8667     + parsers.interrupting_enumerators
8668     + parsers.headerstart
8669     + parsers.html_interrupting
8670
8671 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8672
8673 parsers.endline = parsers.newline
8674                 * (parsers.check_minimal_indent
8675                     * -V("EndlineExceptions")
8676                     + parsers.check_optional_indent
8677                     * -V("EndlineExceptions")
8678                     * -parsers.starter)
8679                 * parsers.spacechar^0
8680
8681 parsers.Endline = parsers.endline
8682                 / writer.soft_line_break
8683

```

```

8684 parsers.EndlineNoSub = parsers.endline
8685
8686 parsers.NoSoftLineBreakEndline
8687     = parsers.newline
8688     * (parsers.check_minimal_indent
8689     * -V("NoSoftLineBreakEndlineExceptions")
8690     + parsers.check_optional_indent
8691     * -V("NoSoftLineBreakEndlineExceptions")
8692     * -parsers.starter)
8693     * parsers.spacechar^0
8694     / writer.space
8695
8696 parsers.EndlineBreak = parsers.backslash * parsers.Endline
8697                       / writer.hard_line_break
8698
8699 parsers.OptionalIndent
8700     = parsers.spacechar^1 / writer.space
8701
8702 parsers.Space         = parsers.spacechar^2 * parsers.Endline
8703                       / writer.hard_line_break
8704     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8705     + parsers.spacechar^1 * parsers.Endline
8706                       / writer.soft_line_break
8707     + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8708
8709 parsers.NoSoftLineBreakSpace
8710     = parsers.spacechar^2 * parsers.Endline
8711                       / writer.hard_line_break
8712     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8713     + parsers.spacechar^1 * parsers.Endline
8714                       / writer.soft_line_break
8715     + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8716
8717 parsers.NonbreakingEndline
8718     = parsers.endline
8719     / writer.soft_line_break
8720
8721 parsers.NonbreakingSpace
8722     = parsers.spacechar^2 * parsers.Endline
8723                       / writer.hard_line_break
8724     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
8725     + parsers.spacechar^1 * parsers.Endline
8726     * parsers.optionalspace
8727     / writer.soft_line_break
8728     + parsers.spacechar^1 * parsers.optionalspace
8729     / writer.nbsp
8730

```

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.

```
8731 function self.auto_link_url(url, attributes)
8732   return writer.link(writer.escape(url),
8733                     url, nil, attributes)
8734 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.

```
8735 function self.auto_link_email(email, attributes)
8736   return writer.link(writer.escape(email),
8737                     "mailto:".email,
8738                     nil, attributes)
8739 end
8740
8741 parsers.AutoLinkUrl = parsers.auto_link_url
8742                       / self.auto_link_url
8743
8744 parsers.AutoLinkEmail
8745                       = parsers.auto_link_email
8746                       / self.auto_link_email
8747
8748 parsers.AutoLinkRelativeReference
8749                       = parsers.auto_link_relative_reference
8750                       / self.auto_link_url
8751
8752 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
8753                       / self.defer_link_and_emphasis_processing
8754
8755 parsers.EscapedChar  = parsers.backslash * C(parsers.escapable) / writer.string
8756
8757 parsers.InlineHtml   = Cs(parsers.html_inline_comment) / writer.inline_html_comment
8758                       + Cs(parsers.html_any_empty_inline_tag
8759                           + parsers.html_inline_instruction
8760                           + parsers.html_inline_cdatasection
8761                           + parsers.html_inline_declaration
8762                           + parsers.html_any_open_inline_tag
8763                           + parsers.html_any_close_tag)
8764                       / writer.inline_html_tag
8765
8766 parsers.HtmlEntity    = parsers.html_entities / writer.string
```

3.1.6.9 Block Elements (local)

```
8767 parsers.DisplayHtml = Cs(parsers.check_trail
```

```

8768         * ( parsers.html_comment
8769           + parsers.html_special_block
8770           + parsers.html_block
8771           + parsers.html_any_block
8772           + parsers.html_instruction
8773           + parsers.html_cdatasection
8774           + parsers.html_declaration))
8775         / writer.block_html_element
8776
8777 parsers.indented_non_blank_line = parsers.indentedline - parsers.blankline
8778
8779 parsers.Verbatim = Cs(
8780     parsers.check_code_trail
8781     * (parsers.line - parsers.blankline)
8782     * ((parsers.check_minimal_blank_indent_and_full_code_trail * pa
8783       * ((parsers.check_minimal_indent / "") * parsers.check_code_t
8784         * (parsers.line - parsers.blankline))^1)^0
8785     ) / self.expandtabs / writer.verbatim
8786
8787 parsers.Blockquote = parsers.blockquote_body
8788                   / writer.blockquote
8789
8790 parsers.ThematicBreak = parsers.thematic_break_lines
8791                       / writer.thematic_break
8792
8793 parsers.Reference = parsers.define_reference_parser
8794                   / self.register_link
8795
8796 parsers.Paragraph = parsers.freeze_trail
8797                   * (Ct((parsers.Inline)^1)
8798                   * (parsers.newline + parsers.eof)
8799                   * parsers.unfreeze_trail
8800                   / writer.paragraph)
8801
8802 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
8803                   / writer.plain

```

3.1.6.10 Lists (local)

```

8804
8805 if options.taskLists then
8806     parsers.tickbox = ( parsers.ticked_box
8807                       + parsers.halfticked_box
8808                       + parsers.unticked_box
8809                       ) / writer.tickbox
8810 else
8811     parsers.tickbox = parsers.fail

```

```

8812 end
8813
8814 parsers.list_blank = parsers.conditionally_indented_blankline
8815
8816 parsers.ref_or_block_list_separated = parsers.sep_group_no_output(parsers.list_blank
8817     * parsers.minimally_indented_ref
8818     + parsers.block_sep_group(parsers.list_blank)
8819     * parsers.minimally_indented_block
8820
8821 parsers.ref_or_block_non_separated = parsers.minimally_indented_ref
8822     + (parsers.succeed / writer.interblocksep)
8823     * parsers.minimally_indented_block
8824     - parsers.minimally_indented_blankline
8825
8826 parsers.tight_list_loop_body_pair =
8827     parsers.create_loop_body_pair(parsers.ref_or_block_non_separated,
8828     parsers.minimally_indented_par_or_plain_no_blank,
8829     (parsers.succeed / writer.interblocksep),
8830     (parsers.succeed / writer.paragraphsep))
8831
8832 parsers.loose_list_loop_body_pair =
8833     parsers.create_loop_body_pair(parsers.ref_or_block_list_separated,
8834     parsers.minimally_indented_par_or_plain,
8835     parsers.block_sep_group(parsers.list_blank),
8836     parsers.par_sep_group(parsers.list_blank))
8837
8838 parsers.tight_list_content_loop = V("Block")
8839     * parsers.tight_list_loop_body_pair.block^0
8840     + (V("Paragraph") + V("Plain"))
8841     * parsers.ref_or_block_non_separated
8842     * parsers.tight_list_loop_body_pair.block^0
8843     + (V("Paragraph") + V("Plain"))
8844     * parsers.tight_list_loop_body_pair.par^0
8845
8846 parsers.loose_list_content_loop = V("Block")
8847     * parsers.loose_list_loop_body_pair.block^0
8848     + (V("Paragraph") + V("Plain"))
8849     * parsers.ref_or_block_list_separated
8850     * parsers.loose_list_loop_body_pair.block^0
8851     + (V("Paragraph") + V("Plain"))
8852     * parsers.loose_list_loop_body_pair.par^0
8853
8854 parsers.list_item_tightness_condition = -( parsers.list_blank^0
8855     * parsers.minimally_indented_ref_or_block
8856     * remove_indent("li")
8857     + remove_indent("li")
8858     * parsers.fail

```

```

8859
8860 parsers.indented_content_tight = Ct( (parsers.blankline / "")
8861     * #parsers.list_blank
8862     * remove_indent("li")
8863     + ( (V("Reference") + (parsers.blankline / ""))
8864     * parsers.check_minimal_indent
8865     * parsers.tight_list_content_loop
8866     + (V("Reference") + (parsers.blankline / ""))
8867     + (parsers.tickbox^-1 / writer.escape)
8868     * parsers.tight_list_content_loop
8869     )
8870     * parsers.list_item_tightness_condition
8871     )
8872
8873 parsers.indented_content_loose = Ct( (parsers.blankline / "")
8874     * #parsers.list_blank
8875     + ( (V("Reference") + (parsers.blankline / ""))
8876     * parsers.check_minimal_indent
8877     * parsers.loose_list_content_loop
8878     + (V("Reference") + (parsers.blankline / ""))
8879     + (parsers.tickbox^-1 / writer.escape)
8880     * parsers.loose_list_content_loop
8881     )
8882     )
8883
8884 parsers.TightListItem = function(starter)
8885     return -parsers.ThematicBreak
8886         * parsers.add_indent(starter, "li")
8887         * parsers.indented_content_tight
8888     end
8889
8890 parsers.LooseListItem = function(starter)
8891     return -parsers.ThematicBreak
8892         * parsers.add_indent(starter, "li")
8893         * parsers.indented_content_loose
8894         * remove_indent("li")
8895     end
8896
8897 parsers.BulletListOfType = function(bullet_type)
8898     local bullet = parsers.bullet(bullet_type)
8899     return ( Ct( parsers.TightListItem(bullet)
8900         * ( (parsers.check_minimal_indent / "")
8901         * parsers.TightListItem(bullet)
8902         )^0
8903     )
8904     * Cc(true)
8905     * -#( (parsers.list_blank^0 / "")

```

```

8906         * parsers.check_minimal_indent
8907         * (bullet - parsers.ThematicBreak)
8908     )
8909     + Ct( parsers.LooseListItem(bullet)
8910         * ( (parsers.list_blank^0 / "")
8911             * (parsers.check_minimal_indent / "")
8912             * parsers.LooseListItem(bullet)
8913             )^0
8914     )
8915     * Cc(false)
8916 ) / writer.bulletlist
8917 end
8918
8919 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
8920                   + parsers.BulletListOfType(parsers.asterisk)
8921                   + parsers.BulletListOfType(parsers.plus)
8922
8923 local function ordered_list(items,tight,starter)
8924     local startnum = starter[2][1]
8925     if options.startNumber then
8926         startnum = tonumber(startnum) or 1 -- fallback for '#'
8927         if startnum ~= nil then
8928             startnum = math.floor(startnum)
8929         end
8930     else
8931         startnum = nil
8932     end
8933     return writer.orderedlist(items,tight,startnum)
8934 end
8935
8936 parsers.OrderedListOfTypes = function(delimiter_type)
8937     local enumerator = parsers.enumerator(delimiter_type)
8938     return Cg(enumerator, "listtype")
8939         * (Ct( parsers.TightListItem(Cb("listtype"))
8940             * ((parsers.check_minimal_indent / "") * parsers.TightListItem(enumerat
8941             * Cc(true)
8942             * -#((parsers.list_blank^0 / "")
8943                 * parsers.check_minimal_indent * enumerator)
8944         + Ct( parsers.LooseListItem(Cb("listtype"))
8945             * ((parsers.list_blank^0 / "")
8946                 * (parsers.check_minimal_indent / "") * parsers.LooseListItem(enumerat
8947             * Cc(false)
8948         ) * Ct(Cb("listtype"))) / ordered_list
8949 end
8950
8951 parsers.OrderedList = parsers.OrderedListOfTypes(parsers.period)
8952                   + parsers.OrderedListOfTypes(parsers.rparent)

```

3.1.6.11 Blank (local)

```
8953 parsers.Blank = parsers.blankline / ""
8954 + V("Reference")
```

3.1.6.12 Headings (local)

```
8955 function parsers.parse_heading_text(s)
8956   local inlines = self.parser_functions.parse_inlines(s)
8957   local flatten_inlines = self.writer.flatten_inlines
8958   self.writer.flatten_inlines = true
8959   local flat_text = self.parser_functions.parse_inlines(s)
8960   flat_text = util.ropetostring(flat_text)
8961   self.writer.flatten_inlines = flatten_inlines
8962   return {flat_text, inlines}
8963 end
8964
8965 -- parse atx header
8966 parsers.AtxHeading = parsers.check_trail_no_rem
8967 * Cg(parsers.heading_start, "level")
8968 * ((C( parsers.optionalspace
8969   * parsers.hash^0
8970   * parsers.optionalspace
8971   * parsers.newline)
8972 + parsers.spacechar^1
8973 * C(parsers.line))
8974 / strip_atx_end
8975 / parsers.parse_heading_text)
8976 * Cb("level")
8977 / writer.heading
8978
8979 parsers.heading_line = parsers.linechar^1
8980 - parsers.thematic_break_lines
8981
8982 parsers.heading_text = parsers.heading_line
8983 * ((V("Endline") / "\n") * (parsers.heading_line - parsers.heading_line)
8984 * parsers.newline^-1)
8985
8986 parsers.SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
8987 * #(parsers.heading_text
8988   * parsers.check_minimal_indent * parsers.check_trail * parsers.heading_line)
8989 * Cs(parsers.heading_text)
8990 / parsers.parse_heading_text
8991 * parsers.check_minimal_indent_and_trail * parsers.heading_line
8992 * parsers.newline
8993 * parsers.unfreeze_trail
8994 / writer.heading
8995
```



```
8996 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_EX output.

```
8997 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.

```
8998   local walkable_syntax = (function(global_walkable_syntax)
8999     local local_walkable_syntax = {}
9000     for lhs, rule in pairs(global_walkable_syntax) do
9001       local_walkable_syntax[lhs] = util.table_copy(rule)
9002     end
9003     return local_walkable_syntax
9004   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.

```
9005   local current_extension_name = nil
9006   self.insert_pattern = function(selector, pattern, pattern_name)
9007     assert(pattern_name == nil or type(pattern_name) == "string")
9008     local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9009     assert(lhs ~= nil,
9010       [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
9011       .. selector .. ["])
9012     assert(walkable_syntax[lhs] ~= nil,
9013       [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
9014     assert(pos == "before" or pos == "after" or pos == "instead of",
9015       [[Expected positional specifier "before", "after", or "instead of", not "]]
9016       .. pos .. ["])
9017     local rule = walkable_syntax[lhs]
9018     local index = nil
9019     for current_index, current_rhs in ipairs(rule) do
9020       if type(current_rhs) == "string" and current_rhs == rhs then
9021         index = current_index
9022         if pos == "after" then
9023           index = index + 1
9024         end
9025         break
9026       end
9026     end
```

```

9027     end
9028     assert(index ~= nil,
9029         [[Rule ]] .. lhs .. [[ -> ]] .. rhs
9030         .. [[ does not exist in markdown grammar]])
9031     local accountable_pattern
9032     if current_extension_name then
9033         accountable_pattern = { pattern, current_extension_name, pattern_name }
9034     else
9035         assert(type(pattern) == "string",
9036             [[reader->insert_pattern() was called outside an extension with ]]
9037             .. [[a PEG pattern instead of a rule name]])
9038         accountable_pattern = pattern
9039     end
9040     if pos == "instead of" then
9041         rule[index] = accountable_pattern
9042     else
9043         table.insert(rule, index, accountable_pattern)
9044     end
9045 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.

```

9046     local syntax =
9047         { "Blocks",
9048
9049           Blocks = V("InitializeState")
9050                 * ( V("ExpectedJekyllData")
9051                   * (V("Blank")^0 / writer.interblocksep)
9052                   )^-1
9053                 * 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.

```

9054                 * ( V("Block")
9055                   * ( V("Blank")^0 * parsers.eof
9056                     + ( V("Blank")^2 / writer.paragraphsep
9057                       + V("Blank")^0 / writer.interblocksep
9058                     )
9059                   )
9060                 + ( V("Paragraph") + V("Plain") )
9061                 * ( V("Blank")^0 * parsers.eof
9062                   + ( V("Blank")^2 / writer.paragraphsep
9063                     + V("Blank")^0 / writer.interblocksep
9064                   )
9065                 )
9066                 * V("Block")
9067                 * ( V("Blank")^0 * parsers.eof

```

```

9068             + ( V("Blank")^2 / writer.paragraphsep
9069             + V("Blank")^0 / writer.interblocksep
9070             )
9071             )
9072             + ( V("Paragraph") + V("Plain") )
9073             * ( V("Blank")^0 * parsers.eof
9074             + V("Blank")^0 / writer.paragraphsep
9075             )
9076             )^0,
9077
9078     ExpectedJekyllData = parsers.fail,
9079
9080     Blank = parsers.Blank,
9081     Reference = parsers.Reference,
9082
9083     Blockquote = parsers.Blockquote,
9084     Verbatim = parsers.Verbatim,
9085     ThematicBreak = parsers.ThematicBreak,
9086     BulletList = parsers.BulletList,
9087     OrderedList = parsers.OrderedList,
9088     DisplayHtml = parsers.DisplayHtml,
9089     Heading = parsers.Heading,
9090     Paragraph = parsers.Paragraph,
9091     Plain = parsers.Plain,
9092
9093     EndlineExceptions = parsers.EndlineExceptions,
9094     NoSoftLineBreakEndlineExceptions
9095     = parsers.NoSoftLineBreakEndlineExceptions,
9096
9097     Str = parsers.Str,
9098     Space = parsers.Space,
9099     NoSoftLineBreakSpace = parsers.NoSoftLineBreakSpace,
9100     OptionalIndent = parsers.OptionalIndent,
9101     Endline = parsers.Endline,
9102     EndlineNoSub = parsers.EndlineNoSub,
9103     NoSoftLineBreakEndline
9104     = parsers.NoSoftLineBreakEndline,
9105     EndlineBreak = parsers.EndlineBreak,
9106     LinkAndEmph = parsers.LinkAndEmph,
9107     Code = parsers.Code,
9108     AutoLinkUrl = parsers.AutoLinkUrl,
9109     AutoLinkEmail = parsers.AutoLinkEmail,
9110     AutoLinkRelativeReference
9111     = parsers.AutoLinkRelativeReference,
9112     InlineHtml = parsers.InlineHtml,
9113     HtmlEntity = parsers.HtmlEntity,
9114     EscapedChar = parsers.EscapedChar,

```

```

9115         Smart           = parsers.Smart,
9116         Symbol          = parsers.Symbol,
9117         SpecialChar     = parsers.fail,
9118         InitializeState = parsers.succeed,
9119     }

```

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]`.

```

9120     self.update_rule = function(rule_name, get_pattern)
9121         assert(current_extension_name ~= nil)
9122         assert(syntax[rule_name] ~= nil,
9123             [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
9124         local previous_pattern
9125         local extension_name
9126         if walkable_syntax[rule_name] then
9127             local previous_accountable_pattern = walkable_syntax[rule_name][1]
9128             previous_pattern = previous_accountable_pattern[1]
9129             extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
9130         else
9131             previous_pattern = nil
9132             extension_name = current_extension_name
9133         end
9134         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

```

```

9135         if type(get_pattern) == "function" then
9136             pattern = get_pattern(previous_pattern)
9137         else
9138             assert(previous_pattern == nil,
9139                 [[Rule ]] .. rule_name ..
9140                 [[ has already been updated by ]] .. extension_name)
9141             pattern = get_pattern
9142         end
9143         local accountable_pattern = { pattern, extension_name, rule_name }

```

```

9144     walkable_syntax[rule_name] = { accountable_pattern }
9145     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.

```

9146     local special_characters = {}
9147     self.add_special_character = function(c)
9148         table.insert(special_characters, c)
9149         syntax.SpecialChar = S(table.concat(special_characters, ""))
9150     end
9151
9152     self.add_special_character("*")
9153     self.add_special_character("[")
9154     self.add_special_character("]")
9155     self.add_special_character("<")
9156     self.add_special_character("!")
9157     self.add_special_character("\\")

```

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

```

9158     self.initialize_named_group = function(name, value)
9159         local pattern = Ct("")
9160         if value ~= nil then
9161             pattern = pattern / value
9162         end
9163         syntax.InitializeState = syntax.InitializeState
9164             * Cg(pattern, name)
9165     end

```

Add a named group for indentation.

```

9166     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

9167     for _, extension in ipairs(extensions) do
9168         current_extension_name = extension.name
9169         extension.extend_writer(writer)
9170         extension.extend_reader(self)
9171     end
9172     current_extension_name = nil

```

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

```

9173     if options.debugExtensions then
9174         local sorted_lhs = {}
9175         for lhs, _ in pairs(walkable_syntax) do
9176             table.insert(sorted_lhs, lhs)
9177         end

```

```

9178     table.sort(sorted_lhs)
9179
9180     local output_lines = {"{"}
9181     for lhs_index, lhs in ipairs(sorted_lhs) do
9182         local encoded_lhs = util.encode_json_string(lhs)
9183         table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": []})
9184         local rule = walkable_syntax[lhs]
9185         for rhs_index, rhs in ipairs(rule) do
9186             local human_readable_rhs
9187             if type(rhs) == "string" then
9188                 human_readable_rhs = rhs
9189             else
9190                 local pattern_name
9191                 if rhs[3] then
9192                     pattern_name = rhs[3]
9193                 else
9194                     pattern_name = "Anonymous Pattern"
9195                 end
9196                 local extension_name = rhs[2]
9197                 human_readable_rhs = pattern_name .. [{" ("] .. extension_name .. [{" )"]}
9198             end
9199             local encoded_rhs = util.encode_json_string(human_readable_rhs)
9200             local output_line = [{" "] .. encoded_rhs
9201             if rhs_index < #rule then
9202                 output_line = output_line .. ", "
9203             end
9204             table.insert(output_lines, output_line)
9205         end
9206         local output_line = [{" "]
9207         if lhs_index < #sorted_lhs then
9208             output_line = output_line .. ", "
9209         end
9210         table.insert(output_lines, output_line)
9211     end
9212     table.insert(output_lines, "}")
9213
9214     local output = table.concat(output_lines, "\n")
9215     local output_filename = options.debugExtensionsFileName
9216     local output_file = assert(io.open(output_filename, "w"),
9217         [[Could not open file ]] .. output_filename .. [{" " for writing]])
9218     assert(output_file:write(output))
9219     assert(output_file:close())
9220 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](#).

```

9221     for lhs, rule in pairs(walkable_syntax) do
9222         syntax[lhs] = parsers.fail
9223         for _, rhs in ipairs(rule) do
9224             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.

```

9225             if type(rhs) == "string" then
9226                 pattern = V(rhs)
9227             else
9228                 pattern = rhs[1]
9229                 if type(pattern) == "string" then
9230                     pattern = V(pattern)
9231                 end
9232             end
9233             syntax[lhs] = syntax[lhs] + pattern
9234         end
9235     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.

```

9236     if options.underscores then
9237         self.add_special_character("_")
9238     end
9239
9240     if not options.codeSpans then
9241         syntax.Code = parsers.fail
9242     else
9243         self.add_special_character("`")
9244     end
9245
9246     if not options.html then
9247         syntax.DisplayHtml = parsers.fail
9248         syntax.InlineHtml = parsers.fail
9249         syntax.HtmlEntity = parsers.fail
9250     else
9251         self.add_special_character("&")
9252     end
9253
9254     if options.preserveTabs then
9255         options.stripIndent = false
9256     end
9257
9258     if not options.smartEllipses then

```

```

9259     syntax.Smart = parsers.fail
9260 else
9261     self.add_special_character(".")
9262 end
9263
9264 if not options.relativeReferences then
9265     syntax.AutoLinkRelativeReference = parsers.fail
9266 end
9267
9268 if options.contentLevel == "inline" then
9269     syntax[1] = "Inlines"
9270     syntax.Inlines = V("InitializeState")
9271         * parsers.Inline^0
9272         * ( parsers.spacing^0
9273             * parsers.eof / "" )
9274     syntax.Space = parsers.Space + parsers.blankline / writer.space
9275 end
9276
9277 local blocks_nested_t = util.table_copy(syntax)
9278 blocks_nested_t.ExpectedJekyllData = parsers.fail
9279 parsers.blocks_nested = Ct(blocks_nested_t)
9280
9281 parsers.blocks = Ct(syntax)
9282
9283 local inlines_t = util.table_copy(syntax)
9284 inlines_t[1] = "Inlines"
9285 inlines_t.Inlines = V("InitializeState")
9286     * parsers.Inline^0
9287     * ( parsers.spacing^0
9288         * parsers.eof / "" )
9289 parsers.inlines = Ct(inlines_t)
9290
9291 local inlines_no_inline_note_t = util.table_copy(inlines_t)
9292 inlines_no_inline_note_t.InlineNote = parsers.fail
9293 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9294
9295 local inlines_no_html_t = util.table_copy(inlines_t)
9296 inlines_no_html_t.DisplayHtml = parsers.fail
9297 inlines_no_html_t.InlineHtml = parsers.fail
9298 inlines_no_html_t.HtmlEntity = parsers.fail
9299 parsers.inlines_no_html = Ct(inlines_no_html_t)
9300
9301 local inlines_nbsp_t = util.table_copy(inlines_t)
9302 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9303 inlines_nbsp_t.Space = parsers.NonbreakingSpace
9304 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9305

```



```

9306     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9307     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9308     inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9309     parsers.inlines_no_link_or_emphasis = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain T_EX output and returns it..

```

9310     return function(input)

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

9311         input = input:gsub("\r\n?", "\n")
9312         if input:sub(-1) ~= "\n" then
9313             input = input .. "\n"
9314         end

```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```

9315         references = {}
9316         local opt_string = {}
9317         for k, _ in pairs(defaultOptions) do
9318             local v = options[k]
9319             if type(v) == "table" then
9320                 for _, i in ipairs(v) do
9321                     opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9322                 end
9323             elseif k ~= "cacheDir" then
9324                 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9325             end
9326         end
9327         table.sort(opt_string)
9328         local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9329         local output
9330         local function convert(input)
9331             local document = self.parser_functions.parse_blocks(input)
9332             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`.

```

9333         local undosep_start, undosep_end
9334         local potential_secend_start, secend_start
9335         local potential_sep_start, sep_start
9336         while true do
9337             -- find a `writer->undosep`
9338             undosep_start, undosep_end = output:find(writer.undosep_text, 1, true)
9339             if undosep_start == nil then break end

```

```

9340     -- skip any preceding section ends
9341     secend_start = undosep_start
9342     while true do
9343         potential_secend_start = secend_start - #writer.secend_text
9344         if potential_secend_start < 1
9345             or output:sub(potential_secend_start, secend_start - 1) ~= writer.sece
9346             break
9347         end
9348         secend_start = potential_secend_start
9349     end
9350     -- find an immediately preceding block element / paragraph separator
9351     sep_start = secend_start
9352     potential_sep_start = sep_start - #writer.interblocksep_text
9353     if potential_sep_start >= 1
9354         and output:sub(potential_sep_start, sep_start - 1) == writer.interblocks
9355         sep_start = potential_sep_start
9356     else
9357         potential_sep_start = sep_start - #writer.paragraphsep_text
9358         if potential_sep_start >= 1
9359             and output:sub(potential_sep_start, sep_start - 1) == writer.paragraph
9360             sep_start = potential_sep_start
9361         end
9362     end
9363     -- remove `writer->undosep` and immediately preceding block element / parag
9364     output = output:sub(1, sep_start - 1)
9365         .. output:sub(secend_start, undosep_start - 1)
9366         .. output:sub(undosep_end + 1)
9367     end
9368     return output
9369 end

```

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

```

9370     if options.eagerCache or options.finalizeCache then
9371         local name = util.cache(options.cacheDir, input, salt, convert,
9372             ".md" .. writer.suffix)
9373         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

9374     else
9375         output = convert(input)
9376     end

```

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

```

9377     if options.finalizeCache then
9378         local file, mode

```

```

9379     if options.frozenCacheCounter > 0 then
9380         mode = "a"
9381     else
9382         mode = "w"
9383     end
9384     file = assert(io.open(options.frozenCacheFileName, mode),
9385         [[Could not open file "]] .. options.frozenCacheFileName
9386         .. [[ for writing]])
9387     assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
9388         .. [[markdownFrozenCache]] .. options.frozenCacheCounter
9389         .. [[\\endcsname{]] .. output .. [[]]] .. "\\n"))
9390     assert(file:close())
9391     end
9392     return output
9393 end
9394 end
9395 return self
9396 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.

```

9397 M.extensions = {}

```

3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```

9398 M.extensions.bracketed_spans = function()
9399     return {
9400         name = "built-in bracketed_spans syntax extension",
9401         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.

```

9402         function self.span(s, attr)
9403             if self.flatten_inlines then return s end
9404             return {"\\markdownRendererBracketedSpanAttributeContextBegin",
9405                 self.attributes(attr),
9406                 s,
9407                 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
9408         end
9409     end, extend_reader = function(self)

```

```

9410     local parsers = self.parsers
9411     local writer = self.writer
9412
9413     local span_label = parsers.lbracket
9414         * (Cs((parsers.alphanumeric^1
9415             + parsers.inticks
9416             + parsers.autolink
9417             + V("InlineHtml")
9418             + ( parsers.backslash * parsers.backslash)
9419             + ( parsers.backslash * (parsers.lbracket + parsers.rbracket
9420             + V("Space") + V("Endline")
9421             + (parsers.any
9422                 - (parsers.newline + parsers.lbracket + parsers.rbracket
9423                 + parsers.blankline^2))))^1)
9424         / self.parser_functions.parse_inlines)
9425     * parsers.rbracket
9426
9427     local Span = span_label
9428         * Ct(parsers.attributes)
9429         / writer.span
9430
9431     self.insert_pattern("Inline before LinkAndEmph",
9432         Span, "Span")
9433 end
9434 }
9435 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.

```

9436 M.extensions.citations = function(citation_nbsps)
9437   return {
9438     name = "built-in citations syntax extension",
9439     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.

```

9440     function self.citations(text_cites, cites)
9441         local buffer = {}
9442         if self.flatten_inlines then
9443             for _,cite in ipairs(cites) do
9444                 if cite.prenote then
9445                     table.insert(buffer, {cite.prenote, " "})
9446                 end
9447                 table.insert(buffer, cite.name)
9448                 if cite.postnote then
9449                     table.insert(buffer, {" ", cite.postnote})
9450                 end
9451             end
9452         else
9453             table.insert(buffer, {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
9454                 "#cites, "})
9455             for _,cite in ipairs(cites) do
9456                 table.insert(buffer, {cite.suppress_author and "-" or "+", "{",
9457                     cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}")})
9458             end
9459         end
9460         return buffer
9461     end
9462 end, extend_reader = function(self)
9463     local parsers = self.parsers
9464     local writer = self.writer
9465
9466     local citation_chars
9467         = parsers.alphanumeric
9468         + S("#$%&-+<>~/_")
9469
9470     local citation_name
9471         = Cs(parsers.dash^-1) * parsers.at
9472         * Cs(citation_chars
9473             * (((citation_chars + parsers.internal_punctuation
9474                 - parsers.comma - parsers.semicolon)
9475                 * -#((parsers.internal_punctuation - parsers.comma
9476                     - parsers.semicolon)^0
9477                     * -(citation_chars + parsers.internal_punctuation
9478                         - parsers.comma - parsers.semicolon)))^0
9479                 * citation_chars)^-1)
9480
9481     local citation_body_prenote
9482         = Cs((parsers.alphanumeric^1

```

```

9483         + parsers.bracketed
9484         + parsers.inticks
9485         + parsers.autolink
9486         + V("InlineHtml")
9487         + V("Space") + V("Endline")
9488         + (parsers.anyescaped
9489           - (parsers.newline + parsers.rbracket + parsers.blankline~
9490             - (parsers.spnl * parsers.dash~-1 * parsers.at))^1)
9491
9492     local citation_body_postnote
9493         = Cs((parsers.alphanumeric~1
9494             + parsers.bracketed
9495             + parsers.inticks
9496             + parsers.autolink
9497             + V("InlineHtml")
9498             + V("Space") + V("Endline")
9499             + (parsers.anyescaped
9500               - (parsers.newline + parsers.rbracket + parsers.semicolon
9501                 + parsers.blankline~2))
9502             - (parsers.spnl * parsers.rbracket))^1)
9503
9504     local citation_body_chunk
9505         = ( citation_body_prenote
9506           * parsers.spnlc_sep
9507           + Cc("")
9508           * parsers.spnlc
9509           )
9510         * citation_name
9511         * (parsers.internal_punctuation - parsers.semicolon)^-
1
9512         * ( parsers.spnlc
9513           * citation_body_postnote
9514           + Cc("")
9515           * parsers.spnlc
9516           )
9517
9518     local citation_body
9519         = citation_body_chunk
9520         * ( parsers.semicolon
9521           * parsers.spnlc
9522           * citation_body_chunk
9523           )^0
9524
9525     local citation_headless_body_postnote
9526         = Cs((parsers.alphanumeric~1
9527             + parsers.bracketed
9528             + parsers.inticks

```

```

9529         + parsers.autolink
9530         + V("InlineHtml")
9531         + V("Space") + V("Endline")
9532         + (parsers.anyescaped
9533           - (parsers.newline + parsers.rbracket + parsers.at
9534             + parsers.semicolon + parsers.blankline^2))
9535         - (parsers.spnl * parsers.rbracket))^0)
9536
9537     local citation_headless_body
9538         = citation_headless_body_postnote
9539         * ( parsers.semicolon
9540           * parsers.spnlc
9541           * citation_body_chunk
9542         )^0
9543
9544     local citations
9545         = function(text_cites, raw_cites)
9546         local function normalize(str)
9547             if str == "" then
9548                 str = nil
9549             else
9550                 str = (citation_nbsps and
9551                       self.parser_functions.parse_inlines_nbsp or
9552                       self.parser_functions.parse_inlines)(str)
9553             end
9554             return str
9555         end
9556
9557         local cites = {}
9558         for i = 1,#raw_cites,4 do
9559             cites[#cites+1] = {
9560                 prenote = normalize(raw_cites[i]),
9561                 suppress_author = raw_cites[i+1] == "-",
9562                 name = writer.identifier(raw_cites[i+2]),
9563                 postnote = normalize(raw_cites[i+3]),
9564             }
9565         end
9566         return writer.citations(text_cites, cites)
9567     end
9568
9569     local TextCitations
9570         = Ct((parsers.spnlc
9571             * Cc("")
9572             * citation_name
9573             * ((parsers.spnlc
9574                 * parsers.lbracket
9575                 * citation_headless_body

```

```

9576             * parsers.rbracket) + Cc(""))^1)
9577         / function(raw_cites)
9578             return citations(true, raw_cites)
9579         end
9580
9581     local ParenthesizedCitations
9582         = Ct((parsers.spnlc
9583             * parsers.lbracket
9584             * citation_body
9585             * parsers.rbracket)^1)
9586         / function(raw_cites)
9587             return citations(false, raw_cites)
9588         end
9589
9590     local Citations = TextCitations + ParenthesizedCitations
9591
9592     self.insert_pattern("Inline before LinkAndEmph",
9593         Citations, "Citations")
9594
9595     self.add_special_character("@")
9596     self.add_special_character("-")
9597 end
9598 }
9599 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.

```

9600 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.

```

9601 local languages_json = (function()
9602     local base, prev, curr
9603     for _, pathname in ipairs{kpse.lookup(language_map, { all=true })} do
9604         local file = io.open(pathname, "r")
9605         if not file then goto continue end
9606         local input = assert(file:read("*a"))
9607         assert(file:close())
9608         local json = input:gsub('[^\n]-:', '[%1]=')
9609         curr = load("_ENV = {}; return "..json")()
9610         if type(curr) == "table" then
9611             if base == nil then

```



```

9612         base = curr
9613     else
9614         setmetatable(prev, { __index = curr })
9615     end
9616     prev = curr
9617 end
9618 ::continue::
9619 end
9620 return base or {}
9621 end)()
9622
9623 return {
9624     name = "built-in content_blocks syntax extension",
9625     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.

```

9626     function self.contentblock(src,suf,type,tit)
9627         if not self.is_writing then return "" end
9628         src = src..".."..suf
9629         suf = suf:lower()
9630         if type == "onlineimage" then
9631             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
9632                 {"",self.string(src),"} ",
9633                 {"",self.uri(src),"} ",
9634                 {"",self.string(tit or ""),"}"}
9635         elseif languages_json[suf] then
9636             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
9637                 {"",self.string(languages_json[suf]),"} ",
9638                 {"",self.string(src),"} ",
9639                 {"",self.uri(src),"} ",
9640                 {"",self.string(tit or ""),"}"}
9641         else
9642             return {"\\markdownRendererContentBlock{" ,suf,"} ",
9643                 {"",self.string(src),"} ",
9644                 {"",self.uri(src),"} ",
9645                 {"",self.string(tit or ""),"}"}
9646         end
9647     end
9648 end, extend_reader = function(self)
9649     local parsers = self.parsers
9650     local writer = self.writer
9651
9652     local contentblock_tail
9653         = parsers.optionaltitle

```

```

9654         * (parsers.newline + parsers.eof)
9655
9656     -- case insensitive online image suffix:
9657     local onlineimagesuffix
9658         = (function(...)
9659             local parser = nil
9660             for _, suffix in ipairs({...}) do
9661                 local pattern=nil
9662                 for i=1,#suffix do
9663                     local char=suffix:sub(i,i)
9664                     char = S(char:lower()..char:upper())
9665                     if pattern == nil then
9666                         pattern = char
9667                     else
9668                         pattern = pattern * char
9669                     end
9670                 end
9671                 if parser == nil then
9672                     parser = pattern
9673                 else
9674                     parser = parser + pattern
9675                 end
9676             end
9677             return parser
9678         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
9679
9680     -- online image url for iA Writer content blocks with mandatory suffix,
9681     -- allowing nested brackets:
9682     local onlineimageurl
9683         = (parsers.less
9684             * Cs((parsers.anyescaped
9685                 - parsers.more
9686                 - parsers.spacing
9687                 - #(parsers.period
9688                     * onlineimagesuffix
9689                     * parsers.more
9690                     * contentblock_tail))^0)
9691             * parsers.period
9692             * Cs(onlineimagesuffix)
9693             * parsers.more
9694             + (Cs((parsers.inparens
9695                 + (parsers.anyescaped
9696                     - parsers.spacing
9697                     - parsers.rparent
9698                     - #(parsers.period
9699                         * onlineimagesuffix
9700                         * contentblock_tail))))^0)

```

```

9701         * parsers.period
9702         * Cs(onlineimagesuffix))
9703     ) * Cc("onlineimage")
9704
9705     -- filename for iA Writer content blocks with mandatory suffix:
9706     local localfilepath
9707         = parsers.slash
9708         * Cs((parsers.anyescaped
9709             - parsers.tab
9710             - parsers.newline
9711             - #(parsers.period
9712                 * parsers.alphanumeric^1
9713                 * contentblock_tail))^1)
9714         * parsers.period
9715         * Cs(parsers.alphanumeric^1)
9716         * Cc("localfile")
9717
9718     local ContentBlock
9719         = parsers.check_trail_no_rem
9720         * (localfilepath + onlineimageurl)
9721         * contentblock_tail
9722         / writer.contentblock
9723
9724     self.insert_pattern("Block before Blockquote",
9725                       ContentBlock, "ContentBlock")
9726 end
9727 }
9728 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.

```

9729 M.extensions.definition_lists = function(tight_lists)
9730   return {
9731     name = "built-in definition_lists syntax extension",
9732     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.

```

9733     local function dlitem(term, defs)
9734       local retVal = {"\\markdownRendererDlItem{",term,""}
9735       for _, def in ipairs(defs) do
9736         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,

```

```

9737             "\\markdownRendererDlDefinitionEnd "}
9738         end
9739         retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
9740         return retVal
9741     end
9742
9743     function self.definitionlist(items,tight)
9744         if not self.is_writing then return "" end
9745         local buffer = {}
9746         for _,item in ipairs(items) do
9747             buffer[#buffer + 1] = dlitem(item.term, item.definitions)
9748         end
9749         if tight and tight_lists then
9750             return {"\\markdownRendererDlBeginTight\n", buffer,
9751                 "\n\\markdownRendererDlEndTight"}
9752         else
9753             return {"\\markdownRendererDlBegin\n", buffer,
9754                 "\n\\markdownRendererDlEnd"}
9755         end
9756     end
9757 end, extend_reader = function(self)
9758     local parsers = self.parsers
9759     local writer = self.writer
9760
9761     local defstartchar = S("~:")
9762
9763     local defstart = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
9764                 * (parsers.tab + parsers.space^-
9765
9766     3)
9765         + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
9766                 * (parsers.tab + parsers.space^-
9767
9768     2)
9767         + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
9768                 * (parsers.tab + parsers.space^-
9769
9770     1)
9769         + parsers.check_trail_length(3) * defstartchar * #parsers.spaci
9770
9771     local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_
9772
9773     local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option
9774
9775     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
9776
9777     local indented_blocks = function(bl)
9778         return Cs( bl
9779             * (blank^1 * (parsers.check_minimal_indent / ""))
9780             * parsers.check_code_trail * -parsers.blankline * bl)^0

```

```

9781         * (blank^1 + parsers.eof))
9782     end
9783
9784     local function definition_list_item(term, defs, _)
9785         return { term = self.parser_functions.parse_inlines(term),
9786                 definitions = defs }
9787     end
9788
9789     local DefinitionListItemLoose
9790         = C(parsers.line) * blank^0
9791         * Ct((parsers.check_minimal_indent * (defstart
9792             * indented_blocks(dlchunk)
9793             / self.parser_functions.parse_blocks_nested))^1)
9794         * Cc(false) / definition_list_item
9795
9796     local DefinitionListItemTight
9797         = C(parsers.line)
9798         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
9799             / self.parser_functions.parse_blocks_nested))^1)
9800         * Cc(true) / definition_list_item
9801
9802     local DefinitionList
9803         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
9804           + Ct(DefinitionListItemTight^1)
9805           * (blank^0
9806             * -DefinitionListItemLoose * Cc(true))
9807           ) / writer.definitionlist
9808
9809     self.insert_pattern("Block after Heading",
9810                       DefinitionList, "DefinitionList")
9811 end
9812 }
9813 end

```

3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

9814 M.extensions.fancy_lists = function()
9815     return {
9816         name = "built-in fancy_lists syntax extension",
9817         extend_writer = function(self)
9818             local options = self.options
9819

```

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.

```

9820     function self.fancylis(items,tight,startnum,numstyle,numdelim)
9821         if not self.is_writing then return "" end
9822         local buffer = {}
9823         local num = startnum
9824         for _,item in ipairs(items) do
9825             if item ~= "" then
9826                 buffer[#buffer + 1] = self.fancyitem(item,num)
9827             end
9828             if num ~= nil and item ~= "" then
9829                 num = num + 1
9830             end
9831         end
9832         local contents = util.intersperse(buffer,"\n")
9833         if tight and options.tightLists then
9834             return {"\markdownRendererFancyOlBeginTight{",
9835                 numstyle,"}{",numdelim,"}",contents,
9836                 "\n\markdownRendererFancyOlEndTight "}
9837         else
9838             return {"\markdownRendererFancyOlBegin{",
9839                 numstyle,"}{",numdelim,"}",contents,
9840                 "\n\markdownRendererFancyOlEnd "}
9841         end
9842     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.

```

9843     function self.fancyitem(s,num)

```

```

9844     if num ~= nil then
9845         return {"\\markdownRendererFancyO1ItemWithNumber{" ,num,"} ",s,
9846             "\\markdownRendererFancyO1ItemEnd "}
9847     else
9848         return {"\\markdownRendererFancyO1Item " ,s,"\\markdownRendererFancyO1ItemEnd"}
9849     end
9850 end
9851 end, extend_reader = function(self)
9852     local parsers = self.parsers
9853     local options = self.options
9854     local writer = self.writer
9855
9856     local function combine_markers_and_delims(markers, delims)
9857         local markers_table = {}
9858         for _,marker in ipairs(markers) do
9859             local start_marker
9860             local continuation_marker
9861             if type(marker) == "table" then
9862                 start_marker = marker[1]
9863                 continuation_marker = marker[2]
9864             else
9865                 start_marker = marker
9866                 continuation_marker = marker
9867             end
9868             for _,delim in ipairs(delims) do
9869                 table.insert(markers_table, {start_marker, continuation_marker, delim})
9870             end
9871         end
9872         return markers_table
9873     end
9874
9875     local function join_table_with_func(func, markers_table)
9876         local pattern = func(table.unpack(markers_table[1]))
9877         for i = 2, #markers_table do
9878             pattern = pattern + func(table.unpack(markers_table[i]))
9879         end
9880         return pattern
9881     end
9882
9883     local lowercase_letter_marker = R("az")
9884     local uppercase_letter_marker = R("AZ")
9885
9886     local roman_marker = function(chars)
9887         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
9888         local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
9889         return m^-3
9890             * (c*m + c*d + d^-1 * c^-3)

```

```

9891         * (x*c + x*1 + 1^-1 * x^-3)
9892         * (i*x + i*v + v^-1 * i^-3)
9893     end
9894
9895     local lowercase_roman_marker = roman_marker({"m", "d", "c", "l", "x", "v", "i"}
9896     local uppercase_roman_marker = roman_marker({"M", "D", "C", "L", "X", "V", "I"}
9897
9898     local lowercase_opening_roman_marker = P("i")
9899     local uppercase_opening_roman_marker = P("I")
9900
9901     local digit_marker = parsers.dig * parsers.dig^-8
9902
9903     local markers = {
9904         {lowercase_opening_roman_marker, lowercase_roman_marker},
9905         {uppercase_opening_roman_marker, uppercase_roman_marker},
9906         lowercase_letter_marker,
9907         uppercase_letter_marker,
9908         lowercase_roman_marker,
9909         uppercase_roman_marker,
9910         digit_marker
9911     }
9912
9913     local delims = {
9914         parsers.period,
9915         parsers.rparent
9916     }
9917
9918     local markers_table = combine_markers_and_delims(markers, delims)
9919
9920     local function enumerator(start_marker, _, delimiter_type, interrupting)
9921         local delimiter_range
9922         local allowed_end
9923         if interrupting then
9924             delimiter_range = P("1")
9925             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
9926         else
9927             delimiter_range = start_marker
9928             allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
9929         end
9930
9931         return parsers.check_trail
9932             * Ct(C(delimiter_range) * C(delimiter_type))
9933             * allowed_end
9934     end
9935
9936     local starter = join_table_with_func(enumerator, markers_table)
9937

```



```

9938     local TightListItem = function(starter)
9939         return parsers.add_indent(starter, "li")
9940             * parsers.indented_content_tight
9941     end
9942
9943     local LooseListItem = function(starter)
9944         return parsers.add_indent(starter, "li")
9945             * parsers.indented_content_loose
9946             * remove_indent("li")
9947     end
9948
9949     local function roman2number(roman)
9950         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
9951             local numeral = 0
9952
9953             local i = 1
9954             local len = string.len(roman)
9955             while i < len do
9956                 local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
9957                     if z1 < z2 then
9958                         numeral = numeral + (z2 - z1)
9959                         i = i + 2
9960                     else
9961                         numeral = numeral + z1
9962                         i = i + 1
9963                     end
9964                 end
9965                 if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
9966             return numeral
9967         end
9968
9969         local function sniffstyle(numstr, delimend)
9970             local numdelim
9971             if delimend == ")" then
9972                 numdelim = "OneParen"
9973             elseif delimend == "." then
9974                 numdelim = "Period"
9975             else
9976                 numdelim = "Default"
9977             end
9978
9979             local num
9980             num = numstr:match("^([I])$")
9981             if num then
9982                 return roman2number(num), "UpperRoman", numdelim
9983             end
9984             num = numstr:match("^([i])$")

```

```

9985     if num then
9986         return roman2number(string.upper(num)), "LowerRoman", numdelim
9987     end
9988     num = numstr:match("^([A-Z])$")
9989     if num then
9990         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
9991     end
9992     num = numstr:match("^([a-z])$")
9993     if num then
9994         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
9995     end
9996     num = numstr:match("^([IVXLCDM]+)")
9997     if num then
9998         return roman2number(num), "UpperRoman", numdelim
9999     end
10000    num = numstr:match("^([ivxlcdm]+)")
10001    if num then
10002        return roman2number(string.upper(num)), "LowerRoman", numdelim
10003    end
10004    return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10005 end
10006
10007 local function fancylist(items,tight,start)
10008     local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
10009     return writer.fancylist(items,tight,
10010                             options.startNumber and startnum or 1,
10011                             numstyle or "Decimal",
10012                             numdelim or "Default")
10013 end
10014
10015 local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
10016     local enumerator_start = enumerator(start_marker, continuation_marker, delimi
10017     local enumerator_cont = enumerator(continuation_marker, continuation_marker,
10018     return Cg(enumerator_start, "listtype")
10019         * (Ct( TightListItem(Cb("listtype"))
10020             * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
10021             * Cc(true)
10022             * -#((parsers.conditionally_indented_blankline^0 / ""))
10023             * parsers.check_minimal_indent * enumerator_cont)
10024         + Ct( LooseListItem(Cb("listtype"))
10025             * ((parsers.conditionally_indented_blankline^0 / ""))
10026             * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_co
10027             * Cc(false)
10028             ) * Ct(Cb("listtype"))) / fancylist
10029 end
10030
10031 local FancyList = join_table_with_func(FancyListOfType, markers_table)

```

```

10032
10033     local Endline = parsers.newline
10034                 * (parsers.check_minimal_indent
10035                   * -parsers.EndlineExceptions
10036                   + parsers.check_optional_indent
10037                   * -parsers.EndlineExceptions
10038                   * -starter)
10039                 * parsers.spacechar^0
10040                 / writer.soft_line_break
10041
10042     self.update_rule("OrderedList", FancyList)
10043     self.update_rule("Endline", Endline)
10044 end
10045 }
10046 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.

```

10047 M.extensions.fenced_code = function(blank_before_code_fence,
10048                                     allow_attributes,
10049                                     allow_raw_blocks)
10050   return {
10051     name = "built-in fenced_code syntax extension",
10052     extend_writer = function(self)
10053       local options = self.options
10054

```

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.

```

10055     function self.fencedCode(s, i, attr)
10056       if not self.is_writing then return "" end
10057       s = s:gsub("\n$", "")
10058       local buf = {}
10059       if attr ~= nil then
10060         table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
10061                           self.attributes(attr)})
10062       end
10063       local name = util.cache_verbatim(options.cacheDir, s)
10064       table.insert(buf, {"\\markdownRendererInputFencedCode{"},

```

```

10065             name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
10066     if attr ~= nil then
10067         table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd{")
10068     end
10069     return buf
10070 end
10071

```

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

```

10072     if allow_raw_blocks then
10073         function self.rawBlock(s, attr)
10074             if not self.is_writing then return "" end
10075             s = s:gsub("\n$", "")
10076             local name = util.cache_verbatim(options.cacheDir, s)
10077             return {"\\markdownRendererInputRawBlock{" ,
10078                 name,"}{" , self.string(attr),"}"}
10079         end
10080     end
10081 end, extend_reader = function(self)
10082     local parsers = self.parsers
10083     local writer = self.writer
10084
10085     local function captures_geq_length(_,i,a,b)
10086         return #a >= #b and i
10087     end
10088
10089     local function strip_enclosing_whitespaces(str)
10090         return str:gsub("^%s*(.)%s*$", "%1")
10091     end
10092
10093     local tilde_infostring = Cs(Cs((V("HtmlEntity")
10094         + parsers.anyescaped
10095         - parsers.newline)^0)
10096         / strip_enclosing_whitespaces)
10097
10098     local backtick_infostring = Cs(Cs((V("HtmlEntity")
10099         + (-#(parsers.backslash * parsers.backtick) *
10100         - parsers.newline
10101         - parsers.backtick)^0)
10102         / strip_enclosing_whitespaces)
10103
10104     local fenceindent
10105
10106     local function has_trail(indent_table)
10107         return indent_table ~= nil and
10108             indent_table.trail ~= nil and

```

```

10109         next(indent_table.trail) ~= nil
10110     end
10111
10112     local function has_indents(indent_table)
10113         return indent_table ~= nil and
10114             indent_table.indents ~= nil and
10115             next(indent_table.indents) ~= nil
10116     end
10117
10118     local function get_last_indent_name(indent_table)
10119         if has_indents(indent_table) then
10120             return indent_table.indents[#indent_table.indents].name
10121         end
10122     end
10123
10124     local function count_fenced_start_indent(_, _, indent_table, trail)
10125         local last_indent_name = get_last_indent_name(indent_table)
10126         fenceindent = 0
10127         if last_indent_name ~= "li" then
10128             fenceindent = #trail
10129         end
10130         return true
10131     end
10132
10133     local fencehead      = function(char, infostring)
10134         return          Cmt(Cb("indent_info") * parsers.check_trail, count_fence
10135             * Cg(char^3, "fencelength")
10136             * parsers.optionalspace
10137             * infostring
10138             * (parsers.newline + parsers.eof)
10139     end
10140
10141     local fencetail      = function(char)
10142         return          parsers.check_trail_no_rem
10143             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10144             * parsers.optionalspace * (parsers.newline + parsers.eof)
10145             + parsers.eof
10146     end
10147
10148     local function process_fenced_line(s, i, indent_table, line_content, is_blank)
10149         local remainder = ""
10150         if has_trail(indent_table) then
10151             remainder = indent_table.trail.internal_remainder
10152         end
10153
10154         if is_blank and get_last_indent_name(indent_table) == "li" then
10155             remainder = ""

```

```

10156         end
10157
10158         local str = remainder .. line_content
10159         local index = 1
10160         local remaining = fenceindent
10161
10162         while true do
10163             local c = str:sub(index, index)
10164             if c == " " and remaining > 0 then
10165                 remaining = remaining - 1
10166                 index = index + 1
10167             elseif c == "\t" and remaining > 3 then
10168                 remaining = remaining - 4
10169                 index = index + 1
10170             else
10171                 break
10172             end
10173         end
10174
10175         return true, str:sub(index)
10176     end
10177
10178     local fencedline = function(char)
10179         return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),
10180     end
10181
10182     local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true))
10183
10184     local TildeFencedCode
10185         = fencehead(parsers.tilde, tilde_infostring)
10186         * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10187             + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
10188         * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
10189
10190     local BacktickFencedCode
10191         = fencehead(parsers.backtick, backtick_infostring)
10192         * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10193             + (parsers.check_minimal_indent / "") * fencedline(parsers.backtick)
10194         * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10195
10196     local infostring_with_attributes
10197         = Ct(C((parsers.linechar
10198             - ( parsers.optionalspace
10199                 * parsers.attributes))^0)
10200         * parsers.optionalspace
10201         * Ct(parsers.attributes))
10202

```

```

10203     local FencedCode
10204         = ((TildeFencedCode + BacktickFencedCode)
10205           / function(infostring, code)
10206               local expanded_code = self.expandtabs(code)
10207
10208               if allow_raw_blocks then
10209                   local raw_attr = lpeg.match(parsers.raw_attribute,
10210                                               infostring)
10211
10212                   if raw_attr then
10213                       return writer.rawBlock(expanded_code, raw_attr)
10214                   end
10215               end
10216
10217               local attr = nil
10218               if allow_attributes then
10219                   local match = lpeg.match(infostring_with_attributes,
10220                                               infostring)
10221
10222                   if match then
10223                       infostring, attr = table.unpack(match)
10224                   end
10225               end
10226               return writer.fencedCode(expanded_code, infostring, attr)
10227           end)
10228
10229     self.insert_pattern("Block after Verbatim",
10230                       FencedCode, "FencedCode")
10231
10232     local fencestart
10233     if blank_before_code_fence then
10234         fencestart = parsers.fail
10235     else
10236         fencestart = fencehead(parsers.backtick, backtick_infostring)
10237         + fencehead(parsers.tilde, tilde_infostring)
10238     end
10239
10240     self.update_rule("EndlineExceptions", function(previous_pattern)
10241         if previous_pattern == nil then
10242             previous_pattern = parsers.EndlineExceptions
10243         end
10244         return previous_pattern + fencestart
10245     end)
10246
10247     self.add_special_character("`")
10248     self.add_special_character("~")
10249 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.

```
10250 M.extensions.fenced_divs = function(blank_before_div_fence)
10251   return {
10252     name = "built-in fenced_divs syntax extension",
10253     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.

```
10254     function self.div_begin(attributes)
10255       local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",
10256                             self.attributes(attributes)}
10257       local end_output = {"\\markdownRendererFencedDivAttributeContextEnd{}}
10258       return self.push_attributes("div", attributes, start_output, end_output)
10259     end
```

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

```
10260     function self.div_end()
10261       return self.pop_attributes("div")
10262     end
10263   end, extend_reader = function(self)
10264     local parsers = self.parsers
10265     local writer = self.writer
```

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

```
10266     local fenced_div_infostring
10267         = C((parsers.linechar
10268             - ( parsers.spacechar^1
10269               * parsers.colon^1))^1)
10270
10271     local fenced_div_begin = parsers.nonindentSPACE
10272         * parsers.colon^3
10273         * parsers.optionalspace
10274         * fenced_div_infostring
10275         * ( parsers.spacechar^1
10276           * parsers.colon^1)^0
10277         * parsers.optionalspace
10278         * (parsers.newline + parsers.eof)
10279
10280     local fenced_div_end = parsers.nonindentSPACE
10281         * parsers.colon^3
10282         * parsers.optionalspace
10283         * (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.

```

10284     self.initialize_named_group("fenced_div_level", "0")
10285     self.initialize_named_group("fenced_div_num_opening_indents")
10286
10287     local function increment_div_level()
10288         local function push_indent_table(s, i, indent_table, -- luacheck: ignore s i
10289                                     fenced_div_num_opening_indents, fenced_div_l
10290             fenced_div_level = tonumber(fenced_div_level) + 1
10291             local num_opening_indents = 0
10292             if indent_table.indents ~= nil then
10293                 num_opening_indents = #indent_table.indents
10294             end
10295             fenced_div_num_opening_indents[fenced_div_level] = num_opening_indents
10296             return true, fenced_div_num_opening_indents
10297         end
10298
10299         local function increment_level(s, i, fenced_div_level) -- luacheck: ignore s i
10300             fenced_div_level = tonumber(fenced_div_level) + 1
10301             return true, tostring(fenced_div_level)
10302         end
10303
10304         return Cg( Cmt( Cb("indent_info")
10305                       * Cb("fenced_div_num_opening_indents")
10306                       * Cb("fenced_div_level"), push_indent_table)
10307                 , "fenced_div_num_opening_indents")
10308             * Cg( Cmt( Cb("fenced_div_level"), increment_level)
10309                 , "fenced_div_level")
10310         end
10311
10312     local function decrement_div_level()
10313         local function pop_indent_table(s, i, fenced_div_indent_table, fenced_div_lev
10314             fenced_div_level = tonumber(fenced_div_level)
10315             fenced_div_indent_table[fenced_div_level] = nil
10316             return true, tostring(fenced_div_level - 1)
10317         end
10318
10319         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
10320                       * Cb("fenced_div_level"), pop_indent_table)
10321                 , "fenced_div_level")
10322     end
10323

```

```

10324
10325     local non_fenced_div_block = parsers.check_minimal_indent * V("Block")
10326                               - parsers.check_minimal_indent_and_trail * fenced_d
10327
10328     local non_fenced_div_paragraph = parsers.check_minimal_indent * V("Paragraph")
10329                                   - parsers.check_minimal_indent_and_trail * fenc
10330
10331     local blank = parsers.minimally_indented_blank
10332
10333     local block_separated = parsers.block_sep_group(blank)
10334                             * non_fenced_div_block
10335
10336     local loop_body_pair = parsers.create_loop_body_pair(block_separated,
10337                                                         non_fenced_div_paragraph,
10338                                                         parsers.block_sep_group(t
10339                                                         parsers.par_sep_group(bla
10340
10341     local content_loop = ( non_fenced_div_block
10342                           * loop_body_pair.block^0
10343                           + non_fenced_div_paragraph
10344                           * block_separated
10345                           * loop_body_pair.block^0
10346                           + non_fenced_div_paragraph
10347                           * loop_body_pair.par^0)
10348                           * blank^0
10349
10350     local FencedDiv = fenced_div_begin
10351                       / function (infostring)
10352                           local attr = lpeg.match(Ct(parsers.attributes), infostring)
10353                           if attr == nil then
10354                               attr = {"." .. infostring}
10355                           end
10356                           return attr
10357                       end
10358                       / writer.div_begin
10359                       * increment_div_level()
10360                       * parsers.skipblanklines
10361                       * Ct(content_loop)
10362                       * parsers.minimally_indented_blank^0
10363                       * parsers.check_minimal_indent_and_trail * fenced_div_end
10364                       * decrement_div_level()
10365                       * (Cc("") / writer.div_end)
10366
10367     self.insert_pattern("Block after Verbatim",
10368                       FencedDiv, "FencedDiv")
10369
10370     self.add_special_character(":")

```

10371

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.

```
10372     local function is_inside_div()
10373         local function check_div_level(s, i, fenced_div_level) -- luacheck: ignore s i
10374             fenced_div_level = tonumber(fenced_div_level)
10375             return fenced_div_level > 0
10376         end
10377
10378         return Cmt(Cb("fenced_div_level"), check_div_level)
10379     end
10380
10381     local function check_indent()
10382         local function compare_indent(s, i, indent_table, -- luacheck: ignore s i
10383             fenced_div_num_opening_indents, fenced_div_level)
10384             fenced_div_level = tonumber(fenced_div_level)
10385             local num_current_indents = (indent_table.current_line_indents ~= nil and
10386                 #indent_table.current_line_indents) or 0
10387             local num_opening_indents = fenced_div_num_opening_indents[fenced_div_level]
10388             return num_current_indents == num_opening_indents
10389         end
10390
10391         return Cmt( Cb("indent_info")
10392             * Cb("fenced_div_num_opening_indents")
10393             * Cb("fenced_div_level"), compare_indent)
10394     end
10395
10396     local fencestart = is_inside_div()
10397         * fenced_div_end
10398         * check_indent()
10399
10400     if not blank_before_div_fence then
10401         self.update_rule("EndlineExceptions", function(previous_pattern)
10402             if previous_pattern == nil then
10403                 previous_pattern = parsers.EndlineExceptions
10404             end
10405             return previous_pattern + fencestart
10406         end)
10407     end
10408 end
10409 }
10410 end
```

3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```
10411 M.extensions.header_attributes = function()
10412   return {
10413     name = "built-in header_attributes syntax extension",
10414     extend_writer = function()
10415     end, extend_reader = function(self)
10416       local parsers = self.parsers
10417       local writer = self.writer
10418
10419       local function strip_atx_end(s)
10420         return s:gsub("%s+##%s*$", "")
10421       end
10422
10423       local AtxHeading = Cg(parsers.heading_start, "level")
10424         * parsers.optionalspace
10425         * (C(((parsers.linechar
10426           - (parsers.attributes
10427             * parsers.optionalspace
10428             * parsers.newline)))
10429           * (parsers.linechar
10430             - parsers.lbrace)^0)^1)
10431         / strip_atx_end
10432         / parsers.parse_heading_text)
10433       * Cg(Ct(parsers.newline
10434         + (parsers.attributes
10435           * parsers.optionalspace
10436           * parsers.newline)), "attributes")
10437       * Cb("level")
10438       * Cb("attributes")
10439       / writer.heading
10440
10441       local function strip_trailing_spaces(s)
10442         return s:gsub("%s*$", "")
10443       end
10444
10445       local heading_line = (parsers.linechar
10446         - (parsers.attributes
10447           * parsers.optionalspace
10448           * parsers.newline))^1
10449         - parsers.thematic_break_lines
10450
10451       local heading_text = heading_line
10452         * ((V("Endline") / "\n") * (heading_line - parsers.heading_
10453         * parsers.newline^-1)
10454
10455       local SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
```

```

10456         * #(heading_text
10457           * (parsers.attributes
10458             * parsers.optionalspace
10459             * parsers.newline)^-1
10460           * parsers.check_minimal_indent * parsers.check_trail *
10461           * Cs(heading_text) / strip_trailing_spaces
10462         / parsers.parse_heading_text
10463         * Cg(Ct((parsers.attributes
10464             * parsers.optionalspace
10465             * parsers.newline)^-1), "attributes")
10466         * parsers.check_minimal_indent_and_trail * parsers.heading
10467         * Cb("attributes")
10468         * parsers.newline
10469         * parsers.unfreeze_trail
10470       / writer.heading
10471
10472       local Heading = AtxHeading + SetextHeading
10473       self.update_rule("Heading", Heading)
10474     end
10475   }
10476 end

```

3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

10477 M.extensions.inline_code_attributes = function()
10478   return {
10479     name = "built-in inline_code_attributes syntax extension",
10480     extend_writer = function()
10481       end, extend_reader = function(self)
10482         local writer = self.writer
10483
10484         local CodeWithAttributes = parsers.inticks
10485           * Ct(parsers.attributes)
10486         / writer.code
10487
10488         self.insert_pattern("Inline before Code",
10489           CodeWithAttributes,
10490           "CodeWithAttributes")
10491       end
10492     }
10493 end

```

3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
10494 M.extensions.line_blocks = function()
```

```
10495   return {
```

```
10496     name = "built-in line_blocks syntax extension",
```

```
10497     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.

```
10498       function self.lineblock(lines)
```

```
10499         if not self.is_writing then return "" end
```

```
10500         local buffer = {}
```

```
10501         for i = 1, #lines - 1 do
```

```
10502           buffer[#buffer + 1] = { lines[i], self.hard_line_break }
```

```
10503         end
```

```
10504         buffer[#buffer + 1] = lines[#lines]
```

```
10505
```

```
10506         return {"\\markdownRendererLineBlockBegin\\n"
```

```
10507             ,buffer,
```

```
10508             "\\n\\markdownRendererLineBlockEnd "}
```

```
10509       end
```

```
10510     end, extend_reader = function(self)
```

```
10511       local parsers = self.parsers
```

```
10512       local writer = self.writer
```

```
10513
```

```
10514       local LineBlock = Ct(
```

```
10515         (Cs(
```

```
10516           ( (parsers.pipe * parsers.space)/""
```

```
10517             * ((parsers.space)/entities.char_entity("nbsp"))^0
```

```
10518             * parsers.linechar^0 * (parsers.newline/""))
```

```
10519             * (-parsers.pipe
```

```
10520               * (parsers.space^1/" ")
```

```
10521               * parsers.linechar^1
```

```
10522               * (parsers.newline/"")
```

```
10523             )^0
```

```
10524             * (parsers.blankline/"")^0
```

```
10525           ) / self.parser_functions.parse_inlines)^1) / writer.lineblock
```

```
10526
```

```
10527       self.insert_pattern("Block after Blockquote",
```

```
10528         LineBlock, "LineBlock")
```

```
10529     end
```

```
10530   }
```

```
10531 end
```

3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

10532 M.extensions.mark = function()
10533   return {
10534     name = "built-in mark syntax extension",
10535     extend_writer = function(self)

```

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

```

10536     function self.mark(s)
10537       if self.flatten_inlines then return s end
10538       return {"\\markdownRendererMark{" , s, "}" }
10539     end
10540   end, extend_reader = function(self)
10541     local parsers = self.parsers
10542     local writer = self.writer
10543
10544     local doubleequals = P("==")
10545
10546     local Mark = parsers.between(V("Inline"), doubleequals, doubleequals)
10547       / function (inlines) return writer.mark(inlines) end
10548
10549     self.add_special_character("=")
10550     self.insert_pattern("Inline before LinkAndEmph",
10551                       Mark, "Mark")
10552   end
10553 }
10554 end

```

3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

10555 M.extensions.link_attributes = function()
10556   return {
10557     name = "built-in link_attributes syntax extension",
10558     extend_writer = function()
10559     end, extend_reader = function(self)
10560       local parsers = self.parsers
10561       local options = self.options
10562

```

The following patterns define link reference definitions with attributes.

```

10563     local define_reference_parser = (parsers.check_trail / "") * parsers.link_label
10564                                     * parsers.spnlc * parsers.url
10565                                     * ( parsers.spnlc_sep * parsers.title * (parsers.
10566                                     * parsers.only_blank
10567                                     + parsers.spnlc_sep * parsers.title * parsers.c
10568                                     + Cc("") * (parsers.spnlc * Ct(parsers.attribut
10569                                     + Cc("") * parsers.only_blank)

```

```

10570
10571     local ReferenceWithAttributes = define_reference_parser
10572                                     / self.register_link
10573
10574     self.update_rule("Reference", ReferenceWithAttributes)
10575

```

The following patterns define direct and indirect links with attributes.

```

10576
10577     local LinkWithAttributesAndEmph = Ct(parsers.link_and_emph_table * Cg(Cc(true),
10578                                     / self.defer_link_and_emphasis_processing
10579
10580     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
10581

```

The following patterns define autolinks with attributes.

```

10582     local AutoLinkUrlWithAttributes
10583         = parsers.auto_link_url
10584         * Ct(parsers.attributes)
10585         / self.auto_link_url
10586
10587     self.insert_pattern("Inline before AutoLinkUrl",
10588                         AutoLinkUrlWithAttributes,
10589                         "AutoLinkUrlWithAttributes")
10590
10591     local AutoLinkEmailWithAttributes
10592         = parsers.auto_link_email
10593         * Ct(parsers.attributes)
10594         / self.auto_link_email
10595
10596     self.insert_pattern("Inline before AutoLinkEmail",
10597                         AutoLinkEmailWithAttributes,
10598                         "AutoLinkEmailWithAttributes")
10599
10600     if options.relativeReferences then
10601
10602         local AutoLinkRelativeReferenceWithAttributes
10603             = parsers.auto_link_relative_reference
10604             * Ct(parsers.attributes)
10605             / self.auto_link_url
10606
10607         self.insert_pattern(
10608             "Inline before AutoLinkRelativeReference",
10609             AutoLinkRelativeReferenceWithAttributes,
10610             "AutoLinkRelativeReferenceWithAttributes")
10611
10612     end
10613

```



```

10614     end
10615   }
10616 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.

```

10617 M.extensions.notes = function(notes, inline_notes)
10618   assert(notes or inline_notes)
10619   return {
10620     name = "built-in notes syntax extension",
10621     extend_writer = function(self)

```

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

```

10622     function self.note(s)
10623       if self.flatten_inlines then return "" end
10624       return {"\\markdownRendererNote{",s,""}
10625     end
10626   end, extend_reader = function(self)
10627     local parsers = self.parsers
10628     local writer = self.writer
10629
10630     if inline_notes then
10631       local InlineNote
10632         = parsers.circumflex
10633         * (parsers.link_label / self.parser_functions.parse_inlines_no_in
10634         / writer.note
10635
10636       self.insert_pattern("Inline after LinkAndEmph",
10637         InlineNote, "InlineNote")
10638     end
10639     if notes then
10640       local function strip_first_char(s)
10641         return s:sub(2)
10642       end
10643
10644       local RawNoteRef
10645         = #(parsers.lbracket * parsers.circumflex)
10646         * parsers.link_label / strip_first_char
10647
10648       local rawnotes = {}
10649
10650       -- like indirect_link

```

```

10651     local function lookup_note(ref)
10652         return writer.defer_call(function()
10653             local found = rawnotes[self.normalize_tag(ref)]
10654             if found then
10655                 return writer.note(
10656                     self.parser_functions.parse_blocks_nested(found))
10657             else
10658                 return {"[",
10659                     self.parser_functions.parse_inlines("^" .. ref), "]" }
10660             end
10661         end)
10662     end
10663
10664     local function register_note(ref,rawnote)
10665         local normalized_tag = self.normalize_tag(ref)
10666         if rawnotes[normalized_tag] == nil then
10667             rawnotes[normalized_tag] = rawnote
10668         end
10669         return ""
10670     end
10671
10672     local NoteRef = RawNoteRef / lookup_note
10673
10674     local optionally_indented_line = parsers.check_optional_indent_and_any_trail
10675
10676     local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt
10677
10678     local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)
10679
10680     local indented_blocks = function(bl)
10681         return Cs( bl
10682             * (blank^1 * (parsers.check_optional_indent / ""))
10683             * parsers.check_code_trail * -parsers.blankline * bl)^0)
10684     end
10685
10686     local NoteBlock
10687         = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10688         * parsers.spnlc * indented_blocks(chunk)
10689         / register_note
10690
10691     local Reference = NoteBlock + parsers.Reference
10692
10693     self.update_rule("Reference", Reference)
10694     self.insert_pattern("Inline before LinkAndEmph",
10695         NoteRef, "NoteRef")
10696 end
10697

```

```

10698     self.add_special_character("^")
10699   end
10700 }
10701 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.

```

10702 M.extensions.pipe_tables = function(table_captions, table_attributes)
10703
10704   local function make_pipe_table_rectangular(rows)
10705     local num_columns = #rows[2]
10706     local rectangular_rows = {}
10707     for i = 1, #rows do
10708       local row = rows[i]
10709       local rectangular_row = {}
10710       for j = 1, num_columns do
10711         rectangular_row[j] = row[j] or ""
10712       end
10713       table.insert(rectangular_rows, rectangular_row)
10714     end
10715     return rectangular_rows
10716   end
10717
10718   local function pipe_table_row(allow_empty_first_column
10719                                , nonempty_column
10720                                , column_separator
10721                                , column)
10722     local row_beginning
10723     if allow_empty_first_column then
10724       row_beginning = -- empty first column
10725                       #(parsers.spacechar^4
10726                        * column_separator)
10727                       * parsers.optionalspace
10728                       * column
10729                       * parsers.optionalspace
10730       -- non-empty first column
10731       + parsers.nonindentSPACE
10732       * nonempty_column^-1
10733       * parsers.optionalspace
10734     else
10735       row_beginning = parsers.nonindentSPACE

```

```

10736             * nonempty_column~-1
10737             * parsers.optionalspace
10738         end
10739
10740         return Ct(row_beginning
10741             * (-- single column with no leading pipes
10742               #(column_separator
10743                 * parsers.optionalspace
10744                 * parsers.newline)
10745               * column_separator
10746               * parsers.optionalspace
10747             -- single column with leading pipes or
10748             -- more than a single column
10749             + (column_separator
10750               * parsers.optionalspace
10751               * column
10752               * parsers.optionalspace)^1
10753             * (column_separator
10754               * parsers.optionalspace)^-1))
10755         end
10756
10757         return {
10758             name = "built-in pipe_tables syntax extension",
10759             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.

```

10760         function self.table(rows, caption, attributes)
10761             if not self.is_writing then return "" end
10762             local buffer = {}
10763             if attributes ~= nil then
10764                 table.insert(buffer,
10765                     "\\markdownRendererTableAttributeContextBegin\n")
10766                 table.insert(buffer, self.attributes(attributes))
10767             end
10768             table.insert(buffer,
10769                 {"\\markdownRendererTable{",
10770                 caption or "", "}{" , #rows - 1, "}{" ,
10771                 #rows[1], "}")
10772             local temp = rows[2] -- put alignments on the first row
10773             rows[2] = rows[1]
10774             rows[1] = temp
10775             for i, row in ipairs(rows) do
10776                 table.insert(buffer, "{")
10777                 for _, column in ipairs(row) do
10778                     if i > 1 then -- do not use braces for alignments

```

```

10779         table.insert(buffer, "{")
10780     end
10781     table.insert(buffer, column)
10782     if i > 1 then
10783         table.insert(buffer, "}")
10784     end
10785     end
10786     table.insert(buffer, "}")
10787 end
10788 if attributes ~= nil then
10789     table.insert(buffer,
10790         "\\markdownRendererTableAttributeContextEnd{")
10791 end
10792 return buffer
10793 end
10794 end, extend_reader = function(self)
10795     local parsers = self.parsers
10796     local writer = self.writer
10797
10798     local table_hline_separator = parsers.pipe + parsers.plus
10799
10800     local table_hline_column = (parsers.dash
10801         - #(parsers.dash
10802             * (parsers.spacechar
10803                 + table_hline_separator
10804                 + parsers.newline)))^1
10805     * (parsers.colon * Cc("r")
10806         + parsers.dash * Cc("d"))
10807     + parsers.colon
10808     * (parsers.dash
10809         - #(parsers.dash
10810             * (parsers.spacechar
10811                 + table_hline_separator
10812                 + parsers.newline)))^1
10813     * (parsers.colon * Cc("c")
10814         + parsers.dash * Cc("l"))
10815
10816     local table_hline = pipe_table_row(false
10817         , table_hline_column
10818         , table_hline_separator
10819         , table_hline_column)
10820
10821     local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
10822         * parsers.optionalspace * parsers.newline)^0
10823     * parsers.check_minimal_indent_and_trail
10824     * (P("Table")^-1 * parsers.colon)
10825     * parsers.optionalspace

```

```

10826
10827     local function strip_trailing_spaces(s)
10828         return s:gsub("%s*$", "")
10829     end
10830
10831     local table_row = pipe_table_row(true
10832                                     , (C((parsers.linechar - parsers.pipe)^1)
10833                                       / strip_trailing_spaces
10834                                       / self.parser_functions.parse_inlines)
10835                                     , parsers.pipe
10836                                     , (C((parsers.linechar - parsers.pipe)^0)
10837                                       / strip_trailing_spaces
10838                                       / self.parser_functions.parse_inlines))
10839
10840     local table_caption
10841     if table_captions then
10842         table_caption = #table_caption_beginning
10843                       * table_caption_beginning
10844         if table_attributes then
10845             table_caption = table_caption
10846                           * (C(((( parsers.linechar
10847                                 - (parsers.attributes
10848                                   * parsers.optionalspace
10849                                   * parsers.newline
10850                                   * -( parsers.optionalspace
10851                                     * parsers.linechar))))
10852                               + ( parsers.newline
10853                                 * #( parsers.optionalspace
10854                                   * parsers.linechar)
10855                                   * C(parsers.optionalspace) / writer.space))
10856                               * (parsers.linechar
10857                                 - parsers.lbrace)^0)^1)
10858                               / self.parser_functions.parse_inlines)
10859                           * (parsers.newline
10860                               + ( Ct(parsers.attributes)
10861                                   * parsers.optionalspace
10862                                   * parsers.newline))
10863         else
10864             table_caption = table_caption
10865                           * C(( parsers.linechar
10866                               + ( parsers.newline
10867                                   * #( parsers.optionalspace
10868                                       * parsers.linechar)
10869                                       * C(parsers.optionalspace) / writer.space))^1)
10870                               / self.parser_functions.parse_inlines
10871                               * parsers.newline
10872     end

```

```

10873     else
10874         table_caption = parsers.fail
10875     end
10876
10877     local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_indent
10878         * table_hline * parsers.newline
10879         * ((parsers.check_minimal_indent / {}) * table_row * parsers.
10880         / make_pipe_table_rectangular
10881         * table_caption~-1
10882         / writer.table
10883
10884     self.insert_pattern("Block after Blockquote",
10885         PipeTable, "PipeTable")
10886 end
10887 }
10888 end

```

3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

10889 M.extensions.raw_inline = function()
10890     return {
10891         name = "built-in raw_inline syntax extension",
10892         extend_writer = function(self)
10893             local options = self.options
10894

```

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

```

10895         function self.rawInline(s, attr)
10896             if not self.is_writing then return "" end
10897             if self.flatten_inlines then return s end
10898             local name = util.cache_verbatim(options.cacheDir, s)
10899             return {"\\markdownRendererInputRawInline{" ,
10900                 name,"}{" , self.string(attr),""}
10901         end
10902     end, extend_reader = function(self)
10903         local writer = self.writer
10904
10905         local RawInline = parsers.inticks
10906             * parsers.raw_attribute
10907             / writer.rawInline
10908
10909         self.insert_pattern("Inline before Code",
10910             RawInline, "RawInline")
10911     end
10912 }

```

```
10913 end
```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
10914 M.extensions.strike_through = function()
10915   return {
10916     name = "built-in strike_through syntax extension",
10917     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.

```
10918       function self.strike_through(s)
10919         if self.flatten_inlines then return s end
10920         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
10921       end
10922     end, extend_reader = function(self)
10923       local parsers = self.parsers
10924       local writer = self.writer
10925
10926       local StrikeThrough = (
10927         parsers.between(parsers.Inline, parsers.doubletildes,
10928           parsers.doubletildes)
10929       ) / writer.strike_through
10930
10931       self.insert_pattern("Inline after LinkAndEmph",
10932         StrikeThrough, "StrikeThrough")
10933
10934       self.add_special_character("~")
10935     end
10936   }
10937 end
```

3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
10938 M.extensions.subscripts = function()
10939   return {
10940     name = "built-in subscripts syntax extension",
10941     extend_writer = function(self)
```

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

```
10942       function self.subscript(s)
10943         if self.flatten_inlines then return s end
10944         return {"\\markdownRendererSubscript{" ,s,"}"}

```



```

10945     end
10946 end, extend_reader = function(self)
10947     local parsers = self.parsers
10948     local writer = self.writer
10949
10950     local Subscript = (
10951         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
10952     ) / writer.subscript
10953
10954     self.insert_pattern("Inline after LinkAndEmph",
10955                         Subscript, "Subscript")
10956
10957     self.add_special_character("~")
10958 end
10959 }
10960 end

```

3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

10961 M.extensions.superscripts = function()
10962     return {
10963         name = "built-in superscripts syntax extension",
10964         extend_writer = function(self)

```

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

```

10965         function self.superscript(s)
10966             if self.flatten_inlines then return s end
10967             return {"\\markdownRendererSuperscript{" ,s,""}"}
10968         end
10969     end, extend_reader = function(self)
10970         local parsers = self.parsers
10971         local writer = self.writer
10972
10973         local Superscript = (
10974             parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
10975         ) / writer.superscript
10976
10977         self.insert_pattern("Inline after LinkAndEmph",
10978                             Superscript, "Superscript")
10979
10980         self.add_special_character("^")
10981     end
10982 }
10983 end

```

3.1.7.19 T_EX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
10984 M.extensions.tex_math = function(tex_math_dollars,
10985                                     tex_math_single_backslash,
10986                                     tex_math_double_backslash)
10987   return {
10988     name = "built-in tex_math syntax extension",
10989     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.

```
10990     function self.display_math(s)
10991       if self.flatten_inlines then return s end
10992       return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
10993     end
```

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

```
10994     function self.inline_math(s)
10995       if self.flatten_inlines then return s end
10996       return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
10997     end
10998   end, extend_reader = function(self)
10999     local parsers = self.parsers
11000     local writer = self.writer
11001
11002     local function between(p, starter, ender)
11003       return (starter * Cs(p * (p - ender)^0) * ender)
11004     end
11005
11006     local function strip_preceding_whitespaces(str)
11007       return str:gsub("^%s*(.-)$", "%1")
11008     end
11009
11010     local allowed_before_closing = B( parsers.backslash * parsers.any
11011                                       + parsers.any * (parsers.any - parsers.backslash)
11012
11013     local allowed_before_closing_no_space = B( parsers.backslash * parsers.any
11014                                               + parsers.any * (parsers.nonspacechar
11015
```

The following patterns implement the Pandoc dollar math syntax extension.

```
11016     local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
11017                                                       + parsers.backslash^-1
11018                                                       * parsers.linechar)
11019                                   - parsers.blankline^2
11020                                   - parsers.dollar
```

```

11021
11022     local inline_math_opening_dollars = parsers.dollar
11023                                     * #(parsers.nonspacechar)
11024
11025     local inline_math_closing_dollars = allowed_before_closing_no_space
11026                                     * parsers.dollar
11027                                     * -#(parsers.digit)
11028
11029     local inline_math_dollars = between(Cs( dollar_math_content),
11030                                       inline_math_opening_dollars,
11031                                       inline_math_closing_dollars)
11032
11033     local display_math_opening_dollars = parsers.dollar
11034                                       * parsers.dollar
11035
11036     local display_math_closing_dollars = parsers.dollar
11037                                       * parsers.dollar
11038
11039     local display_math_dollars = between(Cs( dollar_math_content),
11040                                       display_math_opening_dollars,
11041                                       display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

11042     local backslash_math_content = (parsers.newline * (parsers.check_optional_inde
11043                                     + parsers.linechar)
11044                                     - parsers.blankline^2)

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

11045     local inline_math_opening_double = parsers.backslash
11046                                     * parsers.backslash
11047                                     * parsers.lparent
11048
11049     local inline_math_closing_double = allowed_before_closing
11050                                     * parsers.spacechar^0
11051                                     * parsers.backslash
11052                                     * parsers.backslash
11053                                     * parsers.rparent
11054
11055     local inline_math_double = between(Cs( backslash_math_content),
11056                                       inline_math_opening_double,
11057                                       inline_math_closing_double)
11058                                     / strip_preceding_whitespaces
11059
11060     local display_math_opening_double = parsers.backslash
11061                                       * parsers.backslash
11062                                       * parsers.lbracket

```

```

11063
11064     local display_math_closing_double = allowed_before_closing
11065         * parsers.spacechar^0
11066         * parsers.backslash
11067         * parsers.backslash
11068         * parsers.rbracket
11069
11070     local display_math_double = between(Cs( backslash_math_content),
11071         display_math_opening_double,
11072         display_math_closing_double)
11073         / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

11074     local inline_math_opening_single = parsers.backslash
11075         * parsers.lparent
11076
11077     local inline_math_closing_single = allowed_before_closing
11078         * parsers.spacechar^0
11079         * parsers.backslash
11080         * parsers.rparent
11081
11082     local inline_math_single = between(Cs( backslash_math_content),
11083         inline_math_opening_single,
11084         inline_math_closing_single)
11085         / strip_preceding_whitespaces
11086
11087     local display_math_opening_single = parsers.backslash
11088         * parsers.lbracket
11089
11090     local display_math_closing_single = allowed_before_closing
11091         * parsers.spacechar^0
11092         * parsers.backslash
11093         * parsers.rbracket
11094
11095     local display_math_single = between(Cs( backslash_math_content),
11096         display_math_opening_single,
11097         display_math_closing_single)
11098         / strip_preceding_whitespaces
11099
11100     local display_math = parsers.fail
11101
11102     local inline_math = parsers.fail
11103
11104     if tex_math_dollars then
11105         display_math = display_math + display_math_dollars
11106         inline_math = inline_math + inline_math_dollars
11107     end
11108

```

```

11109     if tex_math_double_backslash then
11110         display_math = display_math + display_math_double
11111         inline_math = inline_math + inline_math_double
11112     end
11113
11114     if tex_math_single_backslash then
11115         display_math = display_math + display_math_single
11116         inline_math = inline_math + inline_math_single
11117     end
11118
11119     local TexMath = display_math / writer.display_math
11120                   + inline_math / writer.inline_math
11121
11122     self.insert_pattern("Inline after LinkAndEmph",
11123                       TexMath, "TexMath")
11124
11125     if tex_math_dollars then
11126         self.add_special_character("$")
11127     end
11128
11129     if tex_math_single_backslash or tex_math_double_backslash then
11130         self.add_special_character("\\")
11131         self.add_special_character("[")
11132         self.add_special_character("]")
11133         self.add_special_character("(")
11134         self.add_special_character("(")
11135     end
11136 end
11137 }
11138 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.

```

11139 M.extensions.jekyll_data = function(expect_jekyll_data)
11140     return {
11141         name = "built-in jekyll_data syntax extension",
11142         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`.

```

11143 function self.jekyllData(d, t, p)
11144     if not self.is_writing then return "" end
11145
11146     local buf = {}
11147
11148     local keys = {}
11149     for k, _ in pairs(d) do
11150         table.insert(keys, k)
11151     end
11152     table.sort(keys)
11153
11154     if not p then
11155         table.insert(buf, "\\markdownRendererJekyllDataBegin")
11156     end
11157
11158     if #d > 0 then
11159         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
11160         table.insert(buf, self.identifier(p or "null"))
11161         table.insert(buf, "}{")
11162         table.insert(buf, #keys)
11163         table.insert(buf, "}")
11164     else
11165         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
11166         table.insert(buf, self.identifier(p or "null"))
11167         table.insert(buf, "}{")
11168         table.insert(buf, #keys)
11169         table.insert(buf, "}")
11170     end
11171
11172     for _, k in ipairs(keys) do
11173         local v = d[k]
11174         local typ = type(v)
11175         k = tostring(k or "null")
11176         if typ == "table" and next(v) ~= nil then
11177             table.insert(
11178                 buf,
11179                 self.jekyllData(v, t, k)
11180             )
11181         else
11182             k = self.identifier(k)
11183             v = tostring(v)
11184             if typ == "boolean" then
11185                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11186                 table.insert(buf, k)
11187                 table.insert(buf, "}{")
11188                 table.insert(buf, v)
11189                 table.insert(buf, "}")

```

```

11190         elseif typ == "number" then
11191             table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11192             table.insert(buf, k)
11193             table.insert(buf, "}{")
11194             table.insert(buf, v)
11195             table.insert(buf, "}")
11196         elseif typ == "string" then
11197             table.insert(buf, "\\markdownRendererJekyllDataString{")
11198             table.insert(buf, k)
11199             table.insert(buf, "}{")
11200             table.insert(buf, t(v))
11201             table.insert(buf, "}")
11202         elseif typ == "table" then
11203             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11204             table.insert(buf, k)
11205             table.insert(buf, "}")
11206         else
11207             error(format("Unexpected type %s for value of " ..
11208                 "YAML key %s", typ, k))
11209         end
11210     end
11211 end
11212
11213 if #d > 0 then
11214     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
11215 else
11216     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
11217 end
11218
11219 if not p then
11220     table.insert(buf, "\\markdownRendererJekyllDataEnd")
11221 end
11222
11223 return buf
11224 end
11225 end, extend_reader = function(self)
11226     local parsers = self.parsers
11227     local writer = self.writer
11228
11229     local JekyllData
11230         = Cmt( C((parsers.line - P("---") - P("..."))^0)
11231             , function(s, i, text) -- luacheck: ignore s i
11232                 local data
11233                 local ran_ok, _ = pcall(function()
11234                     -- TODO: Replace with `require("tinyyaml")` in TeX Live
11235                     local tinyyaml = require("markdown-tinyyaml")
11236                     data = tinyyaml.parse(text, {timestamps=false})

```

```

11237         end)
11238         if ran_ok and data ~= nil then
11239             return true, writer.jekyllData(data, function(s)
11240                 return self.parser_functions.parse_blocks_nested(s)
11241             end, nil)
11242         else
11243             return false
11244         end
11245     end
11246 )
11247
11248     local UnexpectedJekyllData
11249         = P("----")
11250         * parsers.blankline / 0
11251         * #(-parsers.blankline) -- if followed by blank, it's thematic b
11252         * JekyllData
11253         * (P("----") + P("..."))
11254
11255     local ExpectedJekyllData
11256         = ( P("----")
11257             * parsers.blankline / 0
11258             * #(-parsers.blankline) -- if followed by blank, it's thematic
11259             )^-1
11260         * JekyllData
11261         * (P("----") + P("..."))^-1
11262
11263     self.insert_pattern("Block before Blockquote",
11264                       UnexpectedJekyllData, "UnexpectedJekyllData")
11265     if expect_jekyll_data then
11266         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
11267     end
11268 end
11269 }
11270 end

```

3.1.8 Conversion from Markdown to Plain TeX

The `new` function returns a conversion function that takes a markdown string and turns it into a plain TeX output. See Section 2.1.1.

```
11271 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```

11272     options = options or {}
11273     setmetatable(options, { __index = function (_, key)
11274         return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.


```

11275 if options.singletonCache and singletonCache.convert then
11276   for k, v in pairs(defaultOptions) do
11277     if type(v) == "table" then
11278       for i = 1, math.max(#singletonCache.options[k], #options[k]) do
11279         if singletonCache.options[k][i] ~= options[k][i] then
11280           goto miss
11281         end
11282       end
11283       elseif singletonCache.options[k] ~= options[k] then
11284         goto miss
11285       end
11286     end
11287   return singletonCache.convert
11288 end
11289 ::miss::

```

Apply built-in syntax extensions based on [options](#).

```

11290 local extensions = {}
11291
11292 if options.bracketedSpans then
11293   local bracketed_spans_extension = M.extensions.bracketed_spans()
11294   table.insert(extensions, bracketed_spans_extension)
11295 end
11296
11297 if options.contentBlocks then
11298   local content_blocks_extension = M.extensions.content_blocks(
11299     options.contentBlocksLanguageMap)
11300   table.insert(extensions, content_blocks_extension)
11301 end
11302
11303 if options.definitionLists then
11304   local definition_lists_extension = M.extensions.definition_lists(
11305     options.tightLists)
11306   table.insert(extensions, definition_lists_extension)
11307 end
11308
11309 if options.fencedCode then
11310   local fenced_code_extension = M.extensions.fenced_code(
11311     options.blankBeforeCodeFence,
11312     options.fencedCodeAttributes,
11313     options.rawAttribute)
11314   table.insert(extensions, fenced_code_extension)
11315 end
11316
11317 if options.fencedDivs then
11318   local fenced_div_extension = M.extensions.fenced_divs(
11319     options.blankBeforeDivFence)
11320   table.insert(extensions, fenced_div_extension)

```

```

11321 end
11322
11323 if options.headerAttributes then
11324     local header_attributes_extension = M.extensions.header_attributes()
11325     table.insert(extensions, header_attributes_extension)
11326 end
11327
11328 if options.inlineCodeAttributes then
11329     local inline_code_attributes_extension =
11330         M.extensions.inline_code_attributes()
11331     table.insert(extensions, inline_code_attributes_extension)
11332 end
11333
11334 if options.jekyllData then
11335     local jekyll_data_extension = M.extensions.jekyll_data(
11336         options.expectJekyllData)
11337     table.insert(extensions, jekyll_data_extension)
11338 end
11339
11340 if options.linkAttributes then
11341     local link_attributes_extension =
11342         M.extensions.link_attributes()
11343     table.insert(extensions, link_attributes_extension)
11344 end
11345
11346 if options.lineBlocks then
11347     local line_block_extension = M.extensions.line_blocks()
11348     table.insert(extensions, line_block_extension)
11349 end
11350
11351 if options.mark then
11352     local mark_extension = M.extensions.mark()
11353     table.insert(extensions, mark_extension)
11354 end
11355
11356 if options.pipeTables then
11357     local pipe_tables_extension = M.extensions.pipe_tables(
11358         options.tableCaptions, options.tableAttributes)
11359     table.insert(extensions, pipe_tables_extension)
11360 end
11361
11362 if options.rawAttribute then
11363     local raw_inline_extension = M.extensions.raw_inline()
11364     table.insert(extensions, raw_inline_extension)
11365 end
11366
11367 if options.strikeThrough then

```

```

11368     local strike_through_extension = M.extensions.strike_through()
11369     table.insert(extensions, strike_through_extension)
11370 end
11371
11372 if options.subscripts then
11373     local subscript_extension = M.extensions.subscripts()
11374     table.insert(extensions, subscript_extension)
11375 end
11376
11377 if options.superscripts then
11378     local superscript_extension = M.extensions.superscripts()
11379     table.insert(extensions, superscript_extension)
11380 end
11381
11382 if options.texMathDollars or
11383     options.texMathSingleBackslash or
11384     options.texMathDoubleBackslash then
11385     local tex_math_extension = M.extensions.tex_math(
11386         options.texMathDollars,
11387         options.texMathSingleBackslash,
11388         options.texMathDoubleBackslash)
11389     table.insert(extensions, tex_math_extension)
11390 end
11391
11392 if options.notes or options.inlineNotes then
11393     local notes_extension = M.extensions.notes(
11394         options.notes, options.inlineNotes)
11395     table.insert(extensions, notes_extension)
11396 end
11397
11398 if options.citations then
11399     local citations_extension = M.extensions.citations(options.citationNbsps)
11400     table.insert(extensions, citations_extension)
11401 end
11402
11403 if options.fancyLists then
11404     local fancy_lists_extension = M.extensions.fancy_lists()
11405     table.insert(extensions, fancy_lists_extension)
11406 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

11407 for _, user_extension_filename in ipairs(options.extensions) do
11408     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

11409         local pathname = kpse.lookup(filename)
11410         local input_file = assert(io.open(pathname, "r"),
11411             [[Could not open user-defined syntax extension "]]

```

```

11412     .. pathname .. [{" for reading}])
11413 local input = assert(input_file:read("*a"))
11414 assert(input_file:close())
11415 local user_extension, err = load([[
11416     local sandbox = {}
11417     setmetatable(sandbox, {__index = _G})
11418     _ENV = sandbox
11419 ]]) .. input)()
11420 assert(user_extension,
11421     [[Failed to compile user-defined syntax extension "]]
11422     .. pathname .. [{": }]] .. (err or [{"}]))

```

Then, validate the user-defined syntax extension.

```

11423     assert(user_extension.api_version ~= nil,
11424         [[User-defined syntax extension "]] .. pathname
11425         .. [{" does not specify mandatory field "api_version"}])
11426     assert(type(user_extension.api_version) == "number",
11427         [[User-defined syntax extension "]] .. pathname
11428         .. [{" specifies field "api_version" of type "]]
11429         .. type(user_extension.api_version)
11430         .. [{" but "number" was expected}])
11431     assert(user_extension.api_version > 0
11432         and user_extension.api_version <= metadata.user_extension_api_version,
11433         [[User-defined syntax extension "]] .. pathname
11434         .. [{" uses syntax extension API version "]]
11435         .. user_extension.api_version .. [{" but markdown.lua }]]
11436         .. metadata.version .. [{" uses API version }]]
11437         .. metadata.user_extension_api_version
11438         .. [{" , which is incompatible}])
11439
11440     assert(user_extension.grammar_version ~= nil,
11441         [[User-defined syntax extension "]] .. pathname
11442         .. [{" does not specify mandatory field "grammar_version"}])
11443     assert(type(user_extension.grammar_version) == "number",
11444         [[User-defined syntax extension "]] .. pathname
11445         .. [{" specifies field "grammar_version" of type "]]
11446         .. type(user_extension.grammar_version)
11447         .. [{" but "number" was expected}])
11448     assert(user_extension.grammar_version == metadata.grammar_version,
11449         [[User-defined syntax extension "]] .. pathname
11450         .. [{" uses grammar version "]] .. user_extension.grammar_version
11451         .. [{" but markdown.lua }]] .. metadata.version
11452         .. [{" uses grammar version }]] .. metadata.grammar_version
11453         .. [{" , which is incompatible}])
11454
11455     assert(user_extension.finalize_grammar ~= nil,
11456         [[User-defined syntax extension "]] .. pathname
11457         .. [{" does not specify mandatory "finalize_grammar" field}])

```

```

11458     assert(type(user_extension.finalize_grammar) == "function",
11459           [[User-defined syntax extension "]] .. pathname
11460           .. [[ " specifies field "finalize_grammar" of type "]]
11461           .. type(user_extension.finalize_grammar)
11462           .. [[ " 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.)

```

11463     local extension = {
11464       name = [[user-defined "]] .. pathname .. [[ " syntax extension]],
11465       extend_reader = user_extension.finalize_grammar,
11466       extend_writer = function() end,
11467     }
11468     return extension
11469   end)(user_extension_filename)
11470   table.insert(extensions, user_extension)
11471 end

```

Produce a conversion function from markdown to plain \TeX .

```

11472 local writer = M.writer.new(options)
11473 local reader = M.reader.new(writer, options)
11474 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`.

```

11475 collectgarbage("collect")

```

Update the singleton cache.

```

11476 if options.singletonCache then
11477   local singletonCacheOptions = {}
11478   for k, v in pairs(options) do
11479     singletonCacheOptions[k] = v
11480   end
11481   setmetatable(singletonCacheOptions,
11482     { __index = function (_, key)
11483       return defaultOptions[key] end })
11484   singletonCache.options = singletonCacheOptions
11485   singletonCache.convert = convert
11486 end

```

Return the conversion function from markdown to plain \TeX .

```

11487 return convert
11488 end
11489
11490 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.

```
11491
11492 local input
11493 if input_filename then
11494   local input_file = assert(io.open(input_filename, "r"),
11495     [[Could not open file ]] .. input_filename .. [[ for reading]])
11496   input = assert(input_file:read("*a"))
11497   assert(input_file:close())
11498 else
11499   input = assert(io.read("*a"))
11500 end
11501
```

First, ensure that the `options.cacheDir` directory exists.

```
11502 local lfs = require("lfs")
11503 if options.cacheDir and not lfs.isdir(options.cacheDir) then
11504   assert(lfs.mkdir(options["cacheDir"]))
11505 end
```

If Kpathsea has not been loaded before or if LuaTeX has not yet been initialized, configure Kpathsea on top of loading it.

```
11506 local kpse
11507 (function()
11508   local should_initialize = package.loaded.kpse == nil
11509     or tex.initialize ~= nil
11510   kpse = require("kpse")
11511   if should_initialize then
11512     kpse.set_program_name("luatex")
11513   end
11514 end)()
11515 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.

```
11516 if metadata.version ~= md.metadata.version then
11517   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
11518     "markdown.lua " .. md.metadata.version .. ".")
11519 end
11520 local convert = md.new(options)
11521 local output = convert(input)
11522
11523 if output_filename then
11524   local output_file = assert(io.open(output_filename, "w"),
11525     [[Could not open file ]] .. output_filename .. [[ for writing]])
11526   assert(output_file:write(output))
```

```

11527  assert(output_file:close())
11528  else
11529    assert(io.write(output))
11530  end

```

Remove the `options.cacheDir` directory if it is empty.

```

11531  if options.cacheDir then
11532    lfs.rmdir(options["cacheDir"])
11533  end

```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX 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_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```

11534  \ExplSyntaxOn
11535  \cs_if_free:NT
11536    \markdownInfo
11537    {
11538      \cs_new:Npn
11539        \markdownInfo #1
11540        {
11541          \msg_info:nne
11542            { markdown }
11543            { generic-message }
11544            { #1 }
11545        }
11546    }
11547  \cs_if_free:NT
11548    \markdownWarning
11549    {
11550      \cs_new:Npn
11551        \markdownWarning #1
11552        {
11553          \msg_warning:nne
11554            { markdown }
11555            { generic-message }
11556            { #1 }
11557        }
11558    }
11559  \cs_if_free:NT
11560    \markdownError
11561    {

```

```

11562 \cs_new:Npn
11563   \markdownError #1 #2
11564   {
11565     \msg_error:nnee
11566       { markdown }
11567       { generic-message-with-help-text }
11568       { #1 }
11569       { #2 }
11570   }
11571 }
11572 \msg_new:nnn
11573   { markdown }
11574   { generic-message }
11575   { #1 }
11576 \msg_new:nnnn
11577   { markdown }
11578   { generic-message-with-help-text }
11579   { #1 }
11580   { #2 }
11581 \cs_generate_variant:Nn
11582   \msg_info:nnn
11583   { nne }
11584 \cs_generate_variant:Nn
11585   \msg_warning:nnn
11586   { nne }
11587 \cs_generate_variant:Nn
11588   \msg_error:nnnn
11589   { nnee }
11590 \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_EX themes provided with the Markdown package.

```

11591 \ExplSyntaxOn
11592 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
11593 \cs_new:Nn
11594   \@@_plain_tex_load_theme:nn
11595   {
11596     \prop_get:NnNTF
11597       \g_@@_plain_tex_loaded_themes_linenos_prop
11598       { #1 }
11599     \l_tmpa_tl
11600     {
11601       \msg_warning:nnnV
11602       { markdown }

```



```

11603     { repeatedly-loaded-plain-tex-theme }
11604     { #1 }
11605     \l_tmpa_tl
11606   }
11607   {
11608     \msg_info:nnn
11609     { markdown }
11610     { loading-plain-tex-theme }
11611     { #1 }
11612     \prop_gput:Nnx
11613     \g_@@_plain_tex_loaded_themes_linenos_prop
11614     { #1 }
11615     { \tex_the:D \tex_inputlineno:D }
11616     \file_input:n
11617     { markdown theme #2 }
11618   }
11619 }
11620 \msg_new:nnn
11621 { markdown }
11622 { loading-plain-tex-theme }
11623 { Loading~plain~TeX~Markdown~theme~#1 }
11624 \msg_new:nnn
11625 { markdown }
11626 { repeatedly-loaded-plain-tex-theme }
11627 {
11628   Plain~TeX~Markdown~theme~#1~was~previously~
11629   loaded~on~line~#2,~not~loading~it~again
11630 }
11631 \cs_generate_variant:Nn
11632 \prop_gput:Nnn
11633 { Nnx }
11634 \cs_gset_eq:NN
11635 \@@_load_theme:nn
11636 \@@_plain_tex_load_theme:nn
11637 \cs_generate_variant:Nn
11638 \@@_load_theme:nn
11639 { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T_EX theme from within themes for higher-level T_EX formats such as L^AT_EX and ConT_EXt.

```

11640 \cs_new:Npn
11641 \markdownLoadPlainTeXTheme
11642 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

11643   \tl_set:NV

```

```

11644     \l_tmpa_tl
11645     \g_@@_current_theme_tl
11646     \tl_reverse:N
11647     \l_tmpa_tl
11648     \tl_set:Ne
11649     \l_tmpb_tl
11650     {
11651         \tl_tail:V
11652         \l_tmpa_tl
11653     }
11654     \tl_reverse:N
11655     \l_tmpb_tl

```

Next, we munge the theme name.

```

11656     \str_set:NV
11657         \l_tmpa_str
11658         \l_tmpb_tl
11659     \str_replace_all:Nnn
11660         \l_tmpa_str
11661         { / }
11662         { _ }

```

Finally, we load the plain \TeX theme.

```

11663     \@@_plain_tex_load_theme:VV
11664         \l_tmpb_tl
11665         \l_tmpa_str
11666     }
11667 \cs_generate_variant:Nn
11668     \tl_set:Nn
11669     { Ne }
11670 \cs_generate_variant:Nn
11671     \@@_plain_tex_load_theme:nn
11672     { VV }
11673 \ExplSyntaxOff

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

11674 \markdownSetup {
11675     rendererPrototypes = {
11676         tilde = {~},
11677     },
11678 }

```

The `witiko/markdown/defaults` plain \TeX theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

11679 \def\markdownRendererInterblockSeparatorPrototype{\par}%
11680 \def\markdownRendererParagraphSeparatorPrototype{%
11681   \markdownRendererInterblockSeparator}%
11682 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
11683 \def\markdownRendererSoftLineBreakPrototype{ }%
11684 \let\markdownRendererEllipsisPrototype\dots
11685 \def\markdownRendererNbspPrototype{~}%
11686 \def\markdownRendererLeftBracePrototype{\char`\{}%
11687 \def\markdownRendererRightBracePrototype{\char`\}}%
11688 \def\markdownRendererDollarSignPrototype{\char`\$}%
11689 \def\markdownRendererPercentSignPrototype{\char`\}%
11690 \def\markdownRendererAmpersandPrototype{\&}%
11691 \def\markdownRendererUnderscorePrototype{\char`\_}%
11692 \def\markdownRendererHashPrototype{\char`\#}%
11693 \def\markdownRendererCircumflexPrototype{\char`\^}%
11694 \def\markdownRendererBackslashPrototype{\char`\}%
11695 \def\markdownRendererTildePrototype{\char`\~}%
11696 \def\markdownRendererPipePrototype{|}%
11697 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
11698 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
11699 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
11700   \markdownInput{#3}}%
11701 \def\markdownRendererContentBlockOnlineImagePrototype{%
11702   \markdownRendererImage}%
11703 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
11704   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
11705 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
11706 \def\markdownRendererULBeginPrototype{}%
11707 \def\markdownRendererULBeginTightPrototype{}%
11708 \def\markdownRendererULItemPrototype{}%
11709 \def\markdownRendererULItemEndPrototype{}%
11710 \def\markdownRendererULEndPrototype{}%
11711 \def\markdownRendererULEndTightPrototype{}%
11712 \def\markdownRendererOLBeginPrototype{}%
11713 \def\markdownRendererOLBeginTightPrototype{}%
11714 \def\markdownRendererFancyOLBeginPrototype#1#2{\markdownRendererOLBegin}%
11715 \def\markdownRendererFancyOLBeginTightPrototype#1#2{\markdownRendererOLBeginTight}%
11716 \def\markdownRendererOLItemPrototype{}%
11717 \def\markdownRendererOLItemWithNumberPrototype#1{}%
11718 \def\markdownRendererOLItemEndPrototype{}%
11719 \def\markdownRendererFancyOLItemPrototype{\markdownRendererOLItem}%
11720 \def\markdownRendererFancyOLItemWithNumberPrototype{\markdownRendererOLItemWithNumber}%
11721 \def\markdownRendererFancyOLItemEndPrototype{}%
11722 \def\markdownRendererOLEndPrototype{}%
11723 \def\markdownRendererOLEndTightPrototype{}%
11724 \def\markdownRendererFancyOLEndPrototype{\markdownRendererOLEnd}%
11725 \def\markdownRendererFancyOLEndTightPrototype{\markdownRendererOLEndTight}%

```

```

11726 \def\markdownRendererDlBeginPrototype{}%
11727 \def\markdownRendererDlBeginTightPrototype{}%
11728 \def\markdownRendererDlItemPrototype#1{#1}%
11729 \def\markdownRendererDlItemEndPrototype{}%
11730 \def\markdownRendererDlDefinitionBeginPrototype{}%
11731 \def\markdownRendererDlDefinitionEndPrototype{\par}%
11732 \def\markdownRendererDlEndPrototype{}%
11733 \def\markdownRendererDlEndTightPrototype{}%
11734 \def\markdownRendererEmphasisPrototype#1{\it#1}%
11735 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
11736 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
11737 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
11738 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=Opt}%
11739 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
11740 \def\markdownRendererInputVerbatimPrototype#1{%
11741   \par{\tt\input#1\relax{}}\par}%
11742 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
11743   \markdownRendererInputVerbatim{#1}}%
11744 \def\markdownRendererHeadingOnePrototype#1{#1}%
11745 \def\markdownRendererHeadingTwoPrototype#1{#1}%
11746 \def\markdownRendererHeadingThreePrototype#1{#1}%
11747 \def\markdownRendererHeadingFourPrototype#1{#1}%
11748 \def\markdownRendererHeadingFivePrototype#1{#1}%
11749 \def\markdownRendererHeadingSixPrototype#1{#1}%
11750 \def\markdownRendererThematicBreakPrototype{}%
11751 \def\markdownRendererNotePrototype#1{#1}%
11752 \def\markdownRendererCitePrototype#1{}%
11753 \def\markdownRendererTextCitePrototype#1{}%
11754 \def\markdownRendererTickedBoxPrototype{[X]}%
11755 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
11756 \def\markdownRendererUntickedBoxPrototype{[ ]}%
11757 \def\markdownRendererStrikeThroughPrototype#1{#1}%
11758 \def\markdownRendererSuperscriptPrototype#1{#1}%
11759 \def\markdownRendererSubscriptPrototype#1{#1}%
11760 \def\markdownRendererDisplayMathPrototype#1{##1$$$}%
11761 \def\markdownRendererInlineMathPrototype#1{##1$}%
11762 \ExplSyntaxOn
11763 \cs_gset:Npn
11764   \markdownRendererHeaderAttributeContextBeginPrototype
11765   {
11766     \group_begin:
11767     \color_group_begin:
11768   }
11769 \cs_gset:Npn
11770   \markdownRendererHeaderAttributeContextEndPrototype
11771   {
11772     \color_group_end:

```

```

11773     \group_end:
11774   }
11775   \cs_gset_eq:NN
11776     \markdownRendererBracketedSpanAttributeContextBeginPrototype
11777     \markdownRendererHeaderAttributeContextBeginPrototype
11778   \cs_gset_eq:NN
11779     \markdownRendererBracketedSpanAttributeContextEndPrototype
11780     \markdownRendererHeaderAttributeContextEndPrototype
11781   \cs_gset_eq:NN
11782     \markdownRendererFencedDivAttributeContextBeginPrototype
11783     \markdownRendererHeaderAttributeContextBeginPrototype
11784   \cs_gset_eq:NN
11785     \markdownRendererFencedDivAttributeContextEndPrototype
11786     \markdownRendererHeaderAttributeContextEndPrototype
11787   \cs_gset_eq:NN
11788     \markdownRendererFencedCodeAttributeContextBeginPrototype
11789     \markdownRendererHeaderAttributeContextBeginPrototype
11790   \cs_gset_eq:NN
11791     \markdownRendererFencedCodeAttributeContextEndPrototype
11792     \markdownRendererHeaderAttributeContextEndPrototype
11793   \cs_gset:Npn
11794     \markdownRendererReplacementCharacterPrototype
11795     { \codepoint_str_generate:n { fffd } }
11796   \ExplSyntaxOff
11797   \def\markdownRendererSectionBeginPrototype{}%
11798   \def\markdownRendererSectionEndPrototype{}%

```

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.

```

11799   \ExplSyntaxOn
11800   \cs_new:Nn
11801     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11802     {
11803       \str_case:nn
11804         { #2 }
11805         {
11806           { md } { \markdownInput{#1} }
11807           { tex } { \markdownEscape{#1} \unskip }
11808         }
11809     }
11810   \cs_new:Nn
11811     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11812     {
11813       \str_case:nn

```

```

11814     { #2 }
11815     {
11816     { md } { \markdownInput{#1} }
11817     { tex } { \markdownEscape{#1} }
11818     }
11819   }
11820 \cs_gset:Npn
11821   \markdownRendererInputRawInlinePrototype#1#2
11822   {
11823     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11824     { #1 }
11825     { #2 }
11826   }
11827 \cs_gset:Npn
11828   \markdownRendererInputRawBlockPrototype#1#2
11829   {
11830     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11831     { #1 }
11832     { #2 }
11833   }
11834 \ExplSyntaxOff

```

3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position p :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth p .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth p .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth p .

```

11835 \ExplSyntaxOn
11836 \seq_new:N \g_@@_jekyll_data_datatypes_seq
11837 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
11838 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
11839 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```

11840 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
11841 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
11842 {
11843   \seq_if_empty:NF
11844     \g_@@_jekyll_data_datatypes_seq
11845     {
11846       \seq_get_right:NN
11847         \g_@@_jekyll_data_datatypes_seq
11848         \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.

```

11849   \str_if_eq:NNTF
11850     \l_tmpa_tl
11851     \c_@@_jekyll_data_sequence_tl
11852     {
11853       \seq_put_right:Nn
11854         \g_@@_jekyll_data_wildcard_absolute_address_seq
11855         { * }
11856     }
11857     {
11858       \seq_put_right:Nn
11859         \g_@@_jekyll_data_wildcard_absolute_address_seq
11860         { #1 }
11861     }
11862   }
11863 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{{person: {name: Elon, surname: Musk}}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```
11864 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
11865 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
11866 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
11867 {
11868   \seq_pop_left:NN #1 \l_tmpa_tl
11869   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
11870   \seq_put_left:NV #1 \l_tmpa_tl
11871 }
11872 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
11873 {
11874   \markdown_jekyll_data_concatenate_address:NN
11875   \g_@@_jekyll_data_wildcard_absolute_address_seq
11876   \g_@@_jekyll_data_wildcard_absolute_address_tl
11877   \seq_get_right:NN
11878   \g_@@_jekyll_data_wildcard_absolute_address_seq
11879   \g_@@_jekyll_data_wildcard_relative_address_tl
11880 }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```
11881 \cs_new:Nn \markdown_jekyll_data_push:nN
11882 {
11883   \markdown_jekyll_data_push_address_segment:n
11884   { #1 }
11885   \seq_put_right:NV
11886   \g_@@_jekyll_data_datatypes_seq
11887   #2
11888   \markdown_jekyll_data_update_address_tls:
11889 }
11890 \cs_new:Nn \markdown_jekyll_data_pop:
11891 {
11892   \seq_pop_right:NN
11893   \g_@@_jekyll_data_wildcard_absolute_address_seq
11894   \l_tmpa_tl
11895   \seq_pop_right:NN
11896   \g_@@_jekyll_data_datatypes_seq
11897   \l_tmpa_tl
```



```

11898     \markdown_jekyll_data_update_address_tls:
11899   }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

11900 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
11901   {
11902     \keys_set_known:nn
11903       { markdown/jekyllData }
11904       { { #1 } = { #2 } }
11905   }
11906 \cs_generate_variant:Nn
11907   \markdown_jekyll_data_set_keyval:nn
11908   { Vn }
11909 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
11910   {
11911     \markdown_jekyll_data_push:nN
11912       { #1 }
11913     \c_@@_jekyll_data_scalar_tl
11914     \markdown_jekyll_data_set_keyval:Vn
11915     \g_@@_jekyll_data_wildcard_absolute_address_tl
11916     { #2 }
11917     \markdown_jekyll_data_set_keyval:Vn
11918     \g_@@_jekyll_data_wildcard_relative_address_tl
11919     { #2 }
11920     \markdown_jekyll_data_pop:
11921   }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

11922 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
11923   \markdown_jekyll_data_push:nN
11924     { #1 }
11925   \c_@@_jekyll_data_sequence_tl
11926 }
11927 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
11928   \markdown_jekyll_data_push:nN
11929     { #1 }
11930   \c_@@_jekyll_data_mapping_tl
11931 }
11932 \def\markdownRendererJekyllDataSequenceEndPrototype{
11933   \markdown_jekyll_data_pop:
11934 }
11935 \def\markdownRendererJekyllDataMappingEndPrototype{
11936   \markdown_jekyll_data_pop:
11937 }
11938 \def\markdownRendererJekyllDataBooleanPrototype#1#2{

```

```

11939 \markdown_jekyll_data_set_keyvals:nn
11940   { #1 }
11941   { #2 }
11942 }
11943 \def\markdownRendererJekyllDataEmptyPrototype#1{}
11944 \def\markdownRendererJekyllDataNumberPrototype#1#2{
11945   \markdown_jekyll_data_set_keyvals:nn
11946   { #1 }
11947   { #2 }
11948 }
11949 \def\markdownRendererJekyllDataStringPrototype#1#2{
11950   \markdown_jekyll_data_set_keyvals:nn
11951   { #1 }
11952   { #2 }
11953 }
11954 \ExplSyntaxOff

```

If plain \TeX is the top layer, we load the [witiko/markdown/defaults](#) plain \TeX theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

11955 \ExplSyntaxOn
11956 \str_if_eq:VVT
11957   \c_@@_top_layer_tl
11958   \c_@@_option_layer_plain_tex_tl
11959   {
11960     \ExplSyntaxOff
11961     \@@_if_option:nF
11962       { noDefaults }
11963       {
11964         \@@_setup:n
11965           {theme = witiko/markdown/defaults}
11966       }
11967     \ExplSyntaxOn
11968   }
11969 \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).

```

11970 \ExplSyntaxOn
11971 \tl_new:N \g_@@_formatted_lua_options_tl
11972 \cs_new:Nn \@@_format_lua_options:
11973   {
11974     \tl_gclear:N
11975     \g_@@_formatted_lua_options_tl

```

```

11976     \seq_map_function:NN
11977     \g_@@_lua_options_seq
11978     \@@_format_lua_option:n
11979   }
11980 \cs_new:Nn \@@_format_lua_option:n
11981 {
11982   \@@_typecheck_option:n
11983   { #1 }
11984   \@@_get_option_type:nN
11985   { #1 }
11986   \l_tmpa_tl
11987   \bool_case_true:nF
11988   {
11989     {
11990       \str_if_eq_p:VV
11991       \l_tmpa_tl
11992       \c_@@_option_type_boolean_tl ||
11993       \str_if_eq_p:VV
11994       \l_tmpa_tl
11995       \c_@@_option_type_number_tl ||
11996       \str_if_eq_p:VV
11997       \l_tmpa_tl
11998       \c_@@_option_type_counter_tl
11999     }
12000     {
12001       \@@_get_option_value:nN
12002       { #1 }
12003       \l_tmpa_tl
12004       \tl_gput_right:Nx
12005       \g_@@_formatted_lua_options_tl
12006       { #1~::~ \l_tmpa_tl ,~ }
12007     }
12008   }
12009   \str_if_eq_p:VV
12010   \l_tmpa_tl
12011   \c_@@_option_type_clist_tl
12012 }
12013 {
12014   \@@_get_option_value:nN
12015   { #1 }
12016   \l_tmpa_tl
12017   \tl_gput_right:Nx
12018   \g_@@_formatted_lua_options_tl
12019   { #1~::~\c_left_brace_str }
12020   \clist_map_inline:Vn
12021   \l_tmpa_tl
12022   {

```

```

12023         \tl_gput_right:Nx
12024         \g_@@_formatted_lua_options_tl
12025         { "##1" ,~ }
12026     }
12027     \tl_gput_right:Nx
12028     \g_@@_formatted_lua_options_tl
12029     { \c_right_brace_str ,~ }
12030 }
12031 }
12032 {
12033     \@@_get_option_value:nN
12034     { #1 }
12035     \l_tmpa_tl
12036     \tl_gput_right:Nx
12037     \g_@@_formatted_lua_options_tl
12038     { #1~::~ " \l_tmpa_tl " ,~ }
12039 }
12040 }
12041 \cs_generate_variant:Nn
12042 \clist_map_inline:nn
12043 { Vn }
12044 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
12045 \def\markdownLuaOptions{ \g_@@_formatted_lua_options_tl }
12046 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain \TeX . It exposes the `convert` function for the use by any further Lua code.

```
12047 \def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```

12048     local lfs = require("lfs")
12049     local cacheDir = "\markdownOptionCacheDir"
12050     if not lfs.isdir(cacheDir) then
12051         assert(lfs.mkdir(cacheDir))
12052     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.

```

12053     local md = require("markdown")
12054     local convert = md.new(\markdownLuaOptions)
12055 }%

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain \TeX .

```
12056 \def\markdownCleanup{%
```

Remove the `options.cacheDir` directory if it is empty.

```
12057     lfs.rmdir(cacheDir)
```

12058 }%

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.

```
12059 \csname newread\endcsname\markdownInputFileStream
12060 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
12061 \begingroup
12062 \catcode\^^I=12%
12063 \gdef\markdownReadAndConvertTab{^^I}%
12064 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the $\text{\LaTeX} 2_{\epsilon}$ `\filecontents` macro to plain \TeX .

```
12065 \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.

```
12066 \catcode\^^M=13%
12067 \catcode\^^I=13%
12068 \catcode|=0%
12069 \catcode\=12%
12070 |catcode@=14%
12071 |catcode|=12@
12072 |gdef|markdownReadAndConvert#1#2{@
12073 |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```
12074 |markdownIfOption{frozenCache}{@
12075 |immediate|openout|markdownOutputFileStream@
12076 |markdownOptionInputTempFileName|relax@
12077 |markdownInfo{@
12078 |Buffering block-level markdown input into the temporary @
12079 |input file "|markdownOptionInputTempFileName" and scanning @
12080 |for the closing token sequence "#1"@
12081 }@
```

Locally change the category of the special plain \TeX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

12082     |def|do##1{|catcode`##1=12}|dospecials@
12083     |catcode`|=12@
12084     |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 (\sim M) are produced.

```

12085     |def|markdownReadAndConvertStripPercentSign##1{@
12086         |markdownIfOption{stripPercentSigns}{@
12087             |if##1%@
12088                 |expandafter|expandafter|expandafter@
12089                 |markdownReadAndConvertProcessLine@
12090             |else@
12091                 |expandafter|expandafter|expandafter@
12092                 |markdownReadAndConvertProcessLine@
12093                 |expandafter|expandafter|expandafter##1@
12094             |fi@
12095         }{@
12096             |expandafter@
12097             |markdownReadAndConvertProcessLine@
12098             |expandafter##1@
12099         }@
12100     }@

```

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 (\sim M) are produced.

```

12101     |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

12102         |ifx|relax##3|relax@
12103         |markdownIfOption{frozenCache}{}{@
12104             |immediate|write|markdownOutputFileStream{##1}@
12105         }@
12106         |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_EX, `\input` the result of the conversion, and expand the ending control sequence.

```

12107         |def~M{@
12108             |markdownInfo{The ending token sequence was found}@
12109             |markdownIfOption{frozenCache}{}{@

```

```

12110         |immediate|closeout|markdownOutputFileStream@
12111     }@
12112     |endgroup@
12113     |markdownInput{@
12114         |markdownOptionOutputDir@
12115         /|markdownOptionInputTempFileName@
12116     }@
12117     #2}@
12118     |fi@

```

Repeat with the next line.

```
12119     ^^M}@
```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

12120     |catcode`|^I=13@
12121     |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.

```

12122     |catcode`|^M=13@
12123     |def^M##1^M{@
12124         |def^M###1^M{@
12125             |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
12126         ^M}@
12127     ^M}@

```

Reset the character categories back to the former state.

```
12128 |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.

```

12129 \ExplSyntaxOn
12130 \cs_new:Npn
12131   \markdownLuaExecute
12132   #1
12133   {
12134     \int_compare:nNt
12135       { \g_luabridge_method_int }
12136       =
12137       { \c_luabridge_method_shell_int }
12138       {
12139         \sys_if_shell_unrestricted:F
12140         {
12141           \sys_if_shell:TF
12142           {
12143             \msg_error:nn
12144               { markdown }

```

```

12145             { restricted-shell-access }
12146         }
12147     {
12148         \msg_error:nn
12149         { markdown }
12150         { disabled-shell-access }
12151     }
12152 }
12153 }
12154 \luabridge_now:e
12155 { #1 }
12156 }
12157 \cs_generate_variant:Nn
12158 \msg_new:nnnn
12159 { nnnV }
12160 \tl_set:Nn
12161 \l_tmpa_tl
12162 {
12163     You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
12164     --enable-write18~flag,~or~write~shell_escape=t~in~the~
12165     texmf.cnf~file.
12166 }
12167 \msg_new:nnnV
12168 { markdown }
12169 { restricted-shell-access }
12170 { Shell-escape-is-restricted }
12171 \l_tmpa_tl
12172 \msg_new:nnnV
12173 { markdown }
12174 { disabled-shell-access }
12175 { Shell-escape-is-disabled }
12176 \l_tmpa_tl
12177 \ExplSyntaxOff

```

3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

12178 \ExplSyntaxOn
12179 \tl_new:N
12180 \g_@@_after_markinline_tl
12181 \tl_gset:Nn
12182 \g_@@_after_markinline_tl
12183 { \unskip }
12184 \cs_new:Npn
12185 \markinline
12186 {

```


Locally change the category of the special plain T_EX characters to *other* in order to prevent unwanted interpretation of the input markdown text as T_EX code.

```
12187 \group_begin:
12188 \cctab_select:N
12189 \c_other_cctab
```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```
12190 \@@_if_option:nF
12191 { frozenCache }
12192 {
12193 \immediate
12194 \openout
12195 \markdownOutputFileStream
12196 \markdownOptionInputTempFileName
12197 \relax
12198 \msg_info:nne
12199 { markdown }
12200 { buffering-markinline }
12201 { \markdownOptionInputTempFileName }
12202 }
```

Peek ahead and extract the inline markdown text.

```
12203 \peek_regex_replace_once:nnF
12204 { { (.*) } }
12205 {
```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
12206 \c { @@_if_option:nF }
12207 \cB { frozenCache \cE }
12208 \cB {
12209 \c { immediate }
12210 \c { write }
12211 \c { markdownOutputFileStream }
12212 \cB { \1 \cE }
12213 \c { immediate }
12214 \c { closeout }
12215 \c { markdownOutputFileStream }
12216 \cE }
```

Reset the category codes and `\input` the result of the conversion.

```
12217 \c { group_end: }
12218 \c { group_begin: }
12219 \c { @@_setup:n }
12220 \cB { contentLevel = inline \cE }
12221 \c { markdownInput }
12222 \cB {
```

```

12223         \c { markdownOptionOutputDir } /
12224         \c { markdownOptionInputTempFileName }
12225     \cE }
12226     \c { group_end: }
12227     \c { tl_use:N }
12228     \c { g_@@_after_markinline_tl }
12229 }
12230 {
12231     \msg_error:nn
12232     { markdown }
12233     { markinline-peek-failure }
12234     \group_end:
12235     \tl_use:N
12236     \g_@@_after_markinline_tl
12237 }
12238 }
12239 \msg_new:nnn
12240 { markdown }
12241 { buffering-markinline }
12242 { Buffering~inline~markdown~input~into~the~temporary~input~file~"#1". }
12243 \msg_new:nnnn
12244 { markdown }
12245 { markinline-peek-failure }
12246 { Use~of~\iow_char:N \\ markinline~doesn't~match~its~definition }
12247 { The~macro~should~be~followed~by~inline~markdown~text~in~curly~braces }
12248 \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 TeX.

```
12249 \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.

```

12250 \catcode`|=0%
12251 \catcode`\|=12%
12252 \catcode`|&=6%
12253 |gdef|markdownInput#1{|%

```

Change the category code of the percent sign (%) to other, so that a user of the `hybrid` Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

12254 |begingroup
12255 |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.

```
12256      |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache⟨number⟩` macro, and increment `frozenCacheCounter`.

```
12257      |markdownIfOption{frozenCache}{%
12258          |ifnum|markdownOptionFrozenCacheCounter=0|relax
12259              |markdownInfo{Reading frozen cache from
12260                  "|markdownOptionFrozenCacheFileName"}%
12261              |input|markdownOptionFrozenCacheFileName|relax
12262          |fi
12263          |markdownInfo{Including markdown document number
12264              "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
12265          |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
12266          |global|advance|markdownOptionFrozenCacheCounter by 1|relax
12267      }{%
12268          |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L^AT_EX_Mk to track changes to the markdown document.

```
12269      |openin|markdownInputFileStream&1
12270      |closein|markdownInputFileStream
12271      |markdownPrepareLuaOptions
12272      |markdownLuaExecute{%
12273          |markdownPrepare
12274          local file = assert(io.open("&1", "r"),
12275              [[Could not open file "&1" for reading]])
12276          local input = assert(file:read("*a"))
12277          assert(file:close())
12278          print(convert(input))
12279      |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
12280      |markdownIfOption{finalizeCache}{%
12281          |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
12282      }%
12283      |endgroup
12284      }%
12285      |endgroup
```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of T_EX to execute a T_EX document in the middle of a markdown document fragment.

```

12286 \gdef\markdownEscape#1{%
12287   \catcode`\%=14\relax
12288   \catcode`\#=6\relax
12289   \input #1\relax
12290   \catcode`\%=12\relax
12291   \catcode`\#=12\relax
12292 }%

```

3.3 L^AT_EX Implementation

The L^AT_EX implemenation makes use of the fact that, apart from some subtle differences, L^AT_EX implements the majority of the plain T_EX format [12, Section 9]. As a consequence, we can directly reuse the existing plain T_EX implementation.

```

12293 \def\markdownVersionSpace{ }%
12294 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
12295   \markdownVersion\markdownVersionSpace markdown renderer]%

```

3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain T_EX implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the L^AT_EX interface (see Section 2.3.2).

```

12296 \ExplSyntaxOn
12297 \cs_gset_eq:NN
12298   \markinlinePlainTeX
12299   \markinline
12300 \cs_gset:Npn
12301   \markinline
12302   {
12303     \peek_regex_replace_once:nn
12304       { ( \[ (.*) \] ) ? }
12305       {

```

Apply the options locally.

```

12306         \c { group_begin: }
12307         \c { @@_setup:n }
12308         \cB { \2 \cE }
12309         \c { tl_put_right:Nn }
12310         \c { g_@@_after_markinline_tl }
12311         \cB { \c { group_end: } \cE }
12312         \c { markinlinePlainTeX }
12313       }
12314   }
12315 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain \TeX implementation of the `\markdownInput` macro. The `\markdownInput` macro is then redefined to accept an optional argument with options recognized by the \LaTeX interface (see Section 2.3.2).

```
12316 \let\markdownInputPlainTeX\markdownInput
12317 \renewcommand\markdownInput[2][ ]{%
12318   \begingroup
12319     \markdownSetup{#1}%
12320     \markdownInputPlainTeX{#2}%
12321   \endgroup}%
```

The `markdown`, and `markdown*` \LaTeX environments are implemented using the `\markdownReadAndConvert` macro.

```
12322 \ExplSyntaxOn
12323 \renewenvironment
12324   { markdown }
12325   {
```

In our implementation of the `markdown` \LaTeX 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 \LaTeX 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` \LaTeX 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 \TeX via the `\endlinechar` plain \TeX macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
12326   \group_begin:
12327   \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).

```
12328   \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
12329 \peek_regex_replace_once:nmF
12330   { \ *\[r*([~]*)\[^\r]* }
12331   {
```

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.

```
12332     \c { group_end: }
12333     \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
12334     \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 `LATEX` environment.

```
12335     \c { markdownReadAndConvert@markdown } { }
12336   }
12337   {
12338     \group_end:
12339     \markdownReadAndConvert@markdown { }
12340   }
12341 }
12342 { \markdownEnd }
12343 \renewenvironment
12344 { markdown* }
12345 [ 1 ]
12346 {
12347   \msg_warning:nnn
12348   { markdown }
12349   { latex-markdown-star-deprecated }
12350   { #1 }
12351   \@@_setup:n
12352   { #1 }
12353   \markdownReadAndConvert@markdown *
12354 }
12355 { \markdownEnd }
12356 \msg_new:nnn
12357 { markdown }
12358 { latex-markdown-star-deprecated }
12359 {
12360   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
12361   be~removed~in~the~next~major~version~of~the~Markdown~package.
```

```

12362 }
12363 \ExplSyntaxOff
12364 \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*.

```

12365 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
12366 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
12367 \gdef|markdownReadAndConvert@markdown#1<%
12368 |markdownReadAndConvert<\end{markdown#1}>%
12369 <|end<markdown#1>>%
12370 |endgroup

```

3.3.2 Options

The supplied package options are processed using the `markdownSetup` macro.

```

12371 \DeclareOption*{%
12372 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
12373 \ProcessOptions\relax

```

3.3.3 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 L^AT_EX themes provided with the Markdown package.

```

12374 \ExplSyntaxOn
12375 \cs_gset:Nn
12376 \@@_load_theme:nn
12377 {

```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

12378 \ifmarkdownLaTeXLoaded
12379 \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load L^AT_EX themes after the preamble. Otherwise, try loading a plain TeX theme instead.

```

12380 \file_if_exist:nTF
12381 { markdown theme #2.sty }
12382 {
12383 \msg_error:nnn
12384 { markdown }
12385 { latex-theme-after-preamble }

```

```

12386         { #1 }
12387     }
12388     {
12389         \@_plain_tex_load_theme:nn
12390         { #1 }
12391         { #2 }
12392     }
12393     \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L^AT_EX theme if it exists or load a plain T_EX theme otherwise.

```

12394     \file_if_exist:nTF
12395     { markdown theme #2.sty }
12396     {
12397         \msg_info:nnn
12398         { markdown }
12399         { loading-latex-theme }
12400         { #1 }
12401         \RequirePackage
12402         { markdown theme #2 }
12403     }
12404     {
12405         \@_plain_tex_load_theme:nn
12406         { #1 }
12407         { #2 }
12408     }
12409     \fi
12410     \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

12411     \msg_info:nnn
12412     { markdown }
12413     { theme-loading-postponed }
12414     { #1 }
12415     \AtEndOfPackage
12416     {
12417         \@_load_theme:nn
12418         { #1 }
12419         { #2 }
12420     }
12421     \fi
12422 }
12423 \msg_new:nnn
12424 { markdown }
12425 { theme-loading-postponed }
12426 {
12427     Postponing~loading~Markdown~theme~#1~until~

```



```

12428     Markdown~package~has~finished~loading
12429   }
12430   \msg_new:nnn
12431     { markdown }
12432     { loading-latex-theme }
12433     { Loading~LaTeX~Markdown~theme~#1 }
12434   \cs_generate_variant:Nn
12435     \msg_new:nnnn
12436     { nnVV }
12437   \tl_set:Nn
12438     \l_tmpa_tl
12439     { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
12440   \tl_put_right:NV
12441     \l_tmpa_tl
12442     \c_backslash_str
12443   \tl_put_right:Nn
12444     \l_tmpa_tl
12445     { begin{document} }
12446   \tl_set:Nn
12447     \l_tmpb_tl
12448     { Load~Markdown~theme~#1~before~ }
12449   \tl_put_right:NV
12450     \l_tmpb_tl
12451     \c_backslash_str
12452   \tl_put_right:Nn
12453     \l_tmpb_tl
12454     { begin{document} }
12455   \msg_new:nnVV
12456     { markdown }
12457     { latex-theme-after-preamble }
12458     \l_tmpa_tl
12459     \l_tmpb_tl
12460   \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
12461 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
12462 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
12463 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
12464   \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_EX option is disabled and the code block has not been previously typeset:

```

12465 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
12466   \def\next##1 ##2\relax{%
12467     \ifthenelse{\equal{##1}{dot}}{%
12468       \markdownIfOption{frozenCache}{}{%
12469         \immediate\write18{%
12470           if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
12471           then
12472             dot -Tpdf -o #1.pdf #1;
12473             cp #1 #1.pdf.source;
12474             fi}}%

```

We include the typeset image using the image token renderer:

```

12475   \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

12476   }{%
12477     \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}{#3}%
12478   }%
12479 }%
12480 \next#2 \relax}%

```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```

12481 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
12482 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also [Section 1.1.3](#):

```

12483 \RequirePackage{catchfile,grffile}

```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```

12484 \newcount\markdown@witiko@graphicx@http@counter
12485 \markdown@witiko@graphicx@http@counter=0
12486 \newcommand\markdown@witiko@graphicx@http@filename{%
12487   \markdownOptionCacheDir/witiko_graphicx_http%
12488   .\the\markdown@witiko@graphicx@http@counter}%

```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```

12489 \newcommand\markdown@witiko@graphicx@http@download[2]{%
12490   wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}

```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```

12491 \begingroup
12492 \catcode`\%=12
12493 \catcode`\^^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

12494 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
12495   \begingroup
12496     \edef\filename{\markdown@witiko@graphicx@http@filename}^^A

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain `TeX` option is disabled:

```

12497   \markdownIfOption{frozenCache}{-}{^^A
12498     \immediate\write18{^^A
12499       mkdir -p "\markdownOptionCacheDir";
12500       if printf '%s' "#3" | grep -q -E '^https?:';
12501       then

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

12502         OUTPUT_PREFIX="\markdownOptionCacheDir";
12503         OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
12504         OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/)";
12505         OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

12506         if ! [ -e "$OUTPUT" ];
12507         then
12508           \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
12509           printf '%s' "$OUTPUT" > "\filename";
12510         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:

```

12511         else
12512           printf '%s' '#3' > "\filename";
12513         fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

12514     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
12515     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
12516     {#1}{#2}{\filename}{#4}^^A
12517   \endgroup
12518   \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
12519 \endgroup

```

The `witiko/markdown/defaults` `LATEX` theme provides default definitions for token renderer prototypes. First, the `LATEX` theme loads the plain `TeX` theme with the default definitions for plain `TeX`:

12520 `\markdownLoadPlainTeXTheme`

Next, the L^AT_EX theme overrides some of the plain T_EX definitions. See Section 3.3.4 for the actual definitions.

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.

```
12521 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the `paralist` package.

```
12522 \@ifclassloaded{beamer}{}{%
12523   \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
12524   \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
12525 }
```

If we loaded the `paralist` package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```
12526 \ExplSyntaxOn
12527 \@ifpackageloaded{paralist}{
12528   \tl_new:N
12529     \l_@@_latex_fancy_list_item_label_number_style_tl
12530   \tl_new:N
12531     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12532   \cs_new:Nn
12533     \@@_latex_fancy_list_item_label_number:nn
12534     {
12535       \str_case:nn
12536         { #1 }
12537         {
12538           { Decimal } { #2 }
12539           { LowerRoman } { \int_to_roman:n { #2 } }
12540           { UpperRoman } { \int_to_Roman:n { #2 } }
12541           { LowerAlpha } { \int_to_alph:n { #2 } }
12542           { UpperAlpha } { \int_to_Alph:n { #2 } }
12543         }
12544     }
12545   \cs_new:Nn
12546     \@@_latex_fancy_list_item_label_delimiter:n
12547     {
12548       \str_case:nn
12549         { #1 }
12550         {
12551           { Default } { . }
12552           { OneParen } { ) }

```

```

12553         { Period } { . }
12554     }
12555 }
12556 \cs_new:Nn
12557   \@@_latex_fancy_list_item_label:nnn
12558   {
12559     \@@_latex_fancy_list_item_label_number:nn
12560     { #1 }
12561     { #3 }
12562     \@@_latex_fancy_list_item_label_delimiter:n
12563     { #2 }
12564   }
12565 \cs_new:Nn
12566   \@@_latex_paralist_style:nn
12567   {
12568     \str_case:nn
12569       { #1 }
12570       {
12571         { Decimal } { 1 }
12572         { LowerRoman } { i }
12573         { UpperRoman } { I }
12574         { LowerAlpha } { a }
12575         { UpperAlpha } { A }
12576       }
12577     \@@_latex_fancy_list_item_label_delimiter:n
12578     { #2 }
12579   }
12580 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

12581   ulBeginTight = {%
12582     \group_begin:
12583     \pltopsep=\topsep
12584     \plpartopsep=\partopsep
12585     \begin{compactitem}
12586   },
12587   ulEndTight = {
12588     \end{compactitem}
12589     \group_end:
12590   },
12591   fancyOlBegin = {
12592     \group_begin:
12593     \tl_set:Nn
12594       \l_@@_latex_fancy_list_item_label_number_style_tl
12595       { #1 }
12596     \tl_set:Nn

```

```

12597     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12598     { #2 }
12599   \@@_if_option:nTF
12600     { startNumber }
12601     {
12602       \tl_set:Nn
12603         \l_tmpa_tl
12604         { \begin{enumerate} }
12605     }
12606     {
12607       \tl_set:Nn
12608         \l_tmpa_tl
12609         { \begin{enumerate}[ ] }
12610       \tl_put_right:Nx
12611         \l_tmpa_tl
12612         { \@@_latex_paralist_style:nm { #1 } { #2 } }
12613       \tl_put_right:Nn
12614         \l_tmpa_tl
12615         { ] }
12616     }
12617   \tl_use:N
12618     \l_tmpa_tl
12619 },
12620 fancyO1End = {
12621   \end{enumerate}
12622   \group_end:
12623 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

12624   olBeginTight = {%
12625     \group_begin:
12626     \plpartopsep=\partopsep
12627     \pltopsep=\topsep
12628     \begin{compactenum}
12629   },
12630   olEndTight = {
12631     \end{compactenum}
12632     \group_end:
12633   },
12634   fancyO1BeginTight = {
12635     \group_begin:
12636     \tl_set:Nn
12637       \l_@@_latex_fancy_list_item_label_number_style_tl
12638       { #1 }
12639     \tl_set:Nn
12640       \l_@@_latex_fancy_list_item_label_delimiter_style_tl

```

```

12641     { #2 }
12642 \tl_set:Nn
12643   \l_tmpa_tl
12644   {
12645     \plpartopsep=\partopsep
12646     \pltopsep=\topsep
12647   }
12648 @@_if_option:nTF
12649   { startNumber }
12650   {
12651     \tl_put_right:Nn
12652       \l_tmpa_tl
12653       { \begin{compactenum} }
12654   }
12655   {
12656     \tl_put_right:Nn
12657       \l_tmpa_tl
12658       { \begin{compactenum}[ ] }
12659     \tl_put_right:Nx
12660       \l_tmpa_tl
12661       { @@_latex_paralist_style:nn { #1 } { #2 } }
12662     \tl_put_right:Nn
12663       \l_tmpa_tl
12664       { ] }
12665   }
12666 \tl_use:N
12667   \l_tmpa_tl
12668 },
12669 fancyO1EndTight = {
12670   \end{compactenum}
12671   \group_end:
12672 },
12673 fancyO1ItemWithNumber = {
12674   \item
12675   [
12676     @@_latex_fancy_list_item_label:VVn
12677     \l_@@_latex_fancy_list_item_label_number_style_tl
12678     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12679     { #1 }
12680   ]
12681 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

12682 dlBeginTight = {
12683   \group_begin:
12684   \plpartopsep=\partopsep

```

```

12685     \pltopsep=\topsep
12686     \begin{compactdesc}
12687   },
12688   dlEndTight = {
12689     \end{compactdesc}
12690     \group_end:
12691   }}}
12692   \cs_generate_variant:Nn
12693     \@_latex_fancy_list_item_label:nnn
12694     { VVn }
12695 }{
12696   \markdownSetup{rendererPrototypes={
12697     ulBeginTight = {\markdownRendererUlBegin},
12698     ulEndTight = {\markdownRendererUlEnd},
12699     fancyOlBegin = {\markdownRendererOlBegin},
12700     fancyOlEnd = {\markdownRendererOlEnd},
12701     olBeginTight = {\markdownRendererOlBegin},
12702     olEndTight = {\markdownRendererOlEnd},
12703     fancyOlBeginTight = {\markdownRendererOlBegin},
12704     fancyOlEndTight = {\markdownRendererOlEnd},
12705     dlBeginTight = {\markdownRendererDlBegin},
12706     dlEndTight = {\markdownRendererDlEnd}}}
12707 }
12708 \ExplSyntaxOff
12709 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

12710 \ifpackageloaded{unicode-math}{
12711   \markdownSetup{rendererPrototypes={
12712     untickedBox = {\mdlgwhtsquare$},
12713   }}
12714 }{
12715   \RequirePackage{amssymb}
12716   \markdownSetup{rendererPrototypes={
12717     untickedBox = {\square$},
12718   }}
12719 }
12720 \RequirePackage{csvsimple}
12721 \RequirePackage{fancyvrb}
12722 \RequirePackage{graphicx}
12723 \markdownSetup{rendererPrototypes={
12724   hardLineBreak = {\},
12725   leftBrace = {\textbraceleft},
12726   rightBrace = {\textbraceright},
12727   dollarSign = {\textdollar},
12728   underscore = {\textunderscore},

```



```

12729   circumflex = {\textasciicircum},
12730   backslash = {\textbackslash},
12731   tilde = {\textasciitilde},
12732   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T_EX 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,³⁴ we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

12733   codeSpan = {%
12734     \ifmmode
12735       \text{#1}%
12736     \else
12737       \texttt{#1}%
12738     \fi
12739   }}
12740 \ExplSyntaxOn
12741 \markdownSetup{
12742   rendererPrototypes = {
12743     contentBlock = {
12744       \str_case:nnF
12745         { #1 }
12746         {
12747           { csv }
12748           {
12749             \begin{table}
12750               \begin{center}
12751                 \csvautotabular{#3}
12752               \end{center}
12753               \tl_if_empty:nF
12754                 { #4 }
12755                 { \caption{#4} }
12756             \end{table}
12757           }
12758           { tex } { \markdownEscape{#3} }
12759         }
12760         { \markdownInput{#3} }
12761       },
12762     },
12763   }
12764 \ExplSyntaxOff

```

³⁴This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

12765 \markdownSetup{rendererPrototypes={
12766   image = {%
12767     \begin{figure}%
12768     \begin{center}%
12769       \includegraphics[alt={#1}]{#3}%
12770     \end{center}%
12771     \ifx\empty#4\empty\else
12772       \caption{#4}%
12773     \fi
12774   \end{figure}},
12775   ulBegin = {\begin{itemize}},
12776   ulEnd = {\end{itemize}},
12777   olBegin = {\begin{enumerate}},
12778   olItem = {\item{}},
12779   olItemWithNumber = {\item[#1.]},
12780   olEnd = {\end{enumerate}},
12781   dlBegin = {\begin{description}},
12782   dlItem = {\item[#1]},
12783   dlEnd = {\end{description}},
12784   emphasis = {\emph{#1}},
12785   tickedBox = {\$\boxtimes$},
12786   halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

12787 \ExplSyntaxOn
12788 \seq_new:N
12789 \l_@@_header_identifiers_seq
12790 \markdownSetup
12791 {
12792   rendererPrototypes = {
12793     headerAttributeContextBegin = {
12794       \markdownSetup
12795       {
12796         rendererPrototypes = {
12797           attributeIdentifier = {
12798             \seq_put_right:Nn
12799             \l_@@_header_identifiers_seq
12800             { ##1 }
12801           },
12802         },
12803       }
12804     },
12805     headerAttributeContextEnd = {
12806       \seq_map_inline:Nn
12807       \l_@@_header_identifiers_seq
12808       { \label { ##1 } }
12809     \seq_clear:N
12810     \l_@@_header_identifiers_seq

```

```

12811     },
12812   },
12813 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

12814 \bool_new:N
12815   \l_@@_header_unnumbered_bool
12816 \markdownSetup
12817 {
12818   rendererPrototypes = {
12819     headerAttributeContextBegin += {
12820       \markdownSetup
12821       {
12822         rendererPrototypes = {
12823           attributeClassName = {
12824             \bool_if:nT
12825             {
12826               \str_if_eq_p:nn
12827                 { ##1 }
12828                 { unnumbered } &&
12829                 ! \l_@@_header_unnumbered_bool
12830             }
12831           {
12832             \group_begin:
12833             \bool_set_true:N
12834               \l_@@_header_unnumbered_bool
12835             \c@secnumdepth = 0
12836             \markdownSetup
12837             {
12838               rendererPrototypes = {
12839                 sectionBegin = {
12840                   \group_begin:
12841                   },
12842                 sectionEnd = {
12843                   \group_end:
12844                   },
12845                 },
12846               }
12847             }
12848           },
12849         },
12850       }
12851     },
12852   },
12853 }
12854 \ExplSyntaxOff

```

```

12855 \markdownSetup{rendererPrototypes={
12856   superscript = {\textsuperscript{#1}},
12857   subscript = {\textsubscript{#1}},
12858   blockQuoteBegin = {\begin{quotation}},
12859   blockQuoteEnd = {\end{quotation}},
12860   inputVerbatim = {\VerbatimInput{#1}},
12861   thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
12862   note = {\footnote{#1}}}}

```

3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

12863 \RequirePackage{ltxcmds}
12864 \ExplSyntaxOn
12865 \cs_gset:Npn
12866   \markdownRendererInputFencedCodePrototype#1#2#3
12867   {
12868     \tl_if_empty:nTF
12869       { #2 }
12870     { \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.

```

12871   {
12872     \regex_extract_once:nnN
12873       { \w* }
12874       { #2 }
12875     \l_tmpa_seq
12876     \seq_pop_left:NN
12877     \l_tmpa_seq
12878     \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

12879     \ltx@ifpackageloaded
12880     { minted }
12881     {
12882       \catcode`\#=6\relax
12883       \exp_args:NV
12884         \inputminted
12885         \l_tmpa_tl
12886         { #1 }
12887       \catcode`\#=12\relax
12888     }
12889     {

```

When the listings package is loaded, use it for syntax highlighting.

```

12890     \ltx@ifpackageloaded
12891     { listings }
12892     { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostiring were given.

```

12893             { \markdownRendererInputFencedCode{#1}{}{ } }
12894         }
12895     }
12896 }
12897 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

12898 \ExplSyntaxOn
12899 \def\markdownLATEXStrongEmphasis#1{%
12900     \str_if_in:NnTF
12901         \f@series
12902         { b }
12903         { \textnormal{#1} }
12904         { \textbf{#1} }
12905 }
12906 \ExplSyntaxOff
12907 \markdownSetup{rendererPrototypes={strongEmphasis={%
12908     \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L^AT_EX document classes that do not provide chapters.

```

12909 \@ifundefined{chapter}{%
12910     \markdownSetup{rendererPrototypes = {
12911         headingOne = {\section{#1}},
12912         headingTwo = {\subsection{#1}},
12913         headingThree = {\subsubsection{#1}},
12914         headingFour = {\paragraph{#1}},
12915         headingFive = {\subparagraph{#1}}}}
12916 }{%
12917     \markdownSetup{rendererPrototypes = {
12918         headingOne = {\chapter{#1}},
12919         headingTwo = {\section{#1}},
12920         headingThree = {\subsection{#1}},
12921         headingFour = {\subsubsection{#1}},
12922         headingFive = {\paragraph{#1}},
12923         headingSix = {\subparagraph{#1}}}}
12924 }%

```

3.3.4.2 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

12925 \markdownSetup{
12926     rendererPrototypes = {
12927         ulItem = {%
12928             \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
12929         },

```

```

12930   },
12931 }
12932 \def\markdownLaTeXUItem{%
12933   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
12934     \item[\markdownLaTeXCheckbox]%
12935     \expandafter\@gobble
12936   \else
12937     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
12938       \item[\markdownLaTeXCheckbox]%
12939       \expandafter\expandafter\expandafter\@gobble
12940     \else
12941       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
12942         \item[\markdownLaTeXCheckbox]%
12943         \expandafter\expandafter\expandafter\expandafter
12944         \expandafter\expandafter\expandafter\@gobble
12945       \else
12946         \item{}%
12947       \fi
12948     \fi
12949   \fi
12950 }

```

3.3.4.3 HTML elements

If the `html` option is enabled and we are using `TeX4ht`³⁵, we will pass HTML elements to the output HTML document unchanged.

```

12951 \@ifundefined{HCode}{}{
12952   \markdownSetup{
12953     rendererPrototypes = {
12954       inlineHtmlTag = {%
12955         \ifvmode
12956           \IgnorePar
12957         \EndP
12958         \fi
12959         \HCode{#1}%
12960       },
12961       inputBlockHtmlElement = {%
12962         \ifvmode
12963           \IgnorePar
12964         \fi
12965         \EndP
12966         \special{t4ht*#1}%
12967         \par
12968         \ShowPar
12969       },
12970     },

```

³⁵See <https://tug.org/tex4ht/>.

```

12971 }
12972 }

```

3.3.4.4 Citations

Here is a basic implementation for citations that uses the L^AT_EX `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the BibL^AT_EX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

12973 \newcount\markdownLaTeXCitationsCounter
12974
12975 % Basic implementation
12976 \RequirePackage{gobble}
12977 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
12978   \advance\markdownLaTeXCitationsCounter by 1\relax
12979   \ifx\relax#4\relax
12980     \ifx\relax#5\relax
12981       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12982         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
12983         \expandafter\expandafter\expandafter
12984         \expandafter\expandafter\expandafter\expandafter
12985         \@gobblethree
12986       \fi
12987     \else% Before a postnote (#5), dump the accumulator
12988       \ifx\relax#1\relax\else
12989         \cite{#1}%
12990       \fi
12991       \cite[#5]{#6}%
12992       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
12993       \else
12994         \expandafter\expandafter\expandafter
12995         \expandafter\expandafter\expandafter\expandafter
12996         \expandafter\expandafter\expandafter
12997         \expandafter\expandafter\expandafter\expandafter
12998         \markdownLaTeXBasicCitations
12999       \fi
13000       \expandafter\expandafter\expandafter
13001       \expandafter\expandafter\expandafter\expandafter{%
13002       \expandafter\expandafter\expandafter
13003       \expandafter\expandafter\expandafter\expandafter}%
13004       \expandafter\expandafter\expandafter
13005       \expandafter\expandafter\expandafter\expandafter{%
13006       \expandafter\expandafter\expandafter
13007       \expandafter\expandafter\expandafter\expandafter}%
13008       \expandafter\expandafter\expandafter
13009       \@gobblethree
13010     \fi

```

```

13011 \else% Before a prenote (#4), dump the accumulator
13012 \ifx\relax#1\relax\else
13013 \cite{#1}%
13014 \fi
13015 \ifnum\markdownLaTeXCitationsCounter>1\relax
13016 \space % Insert a space before the prenote in later citations
13017 \fi
13018 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
13019 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13020 \else
13021 \expandafter\expandafter\expandafter
13022 \expandafter\expandafter\expandafter\expandafter
13023 \markdownLaTeXBasicCitations
13024 \fi
13025 \expandafter\expandafter\expandafter{%
13026 \expandafter\expandafter\expandafter}%
13027 \expandafter\expandafter\expandafter{%
13028 \expandafter\expandafter\expandafter}%
13029 \expandafter
13030 \@gobblethree
13031 \fi\markdownLaTeXBasicCitations{#1#2#6},}
13032 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
13033
13034 % Natbib implementation
13035 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
13036 \advance\markdownLaTeXCitationsCounter by 1\relax
13037 \ifx\relax#3\relax
13038 \ifx\relax#4\relax
13039 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13040 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
13041 \expandafter\expandafter\expandafter
13042 \expandafter\expandafter\expandafter\expandafter
13043 \@gobbletwo
13044 \fi
13045 \else% Before a postnote (#4), dump the accumulator
13046 \ifx\relax#1\relax\else
13047 \citep{#1}%
13048 \fi
13049 \citep[] [#4]{#5}%
13050 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13051 \else
13052 \expandafter\expandafter\expandafter
13053 \expandafter\expandafter\expandafter\expandafter
13054 \expandafter\expandafter\expandafter
13055 \expandafter\expandafter\expandafter\expandafter
13056 \markdownLaTeXNatbibCitations
13057 \fi

```



```

13058     \expandafter\expandafter\expandafter
13059     \expandafter\expandafter\expandafter\expandafter{%
13060     \expandafter\expandafter\expandafter
13061     \expandafter\expandafter\expandafter\expandafter}%
13062     \expandafter\expandafter\expandafter
13063     \@gobbletwo
13064     \fi
13065 \else% Before a prenote (#3), dump the accumulator
13066     \ifx\relax#1\relax\relax\else
13067         \citep{#1}%
13068     \fi
13069     \citep[#3][#4]{#5}%
13070     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13071     \else
13072         \expandafter\expandafter\expandafter
13073         \expandafter\expandafter\expandafter\expandafter
13074         \markdownLaTeXNatbibCitations
13075     \fi
13076     \expandafter\expandafter\expandafter{%
13077     \expandafter\expandafter\expandafter}%
13078     \expandafter
13079     \@gobbletwo
13080     \fi\markdownLaTeXNatbibCitations{#1,#5}}
13081 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
13082     \advance\markdownLaTeXCitationsCounter by 1\relax
13083     \ifx\relax#3\relax
13084         \ifx\relax#4\relax
13085             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13086                 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
13087                 \expandafter\expandafter\expandafter
13088                 \expandafter\expandafter\expandafter\expandafter
13089                 \@gobbletwo
13090             \fi
13091         \else% After a prenote or a postnote, dump the accumulator
13092             \ifx\relax#1\relax\else
13093                 \citet{#1}%
13094             \fi
13095             , \citet[#3][#4]{#5}%
13096             \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
13097                 ,
13098             \else
13099                 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
13100                     ,
13101                 \fi
13102             \fi
13103             \expandafter\expandafter\expandafter
13104             \expandafter\expandafter\expandafter\expandafter

```

```

13105     \markdownLaTeXNatbibTextCitations
13106     \expandafter\expandafter\expandafter
13107     \expandafter\expandafter\expandafter\expandafter{%
13108     \expandafter\expandafter\expandafter
13109     \expandafter\expandafter\expandafter\expandafter}%
13110     \expandafter\expandafter\expandafter
13111     \@gobbletwo
13112     \fi
13113 \else% After a prenote or a postnote, dump the accumulator
13114     \ifx\relax#1\relax\relax\else
13115         \citet{#1}%
13116     \fi
13117     , \citet[#3][#4]{#5}%
13118     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
13119     ,
13120     \else
13121         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
13122         ,
13123         \fi
13124     \fi
13125     \expandafter\expandafter\expandafter
13126     \markdownLaTeXNatbibTextCitations
13127     \expandafter\expandafter\expandafter{%
13128     \expandafter\expandafter\expandafter}%
13129     \expandafter
13130     \@gobbletwo
13131     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
13132
13133 % BibLaTeX implementation
13134 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
13135     \advance\markdownLaTeXCitationsCounter by 1\relax
13136     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13137         \autocites#1[#3][#4]{#5}%
13138         \expandafter\@gobbletwo
13139     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
13140 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
13141     \advance\markdownLaTeXCitationsCounter by 1\relax
13142     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13143         \textcites#1[#3][#4]{#5}%
13144         \expandafter\@gobbletwo
13145     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
13146
13147 \markdownSetup{rendererPrototypes = {
13148     cite = {%
13149         \markdownLaTeXCitationsCounter=1%
13150         \def\markdownLaTeXCitationsTotal{#1}%
13151         \@ifundefined{autocites}{%

```

```

13152     \@ifundefined{citep}{%
13153         \expandafter\expandafter\expandafter
13154         \markdownLaTeXBasicCitations
13155         \expandafter\expandafter\expandafter{%
13156         \expandafter\expandafter\expandafter}%
13157         \expandafter\expandafter\expandafter{%
13158         \expandafter\expandafter\expandafter}%
13159     }{%
13160         \expandafter\expandafter\expandafter
13161         \markdownLaTeXNatbibCitations
13162         \expandafter\expandafter\expandafter{%
13163         \expandafter\expandafter\expandafter}%
13164     }%
13165 }{%
13166     \expandafter\expandafter\expandafter
13167     \markdownLaTeXBibLaTeXCitations
13168     \expandafter{\expandafter}%
13169 },
13170 textCite = {%
13171     \markdownLaTeXCitationsCounter=1%
13172     \def\markdownLaTeXCitationsTotal{#1}%
13173     \@ifundefined{autocites}{%
13174         \@ifundefined{citep}{%
13175             \expandafter\expandafter\expandafter
13176             \markdownLaTeXBasicTextCitations
13177             \expandafter\expandafter\expandafter{%
13178             \expandafter\expandafter\expandafter}%
13179             \expandafter\expandafter\expandafter{%
13180             \expandafter\expandafter\expandafter}%
13181         }{%
13182             \expandafter\expandafter\expandafter
13183             \markdownLaTeXNatbibTextCitations
13184             \expandafter\expandafter\expandafter{%
13185             \expandafter\expandafter\expandafter}%
13186         }%
13187     }{%
13188         \expandafter\expandafter\expandafter
13189         \markdownLaTeXBibLaTeXTextCitations
13190         \expandafter{\expandafter}%
13191     }}}}

```

3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```

13192 \RequirePackage{url}
13193 \RequirePackage{expl3}
13194 \ExplSyntaxOn

```

```

13195 \def\markdownRendererLinkPrototype#1#2#3#4{
13196   \tl_set:Nn \l_tmpa_tl { #1 }
13197   \tl_set:Nn \l_tmpb_tl { #2 }
13198   \bool_set:Nn
13199     \l_tmpa_bool
13200     {
13201       \tl_if_eq_p:NN
13202         \l_tmpa_tl
13203         \l_tmpb_tl
13204     }
13205   \tl_set:Nn \l_tmpa_tl { #4 }
13206   \bool_set:Nn
13207     \l_tmpb_bool
13208     {
13209       \tl_if_empty_p:N
13210         \l_tmpa_tl
13211     }

```

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.

```

13212   \bool_if:nTF
13213     {
13214       \l_tmpa_bool && \l_tmpb_bool
13215     }
13216     {
13217       \markdownLaTeXRendererAutolink { #2 } { #3 }
13218     }{
13219       \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
13220     }
13221 }
13222 \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.

```

13223   \tl_set:Nn
13224     \l_tmpa_tl
13225     { #2 }
13226   \tl_trim_spaces:N
13227     \l_tmpa_tl
13228   \tl_set:Nx
13229     \l_tmpb_tl
13230     {
13231       \tl_range:Nnn
13232         \l_tmpa_tl
13233         { 1 }
13234         { 1 }
13235     }

```

```

13236 \str_if_eq:NNTF
13237   \l_tmpb_tl
13238   \c_hash_str
13239   {
13240     \tl_set:Nx
13241       \l_tmpb_tl
13242       {
13243         \tl_range:Nnn
13244           \l_tmpa_tl
13245           { 2 }
13246           { -1 }
13247       }
13248     \exp_args:NV
13249     \ref
13250     \l_tmpb_tl
13251   }{
13252     \url { #2 }
13253   }
13254 }
13255 \ExplSyntaxOff
13256 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
13257   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

3.3.4.6 Tables

Here is a basic implementation of tables. If the `booktabs` package is loaded, then it is used to produce horizontal lines.

```

13258 \newcount\markdownLaTeXRowCount
13259 \newcount\markdownLaTeXRowTotal
13260 \newcount\markdownLaTeXColumnCounter
13261 \newcount\markdownLaTeXColumnTotal
13262 \newtoks\markdownLaTeXTable
13263 \newtoks\markdownLaTeXTableAlignment
13264 \newtoks\markdownLaTeXTableEnd
13265 \AtBeginDocument{%
13266   \@ifpackageloaded{booktabs}{%
13267     \def\markdownLaTeXTopRule{\toprule}%
13268     \def\markdownLaTeXMidRule{\midrule}%
13269     \def\markdownLaTeXBottomRule{\bottomrule}%
13270   }{%
13271     \def\markdownLaTeXTopRule{\hline}%
13272     \def\markdownLaTeXMidRule{\hline}%
13273     \def\markdownLaTeXBottomRule{\hline}%
13274   }%
13275 }
13276 \markdownSetup{rendererPrototypes={
13277   table = {%

```

```

13278 \markdownLaTeXTable={}%
13279 \markdownLaTeXTableAlignment={}%
13280 \markdownLaTeXTableEnd={%
13281 \markdownLaTeXBottomRule
13282 \end{tabular}}%
13283 \ifx\empty#1\empty\else
13284 \addto@hook\markdownLaTeXTable{%
13285 \begin{table}
13286 \centering}%
13287 \addto@hook\markdownLaTeXTableEnd{%
13288 \caption{#1}
13289 \end{table}}%
13290 \fi
13291 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
13292 \markdownLaTeXRowCount=0%
13293 \markdownLaTeXRowTotal=#2%
13294 \markdownLaTeXColumnTotal=#3%
13295 \markdownLaTeXRenderTableRow
13296 }
13297 }}
13298 \def\markdownLaTeXRenderTableRow#1{%
13299 \markdownLaTeXColumnCounter=0%
13300 \ifnum\markdownLaTeXRowCount=0\relax
13301 \markdownLaTeXReadAlignments#1%
13302 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
13303 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
13304 \the\markdownLaTeXTableAlignment}}%
13305 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
13306 \else
13307 \markdownLaTeXRenderTableCell#1%
13308 \fi
13309 \ifnum\markdownLaTeXRowCount=1\relax
13310 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
13311 \fi
13312 \advance\markdownLaTeXRowCount by 1\relax
13313 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
13314 \the\markdownLaTeXTable
13315 \the\markdownLaTeXTableEnd
13316 \expandafter\@gobble
13317 \fi\markdownLaTeXRenderTableRow}
13318 \def\markdownLaTeXReadAlignments#1{%
13319 \advance\markdownLaTeXColumnCounter by 1\relax
13320 \if#1d%
13321 \addto@hook\markdownLaTeXTableAlignment{1}%
13322 \else
13323 \addto@hook\markdownLaTeXTableAlignment{#1}%
13324 \fi

```

```

13325 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
13326   \expandafter@gobble
13327 \fi\markdownLaTeXReadAlignments}
13328 \def\markdownLaTeXRenderTableCell#1{%
13329   \advance\markdownLaTeXColumnCounter by 1\relax
13330   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
13331     \addto@hook\markdownLaTeXTable{#1&}%
13332   \else
13333     \addto@hook\markdownLaTeXTable{#1\\}%
13334     \expandafter@gobble
13335   \fi\markdownLaTeXRenderTableCell}

```

3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

13336
13337 \markdownIfOption{lineBlocks}{%
13338   \RequirePackage{verse}
13339   \markdownSetup{rendererPrototypes={
13340     lineBlockBegin = {%
13341       \begingroup
13342         \def\markdownRendererHardLineBreak{\\}%
13343         \begin{verse}%
13344       },
13345     lineBlockEnd = {%
13346       \end{verse}%
13347     \endgroup
13348   },
13349 }}
13350 }{}
13351

```

3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

13352 \ExplSyntaxOn
13353 \keys_define:nn
13354 { markdown/jekyllData }
13355 {
13356   author .code:n = { \author{#1} },
13357   date   .code:n = { \date{#1}   },
13358   title  .code:n = { \title{#1}  },
13359 }

```

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.

```

13360 \markdownSetup{
13361   rendererPrototypes = {
13362     jekyllDataEnd = {
13363       \AddToHook{begindocument/end}{\maketitle}
13364     },
13365   },
13366 }
13367 \ExplSyntaxOff

```

3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

13368 \markdownIfOption{strikeThrough}{%
13369   \RequirePackage{soulutf8}%
13370   \markdownSetup{
13371     rendererPrototypes = {
13372       strikeThrough = {%
13373         \st{#1}%
13374       },
13375     }
13376   }
13377 }{}

```

3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```

13378 \markdownIfOption{mark}{%
13379   \RequirePackage{soulutf8}%
13380   \markdownSetup{
13381     rendererPrototypes = {
13382       mark = {%
13383         \hl{#1}%
13384       },
13385     }
13386   }
13387 }{}

```

3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form

$\langle key \rangle = \langle value \rangle$ set the corresponding keys of the graphicx package to the corresponding values.

```

13388 \ExplSyntaxOn
13389 \@_if_option:nT
13390 { linkAttributes }
13391 {
13392   \RequirePackage{graphicx}
13393   \markdownSetup{
13394     rendererPrototypes = {
13395       imageAttributeContextBegin = {
13396         \group_begin:
13397         \markdownSetup{
13398           rendererPrototypes = {
13399             attributeKeyValue = {
13400               \setkeys
13401                 { Gin }
13402                 { { ##1 } = { ##2 } }
13403             },
13404           },
13405         }
13406       },
13407       imageAttributeContextEnd = {
13408         \group_end:
13409       },
13410     },
13411   }
13412 }
13413 \ExplSyntaxOff

```

3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

13414 \ExplSyntaxOn
13415 \cs_gset:Npn
13416   \markdownRendererInputRawInlinePrototype#1#2
13417   {
13418     \str_case:nnF
13419       { #2 }
13420       {
13421         { latex }
13422         {
13423           \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13424             { #1 }
13425             { tex }
13426         }
13427       }

```

```

13428     {
13429     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13430     { #1 }
13431     { #2 }
13432     }
13433   }
13434 \cs_gset:Npn
13435 \markdownRendererInputRawBlockPrototype#1#2
13436 {
13437   \str_case:nnF
13438   { #2 }
13439   {
13440     { latex }
13441     {
13442       \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13443       { #1 }
13444       { tex }
13445     }
13446   }
13447   {
13448     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13449     { #1 }
13450     { #2 }
13451   }
13452 }
13453 \ExplSyntaxOff
13454 \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.

```

13455 \newcommand\markdownMakeOther{%
13456   \count0=128\relax
13457   \loop
13458     \catcode\count0=11\relax
13459     \advance\count0 by 1\relax
13460   \ifnum\count0<256\repeat}%

```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain

TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L^ATeX package.

```
13461 \def\markdownMakeOther{%
13462   \count0=128\relax
13463   \loop
13464     \catcode\count0=11\relax
13465     \advance\count0 by 1\relax
13466   \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.

```
13467   \catcode`|=12}%
```

3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```
13468 \long\def\inputmarkdown{%
13469   \dosingleempty
13470   \doinputmarkdown}%
13471 \long\def\doinputmarkdown[#1]#2{%
13472   \begingroup
13473     \iffirstargument
13474     \setupmarkdown[#1]%
13475   \fi
13476   \markdownInput{#2}%
13477   \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` 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. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)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.

```
13478 \startluacode
13479   document.markdown_buffering = false
13480   local function preserve_trailing_spaces(line)
13481     if document.markdown_buffering then
13482       line = line:gsub("[ \t][ \t]$", "\t\t")
```

```

13483     end
13484     return line
13485 end
13486 resolvers.installinputlinehandler(preserve_trailing_spaces)
13487 \stopluacode
13488 \begingroup
13489   \catcode`\|=0%
13490   \catcode`\|=12%
13491   |gdef|startmarkdown{%
13492     |ctxlua{document.markdown_buffering = true}%
13493     |markdownReadAndConvert{\stopmarkdown}%
13494     {|stopmarkdown}}%
13495   |gdef|stopmarkdown{%
13496     |ctxlua{document.markdown_buffering = false}%
13497     |markdownEnd}%
13498 |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.

```

13499 \ExplSyntaxOn
13500 \cs_gset:Nn
13501   \@@_load_theme:nn
13502   {

```

Determine whether a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain \TeX theme instead.

```

13503   \file_if_exist:nTF
13504     { t - markdown theme #2.tex }
13505     {
13506       \msg_info:nnn
13507         { markdown }
13508         { loading-context-theme }
13509         { #1 }
13510       \usemodule
13511         [ t ]
13512         [ markdown theme #2 ]
13513     }
13514     {
13515       \@@_plain_tex_load_theme:nn
13516         { #1 }
13517         { #2 }
13518     }
13519   }
13520 \msg_new:nnn

```

```

13521 { markdown }
13522 { loading-context-theme }
13523 { Loading~ConTeXt~Markdown~theme~#1 }
13524 \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:

```
13525 \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.

```

13526 \markdownIfOption{plain}{\iffalse}{\iftrue}
13527 \def\markdownRendererHardLineBreakPrototype{\blank}%
13528 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
13529 \def\markdownRendererRightBracePrototype{\textbraceright}%
13530 \def\markdownRendererDollarSignPrototype{\textdollar}%
13531 \def\markdownRendererPercentSignPrototype{\percent}%
13532 \def\markdownRendererUnderscorePrototype{\textunderscore}%
13533 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
13534 \def\markdownRendererBackslashPrototype{\textbackslash}%
13535 \def\markdownRendererTildePrototype{\textasciitilde}%
13536 \def\markdownRendererPipePrototype{\char`|}%
13537 \def\markdownRendererLinkPrototype#1#2#3#4{%
13538   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
13539     \fi\tt<\hyphenatedurl{#3}>}}%
13540 \usemodule[database]
13541 \defineseparatedlist
13542   [MarkdownConTeXtCSV]
13543   [separator={,},
13544     before=\bTABLE,after=\eTABLE,
13545     first=\bTR,last=\eTR,
13546     left=\bTD,right=\eTD]
13547 \def\markdownConTeXtCSV{csv}
13548 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13549   \def\markdownConTeXtCSV@arg{#1}%
13550   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
13551     \placetable[] [tab:#1]{#4}{%
13552       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
13553   \else
13554     \markdownInput{#3}%
13555   \fi}%

```

```

13556 \def\markdownRendererImagePrototype#1#2#3#4{%
13557   \placefigure [] [] {#4}{\externalfigure[#3]}%
13558 \def\markdownRendererUlBeginPrototype{\startitemize}%
13559 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
13560 \def\markdownRendererUlItemPrototype{\item}%
13561 \def\markdownRendererUlEndPrototype{\stopitemize}%
13562 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
13563 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
13564 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
13565 \def\markdownRendererOlItemPrototype{\item}%
13566 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
13567 \def\markdownRendererOlEndPrototype{\stopitemize}%
13568 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
13569 \definedescription
13570   [MarkdownConTeXtDlItemPrototype]
13571   [location=hanging,
13572    margin=standard,
13573    headstyle=bold]%
13574 \definestartstop
13575   [MarkdownConTeXtDlPrototype]
13576   [before=\blank,
13577    after=\blank]%
13578 \definestartstop
13579   [MarkdownConTeXtDlTightPrototype]
13580   [before=\blank\startpacked,
13581    after=\stoppacked\blank]%
13582 \def\markdownRendererDlBeginPrototype{%
13583   \startMarkdownConTeXtDlPrototype}%
13584 \def\markdownRendererDlBeginTightPrototype{%
13585   \startMarkdownConTeXtDlTightPrototype}%
13586 \def\markdownRendererDlItemPrototype#1{%
13587   \startMarkdownConTeXtDlItemPrototype{#1}}%
13588 \def\markdownRendererDlItemEndPrototype{%
13589   \stopMarkdownConTeXtDlItemPrototype}%
13590 \def\markdownRendererDlEndPrototype{%
13591   \stopMarkdownConTeXtDlPrototype}%
13592 \def\markdownRendererDlEndTightPrototype{%
13593   \stopMarkdownConTeXtDlTightPrototype}%
13594 \def\markdownRendererEmphasisPrototype#1{\em#1}%
13595 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13596 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
13597 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
13598 \def\markdownRendererLineBlockBeginPrototype{%
13599   \begingroup
13600     \def\markdownRendererHardLineBreak{
13601       }%
13602     \startlines

```

```

13603 }%
13604 \def\markdownRendererLineBlockEndPrototype{%
13605     \stoptlines
13606     \endgroup
13607 }%
13608 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13609 \ExplSyntaxOn
13610 \cs_gset:Npn
13611   \markdownRendererInputFencedCodePrototype#1#2#3
13612   {
13613     \tl_if_empty:nTF
13614       { #2 }
13615     { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
  ~~~ latex
  \documentclass{article}
  \begin{document}
    Hello world!
  \end{document}
  ~~~
  \stopmarkdown
\stoptext

```

```

13616   {
13617     \regex_extract_once:nnN
13618       { \w* }
13619       { #2 }
13620     \l_tmpa_seq
13621     \seq_pop_left:NN
13622       \l_tmpa_seq
13623     \l_tmpa_tl

```

```

13624     \typefile[\l_tmpa_t1] []{#1}
13625     }
13626   }
13627 \ExplSyntaxOff
13628 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
13629 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
13630 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
13631 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
13632 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
13633 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
13634 \def\markdownRendererThematicBreakPrototype{%
13635   \blackrule[height=1pt, width=\hsize]}%
13636 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
13637 \def\markdownRendererTickedBoxPrototype{\$ \boxtimes$}
13638 \def\markdownRendererHalfTickedBoxPrototype{\$ \boxdot$}
13639 \def\markdownRendererUntickedBoxPrototype{\$ \square$}
13640 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
13641 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
13642 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
13643 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%

```

3.4.3.2 Tables

There is a basic implementation of tables.

```

13644 \newcount\markdownConTeXtRowCounter
13645 \newcount\markdownConTeXtRowTotal
13646 \newcount\markdownConTeXtColumnCounter
13647 \newcount\markdownConTeXtColumnTotal
13648 \newtoks\markdownConTeXtTable
13649 \newtoks\markdownConTeXtTableFloat
13650 \def\markdownRendererTablePrototype#1#2#3{%
13651   \markdownConTeXtTable={}%
13652   \ifx\empty#1\empty
13653     \markdownConTeXtTableFloat={%
13654       \the\markdownConTeXtTable}%
13655   \else
13656     \markdownConTeXtTableFloat={%
13657       \placetable{#1}{\the\markdownConTeXtTable}}%
13658   \fi
13659   \begingroup
13660   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13661   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13662   \setupTABLE[r][1][topframe=on, bottomframe=on]
13663   \setupTABLE[r][#1][bottomframe=on]
13664   \markdownConTeXtRowCounter=0%
13665   \markdownConTeXtRowTotal=#2%
13666   \markdownConTeXtColumnTotal=#3%

```



```

13667 \markdownConTeXtRenderTableRow}
13668 \def\markdownConTeXtRenderTableRow#1{%
13669 \markdownConTeXtColumnCounter=0%
13670 \ifnum\markdownConTeXtRowCounter=0\relax
13671 \markdownConTeXtReadAlignments#1%
13672 \markdownConTeXtTable={\bTABLE}%
13673 \else
13674 \markdownConTeXtTable=\expandafter{%
13675 \the\markdownConTeXtTable\bTR}%
13676 \markdownConTeXtRenderTableCell#1%
13677 \markdownConTeXtTable=\expandafter{%
13678 \the\markdownConTeXtTable\eTR}%
13679 \fi
13680 \advance\markdownConTeXtRowCounter by 1\relax
13681 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
13682 \markdownConTeXtTable=\expandafter{%
13683 \the\markdownConTeXtTable\eTABLE}%
13684 \the\markdownConTeXtTableFloat
13685 \endgroup
13686 \expandafter\gobbleoneargument
13687 \fi\markdownConTeXtRenderTableRow}
13688 \def\markdownConTeXtReadAlignments#1{%
13689 \advance\markdownConTeXtColumnCounter by 1\relax
13690 \if#1d%
13691 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13692 \fi\if#1l%
13693 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13694 \fi\if#1c%
13695 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
13696 \fi\if#1r%
13697 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
13698 \fi
13699 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13700 \expandafter\gobbleoneargument
13701 \fi\markdownConTeXtReadAlignments}
13702 \def\markdownConTeXtRenderTableCell#1{%
13703 \advance\markdownConTeXtColumnCounter by 1\relax
13704 \markdownConTeXtTable=\expandafter{%
13705 \the\markdownConTeXtTable\bTD#1\eTD}%
13706 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13707 \expandafter\gobbleoneargument
13708 \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`.

```

13709 \ExplSyntaxOn
13710 \cs_gset:Npn
13711   \markdownRendererInputRawInlinePrototype#1#2
13712   {
13713     \str_case:nnF
13714       { #2 }
13715       {
13716         { latex }
13717         {
13718           \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13719             { #1 }
13720             { context }
13721         }
13722       }
13723     {
13724       \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13725         { #1 }
13726         { #2 }
13727     }
13728   }
13729 \cs_gset:Npn
13730   \markdownRendererInputRawBlockPrototype#1#2
13731   {
13732     \str_case:nnF
13733       { #2 }
13734       {
13735         { context }
13736         {
13737           \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13738             { #1 }
13739             { tex }
13740         }
13741       }
13742     {
13743       \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13744         { #1 }
13745         { #2 }
13746     }
13747   }
13748 \cs_gset_eq:NN
13749   \markdownRendererInputRawBlockPrototype
13750   \markdownRendererInputRawInlinePrototype
13751 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}
13752 \ExplSyntaxOff
13753 \stopmodule
13754 \protect

```

At the end of the ConTEXt module, we load the [witiko/markdown/defaults](#)

ConTeXt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```
13755 \markdownIfOption{noDefaults}{-}{-}{
13756   \setupmarkdown[theme=witiko/markdown/defaults]
13757 }
13758 \stopmodule
13759 \protect
```

References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [8] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [9] Vít 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).
- [10] Geoffrey M. Poore. *The minted Package. Highlighted source code in L^AT_EX*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [11] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

- [12] Johannes Braams et al. *The L^AT_EX_{2 ϵ} Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [13] Donald Ervin Knuth. *T_EX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [14] Victor Eijkhout. *T_EX by Topic. A T_EXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

Index

autoIdentifiers	18, 30, 72, 85
blankBeforeBlockquote	18
blankBeforeCodeFence	19
blankBeforeDivFence	19
blankBeforeHeading	19
blankBeforeList	20
bracketedSpans	20, 74
breakableBlockquotes	20
cacheDir	4, 15, 17, 53, 54, 131, 281, 348, 363
citationNbsps	21
citations	21, 76, 77
codeSpans	22
contentBlocks	17, 22
contentBlocksLanguageMap	17
contentLevel	23
debugExtensions	9, 17, 23, 277
debugExtensionsFileName	17, 23
defaultOptions	9, 47, 328
definitionLists	23, 80
eagerCache	15
entities.char_entity	189
entities.dec_entity	189
entities.hex_entity	189
entities.hex_entity_with_x_char	189
expandtabs	242
expectJekyllData	24
extensions	25, 139, 283
extensions.bracketed_spans	283

<code>extensions.citations</code>	284
<code>extensions.content_blocks</code>	288
<code>extensions.definition_lists</code>	291
<code>extensions.fancy_lists</code>	293
<code>extensions.fenced_code</code>	299
<code>extensions.fenced_divs</code>	304
<code>extensions.header_attributes</code>	308
<code>extensions.inline_code_attributes</code>	309
<code>extensions.jekyll_data</code>	325
<code>extensions.line_blocks</code>	310
<code>extensions.link_attributes</code>	311
<code>extensions.mark</code>	310
<code>extensions.notes</code>	313
<code>extensions.pipe_table</code>	315
<code>extensions.raw_inline</code>	319
<code>extensions.strike_through</code>	320
<code>extensions.subscripts</code>	320
<code>extensions.superscripts</code>	321
<code>extensions.tex_math</code>	322
<code>fancyLists</code>	27, 95–99, 364
<code>fencedCode</code>	27, 35, 77, 84, 100, 361
<code>fencedCodeAttributes</code>	28, 72
<code>fencedDiv</code>	85
<code>fencedDivs</code>	28, 37
<code>finalizeCache</code>	16, 18, 29, 29, 53, 54, 130, 282
<code>frozenCache</code>	18, 29, 54, 130, 133, 134, 361, 363
<code>frozenCacheCounter</code>	29, 282, 355
<code>frozenCacheFileName</code>	18, 29, 53, 282
<code>gfmAutoIdentifiers</code>	18, 29, 72, 85
<code>hashEnumerators</code>	30
<code>headerAttributes</code>	30, 37, 72, 85
<code>html</code>	31, 88, 374
<code>hybrid</code>	31, 36, 42, 44, 57, 65, 101, 131, 194, 243, 354
<code>inlineCodeAttributes</code>	31, 72, 78
<code>inlineNotes</code>	32
<code>\input</code>	51
<code>\inputmarkdown</code>	136, 136, 137, 387
<code>inputTempFileName</code>	54, 57, 349, 350, 353
<code>iterlines</code>	242

jeekyllData	3, 24, 25, 32, 108–111
languages_json	288, 288
lineBlocks	33, 90
linkAttributes	33, 72, 89, 92, 260, 384
mark	34, 93, 384
\markdown	129
markdown	129, 129, 357
markdown*	129, 129, 130, 357
\markdown_jeekyll_data_concatenate_address:NN	344
\markdown_jeekyll_data_pop:	344
\markdown_jeekyll_data_push:nN	344
\markdown_jeekyll_data_push_address_segment:n	342
\markdown_jeekyll_data_set_keyval:Nn	345
\markdown_jeekyll_data_set_keyvals:nn	345
\markdown_jeekyll_data_update_address_tls:	344
\markdownBegin	49, 49–51, 127, 129, 136
\markdownCleanup	348
\markdownEnd	49, 49–51, 127, 129, 136
\markdownError	127, 127
\markdownEscape	49, 51, 355
\markdownIfOption	52
\markdownIfSnippetExists	66
\markdownInfo	127, 127
\markdownInput	49, 51, 129, 130, 136, 354, 357
\markdownInputFileStream	349
\markdownInputPlainTeX	357
\markdownLoadPlainTeXTheme	131, 138, 337
\markdownLuaExecute	351, 354
\markdownLuaOptions	346, 348
\markdownMakeOther	127, 386, 387
\markdownOptionFinalizeCache	53
\markdownOptionFrozenCache	53
\markdownOptionHybrid	57
\markdownOptionInputTempFileName	54
\markdownOptionNoDefaults	56
\markdownOptionOutputDir	54, 54
\markdownOptionPlain	55
\markdownOptionStripPercentSigns	56
\markdownOutputFileStream	349
\markdownPrepare	348

<code>\markdownPrepareLuaOptions</code>	346
<code>\markdownReadAndConvert</code>	127, 349, 357, 358, 387
<code>\markdownReadAndConvertProcessLine</code>	350, 351
<code>\markdownReadAndConvertStripPercentSigns</code>	350
<code>\markdownReadAndConvertTab</code>	349
<code>\markdownRendererAttributeClassName</code>	72
<code>\markdownRendererAttributeIdentifier</code>	72
<code>\markdownRendererAttributeKeyValue</code>	72
<code>\markdownRendererBlockQuoteBegin</code>	73
<code>\markdownRendererBlockQuoteEnd</code>	73
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	74
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	74
<code>\markdownRendererCite</code>	76, 77
<code>\markdownRendererCodeSpan</code>	78
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	78
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	78
<code>\markdownRendererContentBlock</code>	79, 79
<code>\markdownRendererContentBlockCode</code>	80
<code>\markdownRendererContentBlockOnlineImage</code>	79
<code>\markdownRendererDisplayMath</code>	106
<code>\markdownRendererDlBegin</code>	80
<code>\markdownRendererDlBeginTight</code>	81
<code>\markdownRendererDlDefinitionBegin</code>	82
<code>\markdownRendererDlDefinitionEnd</code>	82
<code>\markdownRendererDlEnd</code>	82
<code>\markdownRendererDlEndTight</code>	82
<code>\markdownRendererDlItem</code>	81
<code>\markdownRendererDlItemEnd</code>	81
<code>\markdownRendererDocumentBegin</code>	93
<code>\markdownRendererDocumentEnd</code>	93
<code>\markdownRendererEllipsis</code>	38, 83
<code>\markdownRendererEmphasis</code>	83, 114
<code>\markdownRendererFancyOlBegin</code>	95, 96
<code>\markdownRendererFancyOlBeginTight</code>	96
<code>\markdownRendererFancyOlEnd</code>	99
<code>\markdownRendererFancyOlEndTight</code>	99
<code>\markdownRendererFancyOlItem</code>	97
<code>\markdownRendererFancyOlItemEnd</code>	97
<code>\markdownRendererFancyOlItemWithNumber</code>	98
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	84
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	84
<code>\markdownRendererFencedDivAttributeContextBegin</code>	85

<code>\markdownRendererFencedDivAttributeContextEnd</code>	85
<code>\markdownRendererHalfTickedBox</code>	107
<code>\markdownRendererHardLineBreak</code>	91
<code>\markdownRendererHeaderAttributeContextBegin</code>	85
<code>\markdownRendererHeaderAttributeContextEnd</code>	85
<code>\markdownRendererHeadingFive</code>	87
<code>\markdownRendererHeadingFour</code>	87
<code>\markdownRendererHeadingOne</code>	86
<code>\markdownRendererHeadingSix</code>	87
<code>\markdownRendererHeadingThree</code>	86
<code>\markdownRendererHeadingTwo</code>	86
<code>\markdownRendererImage</code>	89
<code>\markdownRendererImageAttributeContextBegin</code>	89
<code>\markdownRendererImageAttributeContextEnd</code>	89
<code>\markdownRendererInlineHtmlComment</code>	88
<code>\markdownRendererInlineHtmlTag</code>	88
<code>\markdownRendererInlineMath</code>	106
<code>\markdownRendererInputBlockHtmlElement</code>	88
<code>\markdownRendererInputFencedCode</code>	77
<code>\markdownRendererInputRawBlock</code>	100
<code>\markdownRendererInputRawInline</code>	99
<code>\markdownRendererInputVerbatim</code>	77
<code>\markdownRendererInterblockSeparator</code>	90
<code>\markdownRendererJekyllDataBegin</code>	108
<code>\markdownRendererJekyllDataBoolean</code>	110
<code>\markdownRendererJekyllDataEmpty</code>	111
<code>\markdownRendererJekyllDataEnd</code>	108
<code>\markdownRendererJekyllDataMappingBegin</code>	109
<code>\markdownRendererJekyllDataMappingEnd</code>	109
<code>\markdownRendererJekyllDataNumber</code>	110
<code>\markdownRendererJekyllDataSequenceBegin</code>	109
<code>\markdownRendererJekyllDataSequenceEnd</code>	110
<code>\markdownRendererJekyllDataString</code>	111
<code>\markdownRendererLineBlockBegin</code>	91
<code>\markdownRendererLineBlockEnd</code>	91
<code>\markdownRendererLink</code>	92, 114
<code>\markdownRendererLinkAttributeContextBegin</code>	92
<code>\markdownRendererLinkAttributeContextEnd</code>	92
<code>\markdownRendererMark</code>	93
<code>\markdownRendererNbsp</code>	94
<code>\markdownRendererNote</code>	94
<code>\markdownRendererOlBegin</code>	95

<code>\markdownRendererOlBeginTight</code>	95
<code>\markdownRendererOlEnd</code>	98
<code>\markdownRendererOlEndTight</code>	98
<code>\markdownRendererOlItem</code>	38, 96
<code>\markdownRendererOlItemEnd</code>	96
<code>\markdownRendererOlItemWithNumber</code>	38, 97
<code>\markdownRendererParagraphSeparator</code>	90
<code>\markdownRendererReplacementCharacter</code>	101
<code>\markdownRendererSectionBegin</code>	100
<code>\markdownRendererSectionEnd</code>	100
<code>\markdownRendererSoftLineBreak</code>	91
<code>\markdownRendererStrikeThrough</code>	104
<code>\markdownRendererStrongEmphasis</code>	84
<code>\markdownRendererSubscript</code>	104
<code>\markdownRendererSuperscript</code>	105
<code>\markdownRendererTable</code>	106
<code>\markdownRendererTableAttributeContextBegin</code>	105
<code>\markdownRendererTableAttributeContextEnd</code>	105
<code>\markdownRendererTextCite</code>	77
<code>\markdownRendererThematicBreak</code>	107
<code>\markdownRendererTickedBox</code>	107
<code>\markdownRendererUlBegin</code>	74
<code>\markdownRendererUlBeginTight</code>	75
<code>\markdownRendererUlEnd</code>	76
<code>\markdownRendererUlEndTight</code>	76
<code>\markdownRendererUlItem</code>	75
<code>\markdownRendererUlItemEnd</code>	75
<code>\markdownRendererUntickedBox</code>	107
<code>\markdownSetup</code>	52, 52, 57, 130, 137, 358, 359
<code>\markdownSetupSnippet</code>	65, 65
<code>\markdownWarning</code>	127, 127
<code>\markinline</code>	49, 50, 51, 129, 352, 356
<code>\markinlinePlainTeX</code>	356
<code>new</code>	7, 16, 328
<code>notes</code>	34, 94
<code>parsers</code>	208, 242
<code>parsers.punctuation</code>	210
<code>pipeTables</code>	6, 35, 41, 106
<code>preserveTabs</code>	35, 39, 242
<code>rawAttribute</code>	35, 36, 100

reader	8, 26, 139, 208, 241, 283
reader->add_special_character	8, 9, 26, 277
reader->auto_link_email	267
reader->auto_link_url	267
reader->create_parser	243
reader->finalize_grammar	273, 333
reader->initialize_named_group	277
reader->insert_pattern	8, 9, 26, 273, 279
reader->lookup_reference	255
reader->normalize_tag	242
reader->options	242
reader->parser_functions	243
reader->parser_functions.name	243
reader->parsers	242, 242
reader->register_link	254
reader->update_rule	273, 276, 279
reader->writer	242
reader.new	241, 241, 333
relativeReferences	36
\setupmarkdown	137
shiftHeadings	6, 37
singletonCache	16
slice	6, 37, 191, 202, 203
smartEllipses	38, 83, 131
\startmarkdown	136, 136, 387
startNumber	38, 96–98
\stopmarkdown	136, 136, 387
strikeThrough	38, 104, 384
stripIndent	39, 243
stripPercentSigns	349, 350
subscripts	39, 104
superscripts	40, 105
syntax	274, 278
tableAttributes	40, 105
tableCaptions	6, 40, 41, 105
taskLists	41, 107, 373
texComments	42, 243
texMathDollars	42, 106
texMathDoubleBackslash	43, 106
texMathSingleBackslash	43, 106

tightLists	43, 75, 76, 81, 83, 95, 96, 98, 99, 364
underscores	44
util.cache	139, 140
util.cache_verbatim	140
util.encode_json_string	140
util.err	139
util.escaper	142
util.expand_tabs_in_line	140
util.flatten	141
util.intersperse	142
util.map	142
util.pathname	143
util.rope_last	141
util.rope_to_string	141
util.table_copy	140
util.walk	140, 141
walkable_syntax	8, 17, 23, 273, 276–278
writer	139, 139, 190, 283
writer->active_attributes	201, 201–203
writer->attribute_type_levels	201
writer->attributes	199
writer->block_html_element	198
writer->blockquote	198
writer->bulletitem	196
writer->bulletlist	196
writer->citations	284
writer->code	194
writer->contentblock	289
writer->defer_call	208, 208
writer->definitionlist	291
writer->display_math	322
writer->div_begin	304
writer->div_end	304
writer->document	199
writer->ellipsis	193
writer->emphasis	198
writer->escape	194
writer->escape_minimal	194
writer->escape_programmatic_text	194
writer->escape_typographic_text	194

writer->escaped_chars	193, 194
writer->escaped_minimal_strings	193, 194
writer->escaped_strings	193
writer->escaped_uri_chars	193, 194
writer->fancyitem	294
writer->fancylist	293
writer->fencedCode	299
writer->flatten_inlines	190, 190
writer->get_state	207
writer->hard_line_break	192
writer->heading	206
writer->identifier	194
writer->image	195
writer->infostring	194
writer->inline_html_comment	197
writer->inline_html_tag	197
writer->inline_math	322
writer->interblocksep	192
writer->is_writing	191, 191
writer->jekyllData	325
writer->lineblock	310
writer->link	195
writer->mark	311
writer->math	194
writer->nbsp	191
writer->note	313
writer->options	190
writer->ordereditem	197
writer->orderedlist	196
writer->pack	192, 282
writer->paragraph	192
writer->paragraphsep	192
writer->plain	191
writer->pop_attributes	201, 202, 203
writer->push_attributes	201, 202, 203
writer->rawBlock	300
writer->rawInline	319
writer->set_state	208
writer->slice_begin	191
writer->slice_end	191
writer->soft_line_break	192
writer->space	191

<code>writer->span</code>	<i>283</i>
<code>writer->strike_through</code>	<i>320</i>
<code>writer->string</code>	<i>194</i>
<code>writer->strong</code>	<i>198</i>
<code>writer->subscript</code>	<i>320</i>
<code>writer->suffix</code>	<i>191</i>
<code>writer->superscript</code>	<i>321</i>
<code>writer->table</code>	<i>316</i>
<code>writer->thematic_break</code>	<i>193</i>
<code>writer->checkbox</code>	<i>198</i>
<code>writer->undosep</code>	<i>192, 281</i>
<code>writer->uri</code>	<i>194</i>
<code>writer->verbatim</code>	<i>198</i>
<code>writer.new</code>	<i>190, 190, 333</i>