

# A Markdown Interpreter for T<sub>E</sub>X

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

Version 3.9.1-0-g92254dfb  
2024-12-17

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>163</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	164
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	376
1.3	Acknowledgements . . . . .	7	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	408
<b>2</b>	<b>Interfaces</b>	<b>7</b>	3.4	ConT <sub>E</sub> Xt Implementation	448
2.1	Lua Interface . . . . .	7			
2.2	Plain T <sub>E</sub> X Interface . . . . .	53			
2.3	L <sup>A</sup> T <sub>E</sub> X Interface . . . . .	149			
2.4	ConT <sub>E</sub> Xt Interface . . . . .	158			
				<b>References</b>	<b>459</b>
				<b>Index</b>	<b>461</b>

## List of Figures

1	A block diagram of the Markdown package . . . . .	8
2	A sequence diagram of typesetting a document using the T <sub>E</sub> X interface . . . . .	49
3	A sequence diagram of typesetting a document using the Lua CLI . . . . .	50
4	Various formats of mathematical formulae . . . . .	74
5	The banner of the Markdown package . . . . .	75
6	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	255

## 1 Introduction

The Markdown package<sup>1</sup> converts CommonMark<sup>2</sup> markup to T<sub>E</sub>X commands. The functionality is provided both as a Lua module and as plain T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X, and ConT<sub>E</sub>Xt macro packages that can be used to directly typeset T<sub>E</sub>X 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

---

<sup>1</sup>See <https://ctan.org/pkg/markdown>.

<sup>2</sup>See <https://commonmark.org/>.

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

```
1 local metadata = {
2   version   = "(((VERSION)))",
3   comment   = "A module for the conversion from markdown "
4             .. "to plain TeX",
5   author    = "John MacFarlane, Hans Hagen, Vít Starý Novotný, "
6             .. "Andrej Genčur",
7   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
8               "2016-2024 Vít Starý Novotný, Andrej Genčur"},
9   license   = "LPPL 1.3c"
10 }
11
12 if not modules then modules = { } end
13 modules['markdown'] = metadata
```

## 1.1 Requirements

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

### 1.1.1 Lua Requirements

The Lua part of the package requires that the following Lua modules are available from within the Lua<sub>TeX</sub> engine (though not necessarily in the LuaMeta<sub>TeX</sub> engine).

**LPeg  $\geq$  0.10** A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg  $\geq$  0.10 is included in Lua<sub>TeX</sub>  $\geq$  0.72.0 (T<sub>E</sub>X Live  $\geq$  2013).

```
14 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 Lua<sub>TeX</sub> (T<sub>E</sub>X Live  $\geq$  2008).

```
15 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 Lua<sub>TeX</sub> (T<sub>E</sub>X Live  $\geq$  2008).

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

---

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

**Kpathsea** A package that implements the loading of third-party Lua libraries and looking up files in the T<sub>E</sub>X directory structure.

```
17 ;(function()
```

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

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

All the abovelisted modules are statically linked into the current version of the LuaT<sub>E</sub>X engine [1, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

**lua-uni-algos** A package that implements Unicode case-folding in T<sub>E</sub>X Live  $\geq$  2020.

```
25 hard lua-uni-algos
26 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.

```
27 hard lua-tinyyaml
```

### 1.1.2 Plain T<sub>E</sub>X Requirements

The plain T<sub>E</sub>X part of the package requires that the plain T<sub>E</sub>X 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<sup>A</sup>T<sub>E</sub>X3 kernel in T<sub>E</sub>X 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.

```
28 hard l3kernel
29 \unprotect
```

```

30 \expandafter\ifx\csname ExplSyntaxOn\endcsname\relax
31   \input expl3-generic
32 \fi

```

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

```

33 hard lt3luabridge

```

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

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

The plain TeX code makes use of the `isdir` method that was added to the Lua File System library by the LuaTeX engine developers [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<sup>A</sup>T<sub>E</sub>X Requirements

The L<sup>A</sup>T<sub>E</sub>X part of the package requires that the L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> format is loaded, a TeX engine that extends  $\epsilon$ -TeX, and all the plain TeX prerequisites (see Section 1.1.2).

```

34 \NeedsTeXFormat{LaTeX2e}
35 \RequirePackage{expl3}

```

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or L<sup>A</sup>T<sub>E</sub>X themes (see Section 2.3.4) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```

36 soft url

```

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

```
37 soft graphics
```

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

The package `paralist` will be used unless the option `experimental` has been enabled, in which case, the package `enumitem` will be used. Furthermore, enabling any test phase [2] will also cause `enumitem` to be used. In a future major version, `enumitem` will replace `paralist` altogether.

```
38 soft enumitem
```

```
39 soft paralist
```

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

```
40 soft fancyvrb
```

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

```
41 soft csvsimple
```

```
42 soft pgf # required by `csvsimple`, which loads `pgfkeys`
```

```
43 soft tools # required by `csvsimple`, which loads `shellesc`
```

```
44 soft etoolbox # required by `csvsimple`, which loads `etoolbox`
```

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

```
45 soft amsmath
```

```
46 soft amsfonts
```

**graphicx** A package that provides extended support for graphics. It is used in the `witiko/diagrams`, and `witiko/graphicx/http` plain T<sub>E</sub>X themes, see Section 2.2.3.

```
47 soft graphics
```

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

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

**soul and xcolor** Packages that are used in the default renderer prototypes for strike-throughs and marked text in pdf $\TeX$ .

```
50 soft soul
51 soft xcolor
```

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

```
52 soft lua-ul
53 soft luacolor
```

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

```
54 soft ltxcmds
```

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

```
55 soft luaxml
```

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

```
56 soft verse
```

#### 1.1.4 Con $\TeX$ t Prerequisites

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

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

## 1.2 Feedback

Please use the Markdown project page on GitHub<sup>4</sup> to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the  $\TeX$ - $\LaTeX$  Stack Exchange.<sup>5</sup> community question answering web site under the `markdown` tag.

---

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

<sup>5</sup>See <https://tex.stackexchange.com>.

### 1.3 Acknowledgements

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

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

Funding by the Faculty of Informatics at the Masaryk University in Brno [3] is gratefully acknowledged.

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

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

## 2 Interfaces

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

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

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

### 2.1 Lua Interface

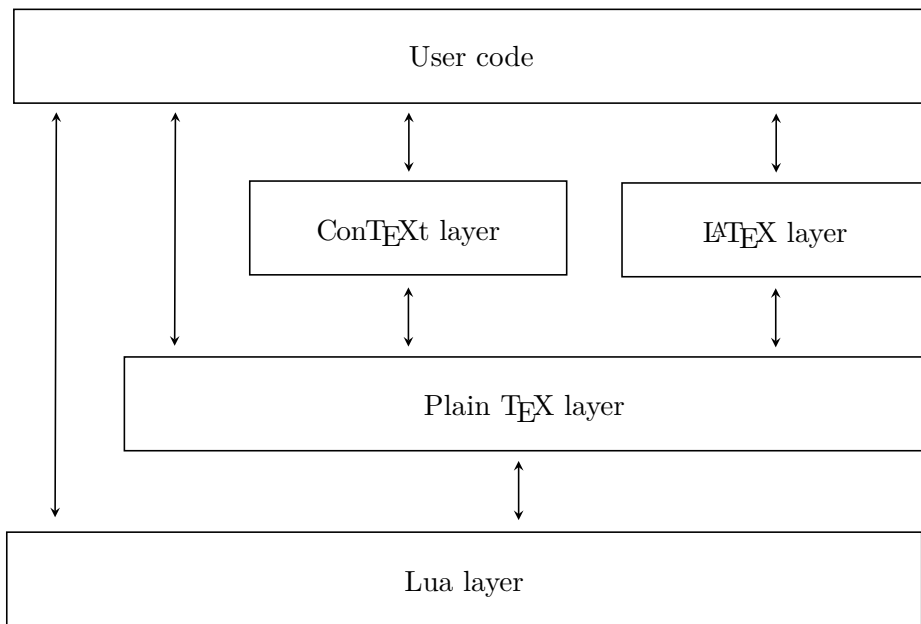
The Lua interface provides the conversion from UTF-8 encoded markdown to plain  $\text{T}_{\text{E}}\text{X}$ . This interface is used by the plain  $\text{T}_{\text{E}}\text{X}$  implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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

```
57 local M = {metadata = metadata}
```

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

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



**Figure 1: A block diagram of the Markdown package**

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

The following example Lua code converts the markdown string `Hello *world*!` to a T<sub>E</sub>X output using the default options and prints the T<sub>E</sub>X 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.

```

58 local walkable_syntax = {

```



```

59 Block = {
60   "Blockquote",
61   "Verbatim",
62   "ThematicBreak",
63   "BulletList",
64   "OrderedList",
65   "DisplayHtml",
66   "Heading",
67 },
68 BlockOrParagraph = {
69   "Block",
70   "Paragraph",
71   "Plain",
72 },
73 Inline = {
74   "Str",
75   "Space",
76   "Endline",
77   "EndlineBreak",
78   "LinkAndEmph",
79   "Code",
80   "AutoLinkUrl",
81   "AutoLinkEmail",
82   "AutoLinkRelativeReference",
83   "InlineHtml",
84   "HtmlEntity",
85   "EscapedChar",
86   "Smart",
87   "Symbol",
88 },
89 }

```

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.

```
90 local defaultOptions = {}
```

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

```
91 \ExplSyntaxOn
92 \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.

```
93 \prop_new:N \g_@@_lua_option_types_prop
94 \prop_new:N \g_@@_default_lua_options_prop
95 \seq_new:N \g_@@_option_layers_seq
96 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
97 \seq_gput_right:NV
98   \g_@@_option_layers_seq
99   \c_@@_option_layer_lua_tl
100 \cs_new:Nn
101   \@@_add_lua_option:nnn
102   {
103     \@@_add_option:Vnnn
104       \c_@@_option_layer_lua_tl
105       { #1 }
106       { #2 }
107       { #3 }
108   }
109 \cs_new:Nn
110   \@@_add_option:nnnn
111   {
112     \seq_gput_right:cn
113       { g_@@_ #1 _options_seq }
114       { #2 }
115     \prop_gput:cnn
116       { g_@@_ #1 _option_types_prop }
117       { #2 }
118       { #3 }
119     \prop_gput:cnn
120       { g_@@_default_ #1 _options_prop }
121       { #2 }
122       { #4 }
123     \@@_typecheck_option:n
124       { #2 }
125   }
```

```

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

```

```

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

```

```

220 \cs_new:Nn
221   \@@_get_option_type: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_@@_ ##1 _option_types_prop }
230           { #1 }
231         \l_tmpa_tl
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     \seq_if_in:NVF
247       \g_@@_option_types_seq
248       \l_tmpa_tl
249     {
250       \msg_error:nnnV
251         { markdown }
252         { unknown-option-type }
253         { #1 }
254       \l_tmpa_tl
255     }
256     \tl_set_eq:NN
257       #2
258       \l_tmpa_tl
259   }
260 \msg_new:nnn
261   { markdown }
262   { unknown-option-type }
263   {
264     Option~#1~has~unknown~type~#2.
265   }
266 \msg_new:nnn

```

```

267 { markdown }
268 { undefined-option }
269 {
270   Option~#1~is~undefined.
271 }
272 \cs_new:Nn
273   \@@_get_default_option_value:nN
274   {
275     \bool_set_false:N
276       \l_tmpa_bool
277     \seq_map_inline:Nn
278       \g_@@_option_layers_seq
279       {
280         \prop_get:cnNT
281           { g_@@_default_ ##1 _options_prop }
282           { #1 }
283           #2
284           {
285             \bool_set_true:N
286               \l_tmpa_bool
287             \seq_map_break:
288           }
289         }
290     \bool_if:nF
291       \l_tmpa_bool
292       {
293         \msg_error:nnn
294           { markdown }
295           { undefined-option }
296           { #1 }
297       }
298   }
299 \cs_new:Nn
300   \@@_get_option_value:nN
301   {
302     \@@_option_tl_to_csname:nN
303       { #1 }
304     \l_tmpa_tl
305     \cs_if_free:cTF
306       { \l_tmpa_tl }
307     {
308       \@@_get_default_option_value:nN
309         { #1 }
310       #2
311     }
312     {
313       \@@_get_option_type:nN

```

```

314     { #1 }
315     \l_tmpa_tl
316     \str_if_eq:NNTF
317     \c_@@_option_type_counter_tl
318     \l_tmpa_tl
319     {
320       \@@_option_tl_to_csname:nN
321       { #1 }
322       \l_tmpa_tl
323       \tl_set:Nx
324       #2
325       { \the \cs:w \l_tmpa_tl \cs_end: }
326     }
327     {
328       \@@_option_tl_to_csname:nN
329       { #1 }
330       \l_tmpa_tl
331       \tl_set:Nv
332       #2
333       { \l_tmpa_tl }
334     }
335   }
336 }
337 \cs_new:Nn \@@_option_tl_to_csname:nN
338 {
339   \tl_set:Nn
340   \l_tmpa_tl
341   { \str_uppercase:n { #1 } }
342   \tl_set:Nx
343   #2
344   {
345     markdownOption
346     \tl_head:f { \l_tmpa_tl }
347     \tl_tail:n { #1 }
348   }
349 }

```

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.

```

350 \cs_new:Nn \@@_with_various_cases:nn
351 {
352   \seq_clear:N
353   \l_tmpa_seq
354   \seq_map_inline:Nn
355   \g_@@_cases_seq
356   {

```

```

357     \tl_set:Nn
358     \l_tmpa_tl
359     { #1 }
360     \use:c { ##1 }
361     \l_tmpa_tl
362     \seq_put_right:NV
363     \l_tmpa_seq
364     \l_tmpa_tl
365   }
366   \seq_map_inline:Nn
367   \l_tmpa_seq
368   { #2 }
369 }

```

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

```

370 \cs_new:Nn \@@_with_various_cases_break:
371 {
372   \seq_map_break:
373 }

```

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

```

374 \seq_new:N \g_@@_cases_seq
375 \cs_new:Nn \@@_camel_case:N
376 {
377   \regex_replace_all:nnN
378   { _ ([a-z]) }
379   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
380   #1
381   \tl_set:Nx
382   #1
383   { #1 }
384 }
385 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
386 \cs_new:Nn \@@_snake_case:N
387 {
388   \regex_replace_all:nnN
389   { ([a-z])([A-Z]) }
390   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
391   #1
392   \tl_set:Nx
393   #1
394   { #1 }
395 }
396 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```



### 2.1.4 General Behavior

`eagerCache=true, false`

default: `true`

`true` Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. Furthermore, it can also significantly improve the processing speed for documents that require multiple compilation runs, since each markdown document is only converted once. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing.

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

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

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

```
397 \@@_add_lua_option:nnn
398   { eagerCache }
399   { boolean }
400   { true }
401 defaultOptions.eagerCache = true
```

`experimental=true, false`

default: `false`

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

At the moment, this just means that the version `experimental` of the theme `witiko/markdown/defaults` will be loaded and warnings for hard-deprecated features will become errors. However, the effects may extend to other areas in the future as well.

`false` Experimental features will be disabled.

```

402 \@@_add_lua_option:nnn
403   { experimental }
404   { boolean }
405   { false }

406 defaultOptions.experimental = false

```

`singletonCache=true, false`

default: true

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

**false** Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also #226 (comment)<sup>6</sup>. This was the default behavior until version 3.0.0 of the Markdown package.

```

407 \@@_add_lua_option:nnn
408   { singletonCache }
409   { boolean }
410   { true }

411 defaultOptions.singletonCache = true

412 local singletonCache = {
413   convert = nil,
414   options = nil,
415 }

```

`unicodeNormalization=true, false`

default: true

**true** Markdown documents will be normalized using one of the four Unicode normalization forms<sup>7</sup> before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

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

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

<sup>7</sup>See <https://unicode.org/faq/normalization.html>.

```

416 \@@_add_lua_option:nnn
417   { unicodeNormalization }
418   { boolean }
419   { true }

420 defaultOptions.unicodeNormalization = true

```

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

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

```

421 \@@_add_lua_option:nnn
422   { unicodeNormalizationForm }
423   { string }
424   { nfc }

425 defaultOptions.unicodeNormalizationForm = "nfc"

```

### 2.1.5 File and Directory Names

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

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain T<sub>E</sub>X 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.

```

426 \@@_add_lua_option:nnn
427   { cacheDir }
428   { path }
429   { \markdownOptionOutputDir / _markdown_\jobname }
430 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.

```

431 \@@_add_lua_option:nnn
432   { contentBlocksLanguageMap }
433   { path }
434   { markdown-languages.json }
435 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.

```

436 \@@_add_lua_option:nnn
437   { debugExtensionsFileName }
438   { path }
439   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
440 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 T<sub>E</sub>X document that contains markdown documents without invoking Lua using the `frozenCache` plain T<sub>E</sub>X option. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```

441 \@@_add_lua_option:nnn
442   { frozenCacheFileName }
443   { path }
444   { \markdownOptionCacheDir / frozenCache.tex }
445 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

## 2.1.6 Parser Options

`autoIdentifiers=true, false` default: false

**true** Enable the Pandoc auto identifiers syntax extension<sup>8</sup>:

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

```
# 123 Sesame Street
```

**false** Disable the Pandoc auto identifiers syntax extension.

See also the option `gfmAutoIdentifiers`.

```

446 \@@_add_lua_option:nnn
447   { autoIdentifiers }
448   { boolean }
449   { false }
450 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.

```

451 \@@_add_lua_option:nnn
452   { blankBeforeBlockquote }
453   { boolean }
454   { false }
455 defaultOptions.blankBeforeBlockquote = false

```

---

<sup>8</sup>See [https://pandoc.org/MANUAL.html#extension-auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-auto_identifiers).

`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.

```
456 \@@_add_lua_option:nnn
457   { blankBeforeCodeFence }
458   { boolean }
459   { false }
460 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.

```
461 \@@_add_lua_option:nnn
462   { blankBeforeDivFence }
463   { boolean }
464   { false }
465 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.

```
466 \@@_add_lua_option:nnn
467   { blankBeforeHeading }
468   { boolean }
469   { false }
470 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.

```
471 \@@_add_lua_option:nnn
472   { blankBeforeList }
473   { boolean }
474   { false }
475 defaultOptions.blankBeforeList = false
```

`bracketedSpans=true, false` default: false

**true** Enable the Pandoc bracketed span syntax extension<sup>9</sup>:

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

**false** Disable the Pandoc bracketed span syntax extension.

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

480 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.

```
481 \@@_add_lua_option:nnn
482   { breakableBlockquotes }
483   { boolean }
484   { true }

485 defaultOptions.breakableBlockquotes = true
```

`citationNbsps=true, false` default: false

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

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

```
486 \@@_add_lua_option:nnn
487   { citationNbsps }
488   { boolean }
489   { true }

490 defaultOptions.citationNbsps = true
```

---

<sup>9</sup>See [https://pandoc.org/MANUAL.html#extension-bracketed\\_spans](https://pandoc.org/MANUAL.html#extension-bracketed_spans).

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension<sup>10</sup>:

```
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.

```
491 \@@_add_lua_option:nmn
492 { citations }
493 { boolean }
494 { false }
495 defaultOptions.citations = false
```

`codeSpans=true, false`

default: true

`true` Enable the code span syntax:

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

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

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

```
496 \@@_add_lua_option:nmn
497 { codeSpans }
498 { boolean }
499 { true }
500 defaultOptions.codeSpans = true
```

<sup>10</sup>See <https://pandoc.org/MANUAL.html#extension-citations>.



`contentBlocks=true, false`

default: `false`

`true`

: Enable the iA Writer content blocks syntax extension [4]:

```
``` 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.

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

505 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
```

```
506 \@@_add_lua_option:nnn
507 { contentLevel }
508 { string }
509 { block }

510 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.

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

515 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.

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

520 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false` default: false

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

```
521 \@@_add_lua_option:nnn
522 { ensureJekyllData }
523 { boolean }
524 { false }
525 defaultOptions.ensureJekyllData = false
```

`expectJekyllData=true, false` default: false

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

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

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

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

```
526 \@@_add_lua_option:nmn
527 { expectJekyllData }
528 { boolean }
529 { false }

530 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:

```

531 metadata.user_extension_api_version = 2
532 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.

```

533 \cs_generate_variant:Nn
534 \@@_add_lua_option:nnn
535 { nnV }
536 \@@_add_lua_option:nnV
537 { extensions }
538 { clist }
539 \c_empty_clist

540 defaultOptions.extensions = {}

```

`fancyLists=true, false`

default: false

**true** Enable the Pandoc fancy list syntax extension<sup>11</sup>:

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

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

```
541 \@@_add_lua_option:nnn
542 { fancyLists }
543 { boolean }
544 { false }
545 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.

```
546 \@@_add_lua_option:nnn
547 { fencedCode }
548 { boolean }
549 { true }
550 defaultOptions.fencedCode = true
```

<sup>11</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>12</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.

```
551 \@@_add_lua_option:nnn
552 { fencedCodeAttributes }
553 { boolean }
554 { false }

555 defaultOptions.fencedCodeAttributes = false
```

`fencedDivs=true, false`

default: false

**true** Enable the Pandoc fenced div syntax extension<sup>13</sup>:

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

And another.
:::::
```

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

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

560 defaultOptions.fencedDivs = false
```

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

<sup>13</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.

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

565 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>`.

```
566 \@@_add_lua_option:nnn
567   { frozenCacheCounter }
568   { counter }
569   { 0 }

570 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers=true, false`

default: `false`

`true` Enable the Pandoc GitHub-flavored auto identifiers syntax extension<sup>14</sup>:

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

# 123 Sesame Street
```

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

---

<sup>14</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](#).

```
571 \@@_add_lua_option:nnn
572 { gfmAutoIdentifiers }
573 { boolean }
574 { false }

575 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.

```
576 \@@_add_lua_option:nnn
577 { hashEnumerators }
578 { boolean }
579 { false }

580 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.

```
581 \@@_add_lua_option:nnn
582 { headerAttributes }
583 { boolean }
584 { false }

585 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.

```
586 \@@_add_lua_option:nmn
587   { html }
588   { boolean }
589   { true }

590 defaultOptions.html = true
```

`hybrid=true, false`

default: `false`

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

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

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

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

```
Here is a mathematical formula:

/math-formula.tex
```

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

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

Here is a mathematical formula:

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

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

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

Here is a mathematical formula:

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

```
591 \@@_add_lua_option:nnn
592   { hybrid }
593   { boolean }
594   { false }
595 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false`

default: false

`true` Enable the Pandoc inline code span attribute extension<sup>15</sup>:

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

<sup>15</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.

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

600 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false` default: `false`

`true` Enable the Pandoc inline note syntax extension<sup>16</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.

```
601 \@@_add_lua_option:nnn
602 { inlineNotes }
603 { boolean }
604 { false }

605 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false` default: `false`

`true` Enable the Pandoc YAML metadata block syntax extension<sup>17</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>16</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

<sup>17</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.

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

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

**true** Enable the Pandoc link and image attribute syntax extension<sup>18</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.

```
611 \@@_add_lua_option:nnn
612 { linkAttributes }
613 { boolean }
614 { false }
615 defaultOptions.linkAttributes = false
```

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

**true** Enable the Pandoc line block syntax extension<sup>19</sup>:

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

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

```
616 \@@_add_lua_option:nnn
617 { lineBlocks }
618 { boolean }
619 { false }
620 defaultOptions.lineBlocks = false
```

---

<sup>18</sup>See [https://pandoc.org/MANUAL.html#extension-link\\_attributes](https://pandoc.org/MANUAL.html#extension-link_attributes).

<sup>19</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>20</sup>:

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

`false` Disable the Pandoc mark syntax extension.

```
621 \@@_add_lua_option:nnn
622   { mark }
623   { boolean }
624   { false }
625 defaultOptions.mark = false
```

`notes=true, false` default: false

`true` Enable the Pandoc note syntax extension<sup>21</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.

```
626 \@@_add_lua_option:nnn
627   { notes }
628   { boolean }
629   { false }
630 defaultOptions.notes = false
```

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

<sup>21</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.

```

631 \@@_add_lua_option:nnn
632 { pipeTables }
633 { boolean }
634 { false }

635 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.

```

636 \@@_add_lua_option:nnn
637 { preserveTabs }
638 { boolean }
639 { true }

640 defaultOptions.preserveTabs = true

```

`rawAttribute=true, false`

default: false

**true** Enable the Pandoc raw attribute syntax extension<sup>22</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 \\

```

<sup>22</sup>See [https://pandoc.org/MANUAL.html#extension-raw\\_attribute](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.

```
641 \@@_add_lua_option:nnn
642 { rawAttribute }
643 { boolean }
644 { false }
645 defaultOptions.rawAttribute = false
```

`relativeReferences=true, false`

default: `false`

`true`      Enable relative references<sup>23</sup> in autolinks:

```
I conclude in Section <#conclusion>.

Conclusion {#conclusion}
=====

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

`false`      Disable relative references in autolinks.

```
646 \@@_add_lua_option:nnn
647 { relativeReferences }
648 { boolean }
649 { false }
650 defaultOptions.relativeReferences = false
```

---

<sup>23</sup>See <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.

```
651 \@@_add_lua_option:nnn
652   { shiftHeadings }
653   { number }
654   { 0 }
655 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.

```
656 \@@_add_lua_option:nnn
657   { slice }
658   { slice }
659   { ^-$ }
660 defaultOptions.slice = "^ $"
```

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

- |                    |  |
|--------------------|--|
| <code>true</code>  | Convert any ellipses in the input to the <code>\markdownRendererEllipsis</code> TeX macro. |
| <code>false</code> | Preserve all ellipses in the input.  |

```

661 \@@_add_lua_option:nnn
662   { smartEllipses }
663   { boolean }
664   { false }

665 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 `\markdownRendererO1ItemWithNumber` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererO1Item` T<sub>E</sub>X macro.

```

666 \@@_add_lua_option:nnn
667   { startNumber }
668   { boolean }
669   { true }

670 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

- true**      Enable the Pandoc strike-through syntax extension<sup>24</sup>:

This ~~is deleted text.~~

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

```

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

675 defaultOptions.strikeThrough = false

```

---

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

`stripIndent=true, false`

default: `false`

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

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

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

```
676 \@@_add_lua_option:nmn
677   { stripIndent }
678   { boolean }
679   { false }
680 defaultOptions.stripIndent = false
```

`subscripts=true, false`

default: `false`

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

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

`false` Disable the Pandoc subscript syntax extension.

```
681 \@@_add_lua_option:nmn
682   { subscripts }
683   { boolean }
684   { false }
685 defaultOptions.subscripts = false
```

---

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

`superscripts=true, false`

default: false

**true** Enable the Pandoc superscript syntax extension<sup>26</sup>:

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

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

```
686 \@@_add_lua_option:nnn
687 { superscripts }
688 { boolean }
689 { false }

690 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.

```
691 \@@_add_lua_option:nnn
692 { tableAttributes }
693 { boolean }
694 { false }

695 defaultOptions.tableAttributes = false
```

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

`tableCaptions=true, false`

default: `false`

`true`

: Enable the Pandoc table caption syntax extension<sup>27</sup> for pipe tables (see the `pipeTables` option).

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

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

`false` Disable the Pandoc table caption syntax extension.

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

700 defaultOptions.tableCaptions = false
```

`taskLists=true, false`

default: `false`

`true` Enable the Pandoc task list syntax extension<sup>28</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.

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

705 defaultOptions.taskLists = false
```

<sup>27</sup>See [https://pandoc.org/MANUAL.html#extension-table\\_captions](https://pandoc.org/MANUAL.html#extension-table_captions).

<sup>28</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.

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

`texMathDollars=true, false`

default: false

**true** Enable the Pandoc dollar math syntax extension<sup>29</sup>:

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

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

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

---

<sup>29</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>30</sup>:

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

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

```
716 \\@@_add_lua_option:nnn
717   { texMathDoubleBackslash }
718   { boolean }
719   { false }

720 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

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

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

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

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

725 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>30</sup>See [https://pandoc.org/MANUAL.html#extension-tex\\_math\\_double\\_backslash](https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash).

<sup>31</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.

```

726 \@@_add_lua_option:nmn
727 { tightLists }
728 { boolean }
729 { true }

730 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.

```

731 \@@_add_lua_option:nmn
732 { underscores }
733 { boolean }
734 { true }
735 \ExplSyntaxOff

736 defaultOptions.underscores = true

```

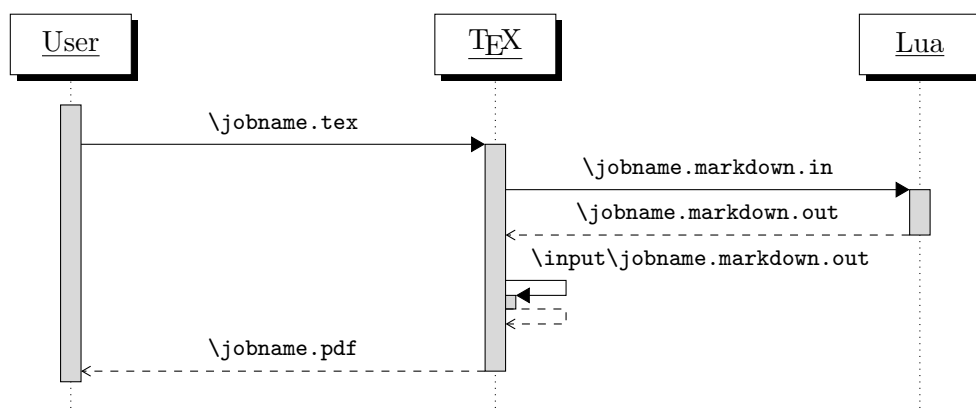


### 2.1.7 Command-Line Interface

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.

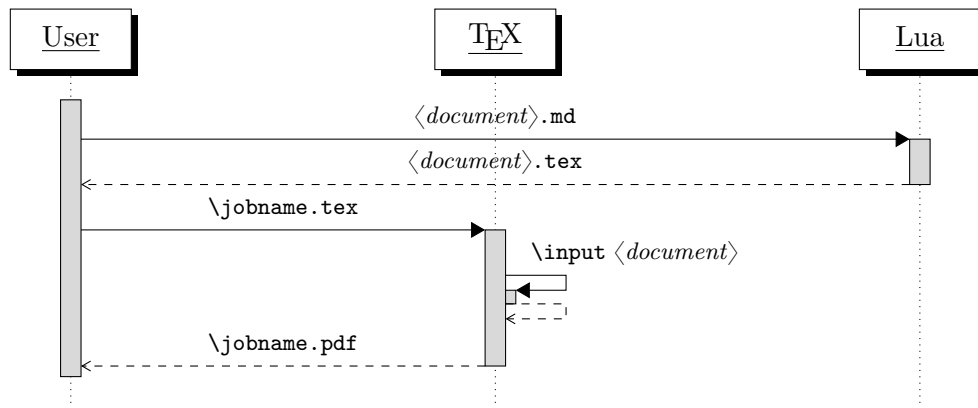
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted  $\text{T}_{\text{E}}\text{X}$  documents are cached on the file system, taking up increasing amount of space. Unless the  $\text{T}_{\text{E}}\text{X}$  engine includes a Lua 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**

```
737
738 local HELP_STRING = [[
739 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
740 where OPTIONS are documented in the Lua interface section of the
741 technical Markdown package documentation.
742
743 When OUTPUT_FILE is unspecified, the result of the conversion will be
744 written to the standard output. When INPUT_FILE is also unspecified, the
745 result of the conversion will be read from the standard input.
746
747 Report bugs to: witiko@mail.muni.cz
748 Markdown package home page: <https://github.com/witiko/markdown>]]
749
```



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

```

750 local VERSION_STRING = [[
751 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
752
753 Copyright (C) ]] .. table.concat(metadata.copyright,
754                                 "\nCopyright (C) ") .. [[
755
756 License: ]] .. metadata.license
757
758 local function warn(s)
759   io.stderr:write("Warning: " .. s .. "\n")
760 end
761
762 local function error(s)
763   io.stderr:write("Error: " .. s .. "\n")
764   os.exit(1)
765 end
  
```

To make it easier to copy-and-paste options from Pandoc [5] 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 [6] also show that snake\_case is faster to read than camelCase.

```

766 local function camel_case(option_name)
767   local cased_option_name = option_name:gsub("_(%l)", function(match)
768     return match:sub(2, 2):upper()
769   end)
770   return cased_option_name
771 end
772
773 local function snake_case(option_name)
774   local cased_option_name = option_name:gsub("%l%u", function(match)
  
```

```

775     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
776   end)
777   return cased_option_name
778 end
779
780 local cases = {camel_case, snake_case}
781 local various_case_options = {}
782 for option_name, _ in pairs(defaultOptions) do
783   for _, case in ipairs(cases) do
784     various_case_options[case(option_name)] = option_name
785   end
786 end
787
788 local process_options = true
789 local options = {}
790 local input_filename
791 local output_filename
792 for i = 1, #arg do
793   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.

```

794     if arg[i] == "--" then
795       process_options = false
796       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.

```

797     elseif arg[i]:match("=") then
798       local key, value = arg[i]:match("(.)=(.*)")
799       if defaultOptions[key] == nil and
800         various_case_options[key] ~= nil then
801         key = various_case_options[key]
802       end

```

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

```

803     local default_type = type(defaultOptions[key])
804     if default_type == "boolean" then
805       options[key] = (value == "true")
806     elseif default_type == "number" then
807       options[key] = tonumber(value)
808     elseif default_type == "table" then
809       options[key] = {}
810       for item in value:gmatch("[^,]+") do

```

```

811         table.insert(options[key], item)
812     end
813     else
814         if default_type ~= "string" then
815             if default_type == "nil" then
816                 warn('Option "' .. key .. '" not recognized.')
817             else
818                 warn('Option "' .. key .. '" type not recognized, ' ..
819                     'please file a report to the package maintainer.')
820             end
821             warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
822                 key .. '" as a string.')
823         end
824         options[key] = value
825     end
826     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.

```

827     elseif arg[i] == "--help" or arg[i] == "-h" then
828         print(HELP_STRING)
829         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.

```

830     elseif arg[i] == "--version" or arg[i] == "-v" then
831         print(VERSION_STRING)
832         os.exit()
833     end
834 end

```

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

```

835     if input_filename == nil then
836         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  $\text{T}_{\text{E}}\text{X}$  document that will result from the conversion.

```

837     elseif output_filename == nil then
838         output_filename = arg[i]
839     else
840         error('Unexpected argument: "' .. arg[i] .. "'.')
841     end
842     ::continue::

```

```
843 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 T<sub>E</sub>X document `hello.tex`. After the Markdown package for our T<sub>E</sub>X format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

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

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

```
844 \def\markdownLastModified{((LASTMODIFIED))}%  
845 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

It is expected that the special plain T<sub>E</sub>X characters have the expected category codes, when `\inputting` the file.

### 2.2.1 Typesetting Markdown and YAML

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

#### 2.2.1.1 Typesetting Markdown and YAML directly

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

```
846 \let\markdownBegin\relax  
847 \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 [7, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```
848 \let\yamlBegin\relax
849 \def\yamlEnd{\markdownEnd\endgroup}
```

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

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

```
\input markdown
\yamlBegin
title: _Hello_ **world** ...
```

```
author: John Doe
\yamlEnd
\bye
```

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

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

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

```
850 \let\markinline\relax
```

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

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

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

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

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

### 2.2.1.2 Typesetting Markdown and YAML from external documents

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

```
851 \let\markdownInput\relax
```

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

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

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

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

```
852 \def\yamlInput#1{%
853   \begingroup
854   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
855   \markdownInput{#1}%
856   \endgroup
857 }%
```

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

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

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

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

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

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

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

```
858 \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.

```
859 \ExplSyntaxOn
860 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
861 \cs_generate_variant:Nn
862   \tl_const:Nn
863   { NV }
864 \tl_if_exist:NF
865   \c_@@_top_layer_tl
866   {
867     \tl_const:NV
868       \c_@@_top_layer_tl
869       \c_@@_option_layer_plain_tex_tl
870   }
```

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

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

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

```
872 \prop_new:N \g_@@_plain_tex_option_types_prop
873 \prop_new:N \g_@@_default_plain_tex_options_prop
874 \seq_gput_right:NV
875   \g_@@_option_layers_seq
876   \c_@@_option_layer_plain_tex_tl
877 \cs_new:Nn
878   \@@_add_plain_tex_option:nnn
879   {
880     \@@_add_option:Vnnn
881       \c_@@_option_layer_plain_tex_tl
882       { #1 }
883       { #2 }
884       { #3 }
885   }
```

The plain  $\TeX$  options may be also be specified via the `\markdownSetup` macro. Here, the plain  $\TeX$  options are represented by a comma-delimited list of  $\langle key \rangle = \langle value \rangle$  pairs. For boolean options, the  $= \langle value \rangle$  part is optional, and  $\langle key \rangle$  will be interpreted

as `<key>=true` if the `=<value>` part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```

886 \cs_new:Nn
887   \@@_setup:n
888   {
889     \keys_set:nn
890       { markdown/options }
891       { #1 }
892   }
893 \cs_gset_eq:NN
894   \markdownSetup
895   \@@_setup:n

```

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

```

896 \cs_gset_eq:NN
897   \yamlSetup
898   \markdownSetup

```

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

```

899 \prg_new_conditional:Nnn
900   \@@_if_option:n
901   { TF, T, F }
902   {
903     \@@_get_option_type:nN
904       { #1 }
905     \l_tmpa_tl
906     \str_if_eq:NNF
907       \l_tmpa_tl
908       \c_@@_option_type_boolean_tl
909     {
910       \msg_error:nxxx
911         { markdown }
912         { expected-boolean-option }
913         { #1 }
914         { \l_tmpa_tl }
915     }
916     \@@_get_option_value:nN
917       { #1 }
918     \l_tmpa_tl
919     \str_if_eq:NNTF
920       \l_tmpa_tl
921       \c_@@_option_value_true_tl
922     { \prg_return_true: }
923     { \prg_return_false: }

```

```

924 }
925 \msg_new:nnn
926 { markdown }
927 { expected-boolean-option }
928 {
929   Option~#1~has~type~#2,~
930   but~a~boolean~was~expected.
931 }
932 \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`.

```

933 \@@_add_plain_tex_option:nnn
934 { frozenCache }
935 { boolean }
936 { false }

```

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

1. Remove the `cacheDir` cache directory with stale auxiliary cache files.
2. Enable the `finalizeCache` option.
4. Typeset the plain T<sub>E</sub>X document to populate and finalize the cache.
5. Enable the `frozenCache` option.
6. Publish the source code of the plain T<sub>E</sub>X document and the `cacheDir` directory.

**2.2.2.2 File and Directory Names** The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`.

The expansion of this macro must not contain quotation marks (") or backslash symbols (\). Mind that T<sub>E</sub>X engines tend to put quotation marks around `\jobname`, when it contains spaces.

```

937 \@@_add_plain_tex_option:nnn
938 { inputTempFileName }
939 { path }
940 { \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.

```

941 \@@_add_plain_tex_option:nnn
942   { outputDir }
943   { path }
944   { . }

```

### 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  $\TeX$  formats such as  $\LaTeX$  and  $\ConTeXt$ . Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

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

```
945 \@@_add_plain_tex_option:nnn
946   { plain }
947   { boolean }
948   { 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.

```
949 \@@_add_plain_tex_option:nnn
950   { noDefaults }
951   { boolean }
952   { 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 [8] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
953 \seq_gput_right:Nn
954   \g_@@_plain_tex_options_seq
955   { stripPercentSigns }
956 \prop_gput:Nnn
957   \g_@@_plain_tex_option_types_prop
958   { stripPercentSigns }
959   { boolean }
960 \prop_gput:Nnx
961   \g_@@_default_plain_tex_options_prop
962   { stripPercentSigns }
963   { 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.

```
964 \cs_new:Nn
965   \@@_define_option_commands_and_keyvals:
966   {
967     \seq_map_inline:Nn
968       \g_@@_option_layers_seq
969       {
970         \seq_map_inline:cn
971           { g_@@_ ##1 _options_seq }
972           {
973             \@@_define_option_command:n
974               { #####1 }
```

To make it easier to copy-and-paste options from Pandoc [5] 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 [6] also show that `snake_case` is faster to read than `camelCase`.

```
975         \@@_with_various_cases:nn
976           { #####1 }
977         {
978           \@@_define_option_keyval:nnn
979             { ##1 }
980             { #####1 }
981             { #####1 }
982         }
983     }
984 }
985 }
986 \cs_new:Nn
987   \@@_define_option_command:n
988   {
```

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.

```

989   \str_if_eq:nnTF
990     { #1 }
991     { outputDir }
992     { \@@_define_option_command_output_dir: }
993     {

```

Do not override options defined before loading the package.

```

994     \@@_option_tl_to_csname:nN
995     { #1 }
996     \l_tmpa_tl
997   \cs_if_exist:cF
998     { \l_tmpa_tl }
999     {
1000       \@@_get_default_option_value:nN
1001       { #1 }
1002       \l_tmpa_tl
1003       \@@_set_option_value:nV
1004       { #1 }
1005       \l_tmpa_tl
1006     }
1007   }
1008 }
1009 \ExplSyntaxOff
1010 \input lt3luabridge.tex

```

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.

```

1011 \ExplSyntaxOn
1012 \cs_new:Nn
1013   \@@_define_option_command_output_dir:
1014   {
1015     \cs_if_free:NT
1016       \markdownOptionOutputDir
1017       {
1018         \bool_if:nTF
1019           {
1020             \cs_if_exist_p:N
1021               \luabridge_tl_set:Nn &&
1022             (
1023               \int_compare_p:nNn
1024                 { \g_luabridge_method_int }
1025                 =
1026                 { \c_luabridge_method_directlua_int } ||

```

```

1027         \sys_if_shell_unrestricted_p:
1028     )
1029     }
1030     {

```

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

```

1031         \group_begin:
1032         \cctab_select:N
1033         \c_str_cctab
1034         \luabridge_tl_set:Nn
1035         \l_tmpa_tl
1036         { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
1037         \tl_gset:NV
1038         \markdownOptionOutputDir
1039         \l_tmpa_tl
1040         \group_end:
1041     }
1042     {
1043         \tl_gset:Nn
1044         \markdownOptionOutputDir
1045         { . }
1046     }
1047 }
1048 }
1049 \cs_new:Nn
1050 \@@_set_option_value:nn
1051 {
1052     \@@_define_option:n
1053     { #1 }
1054     \@@_get_option_type:nN
1055     { #1 }
1056     \l_tmpa_tl
1057     \str_if_eq:NNTF
1058     \c_@@_option_type_counter_tl
1059     \l_tmpa_tl
1060     {
1061         \@@_option_tl_to_csname:nN
1062         { #1 }
1063         \l_tmpa_tl
1064         \int_gset:cn
1065         { \l_tmpa_tl }
1066         { #2 }
1067     }
1068     {
1069         \@@_option_tl_to_csname:nN

```



```

1070         { #1 }
1071         \l_tmpa_tl
1072     \cs_set:cpn
1073     { \l_tmpa_tl }
1074     { #2 }
1075 }
1076 }
1077 \cs_generate_variant:Nn
1078 \@@_set_option_value:nn
1079 { nV }
1080 \cs_new:Nn
1081 \@@_define_option:n
1082 {
1083     \@@_option_tl_to_csname:nN
1084     { #1 }
1085     \l_tmpa_tl
1086     \cs_if_free:cT
1087     { \l_tmpa_tl }
1088     {
1089         \@@_get_option_type:nN
1090         { #1 }
1091         \l_tmpb_tl
1092         \str_if_eq:NNT
1093         \c_@@_option_type_counter_tl
1094         \l_tmpb_tl
1095         {
1096             \@@_option_tl_to_csname:nN
1097             { #1 }
1098             \l_tmpa_tl
1099             \int_new:c
1100             { \l_tmpa_tl }
1101         }
1102     }
1103 }
1104 \cs_new:Nn
1105 \@@_define_option_keyval:nnn
1106 {
1107     \prop_get:cnN
1108     { g_@@_ #1 _option_types_prop }
1109     { #2 }
1110     \l_tmpa_tl
1111     \str_if_eq:VVTF
1112     \l_tmpa_tl
1113     \c_@@_option_type_boolean_tl
1114     {
1115         \keys_define:nn
1116         { markdown/options }

```

```
1117         {
```

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

```
1118         #3 .code:n = {
1119             \tl_set:Nx
1120             \l_tmpa_tl
1121             {
1122                 \str_case:nnF
1123                 { ##1 }
1124                 {
1125                     { yes } { true }
1126                     { no } { false }
1127                 }
1128                 { ##1 }
1129             }
1130             \@@_set_option_value:nV
1131             { #2 }
1132             \l_tmpa_tl
1133         },
1134         #3 .default:n = { true },
1135     }
1136 }
1137 {
1138     \keys_define:nn
1139     { markdown/options }
1140     {
1141         #3 .code:n = {
1142             \@@_set_option_value:nn
1143             { #2 }
1144             { ##1 }
1145         },
1146     }
1147 }
```

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.

```
1148     \str_if_eq:VVT
1149     \l_tmpa_tl
1150     \c_@@_option_type_clist_tl
1151     {
1152         \tl_set:Nn
1153         \l_tmpa_tl
1154         { #3 }
```

```

1155     \tl_reverse:N
1156     \l_tmpa_tl
1157     \str_if_eq:enF
1158     {
1159         \tl_head:V
1160         \l_tmpa_tl
1161     }
1162     { s }
1163     {
1164         \msg_error:nnn
1165         { markdown }
1166         { malformed-name-for-clist-option }
1167         { #3 }
1168     }
1169     \tl_set:Nx
1170     \l_tmpa_tl
1171     {
1172         \tl_tail:V
1173         \l_tmpa_tl
1174     }
1175     \tl_reverse:N
1176     \l_tmpa_tl
1177     \tl_put_right:Nn
1178     \l_tmpa_tl
1179     {
1180         .code:n = {
1181             \@@_get_option_value:nN
1182             { #2 }
1183             \l_tmpa_tl
1184             \clist_set:NV
1185             \l_tmpa_clist
1186             { \l_tmpa_tl, { ##1 } }
1187             \@@_set_option_value:nV
1188             { #2 }
1189             \l_tmpa_clist
1190         }
1191     }
1192     \keys_define:nV
1193     { markdown/options }
1194     \l_tmpa_tl
1195 }
1196 }
1197 \cs_generate_variant:Nn
1198 \clist_set:Nn
1199 { NV }
1200 \cs_generate_variant:Nn
1201 \keys_define:nn

```

```

1202 { nV }
1203 \cs_generate_variant:Nn
1204 \@@_set_option_value:nn
1205 { nV }
1206 \prg_generate_conditional_variant:Nnn
1207 \str_if_eq:nn
1208 { en }
1209 { p, F }
1210 \msg_new:nnn
1211 { markdown }
1212 { malformed-name-for-clist-option }
1213 {
1214   Clist-option-name~#1~does~not~end~with~-s.
1215 }

```

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

```

1216 \str_if_eq:VVT
1217 \c_@@_top_layer_tl
1218 \c_@@_option_layer_plain_tex_tl
1219 {
1220   \@@_define_option_commands_and_keyvals:
1221 }
1222 \ExplSyntaxOff

```

### 2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a T<sub>E</sub>X document (further referred to as *a theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [9, Section 15.1].

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

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

If `@<theme version>` is specified after `<theme name>`, then the text *theme version* will be available in the macro `\markdownThemeVersion` when the theme is loaded. If `@<theme version>` is not specified, the macro `\markdownThemeVersion` will contain the text `latest` [10].

```

1223 \ExplSyntaxOn
1224 \keys_define:nn
1225   { markdown/options }
1226   {
1227     theme .code:n = {
1228       \@@_set_theme:n
1229       { #1 }
1230     },
1231     import .code:n = {
1232       \tl_set:Nn
1233       \l_tmpa_tl
1234       { #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.

```

1235     \tl_replace_all:NnV
1236     \l_tmpa_tl
1237     { / }
1238     \c_backslash_str
1239     \keys_set:nV
1240     { markdown/options/import }
1241     \l_tmpa_tl
1242   },
1243 }

```

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

```

1244 \seq_new:N
1245 \g_@@_theme_names_seq
1246 \seq_new:N
1247 \g_@@_theme_versions_seq
1248 \tl_new:N

```

```

1249 \g_@@_current_theme_tl
1250 \tl_gset:Nn
1251 \g_@@_current_theme_tl
1252 { }
1253 \seq_gput_right:NV
1254 \g_@@_theme_names_seq
1255 \g_@@_current_theme_tl
1256 \cs_new:Npn
1257 \markdownThemeVersion
1258 { }
1259 \seq_gput_right:NV
1260 \g_@@_theme_versions_seq
1261 \g_@@_current_theme_tl
1262 \cs_new:Nn
1263 \@@_set_theme:n
1264 {

```

First, we validate the theme name.

```

1265 \str_if_in:nnF
1266 { #1 }
1267 { / }
1268 {
1269 \msg_error:nnn
1270 { markdown }
1271 { unqualified-theme-name }
1272 { #1 }
1273 }
1274 \str_if_in:nnT
1275 { #1 }
1276 { _ }
1277 {
1278 \msg_error:nnn
1279 { markdown }
1280 { underscores-in-theme-name }
1281 { #1 }
1282 }

```

Next, we extract the theme version.

```

1283 \str_if_in:nnTF
1284 { #1 }
1285 { @ }
1286 {
1287 \regex_extract_once:nnN
1288 { (.*?) @ (.*?) }
1289 { #1 }
1290 \l_tmpa_seq
1291 \seq_gpop_left:NN
1292 \l_tmpa_seq

```

```

1293     \l_tmpa_tl
1294     \seq_gpop_left:NN
1295     \l_tmpa_seq
1296     \l_tmpa_tl
1297     \tl_gset:NV
1298     \g_@@_current_theme_tl
1299     \l_tmpa_tl
1300     \seq_gpop_left:NN
1301     \l_tmpa_seq
1302     \l_tmpa_tl
1303     \cs_gset:Npe
1304     \markdownThemeVersion
1305     {
1306         \tl_use:N
1307         \l_tmpa_tl
1308     }
1309 }
1310 {
1311     \tl_gset:Nn
1312     \g_@@_current_theme_tl
1313     { #1 }
1314     \cs_gset:Npn
1315     \markdownThemeVersion
1316     { latest }
1317 }

```

Next, we munge the theme name.

```

1318     \str_set:NV
1319     \l_tmpa_str
1320     \g_@@_current_theme_tl
1321     \str_replace_all:Nnn
1322     \l_tmpa_str
1323     { / }
1324     { _ }

```

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

```

1325     \tl_set:NV
1326     \l_tmpa_tl
1327     \g_@@_current_theme_tl
1328     \tl_put_right:Nn
1329     \g_@@_current_theme_tl
1330     { / }
1331     \seq_gput_right:NV
1332     \g_@@_theme_names_seq
1333     \g_@@_current_theme_tl
1334     \seq_gput_right:NV
1335     \g_@@_theme_versions_seq

```

```

1336     \markdownThemeVersion
1337     \@@_load_theme:VeV
1338     \l_tmpa_tl
1339     { \markdownThemeVersion }
1340     \l_tmpa_str

```

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

```

1341     \seq_gpop_right:NN
1342     \g_@@_theme_names_seq
1343     \l_tmpa_tl
1344     \seq_get_right:NN
1345     \g_@@_theme_names_seq
1346     \l_tmpa_tl
1347     \tl_gset:NV
1348     \g_@@_current_theme_tl
1349     \l_tmpa_tl
1350     \seq_gpop_right:NN
1351     \g_@@_theme_versions_seq
1352     \l_tmpa_tl
1353     \seq_get_right:NN
1354     \g_@@_theme_versions_seq
1355     \l_tmpa_tl
1356     \cs_gset:Npe
1357     \markdownThemeVersion
1358     {
1359         \tl_use:N
1360         \l_tmpa_tl
1361     }
1362 }
1363 \msg_new:nnnn
1364 { markdown }
1365 { unqualified-theme-name }
1366 { Won't~load~theme~with~unqualified~name~#1 }
1367 { Theme~names~must~contain~at~least~one~forward~slash }
1368 \msg_new:nnnn
1369 { markdown }
1370 { underscores-in-theme-name }
1371 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1372 { Theme~names~must~not~contain~underscores~in~their~names }
1373 \cs_generate_variant:Nn
1374 \tl_replace_all:Nnn
1375 { NnV }
1376 \cs_generate_variant:Nn
1377 \cs_gset:Npn
1378 { Npe }

```

We also define the prop `\g_@@_plain_tex_built_in_themes_prop` that contains



the code of built-in themes. This is a packaging optimization, so that built-in themes does not need to be distributed in many small files.

```
1379 \prop_new:N
1380 \g_@@_plain_tex_built_in_themes_prop
```

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

**witiko/diagrams@v1** 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/diagrams@v1]{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;

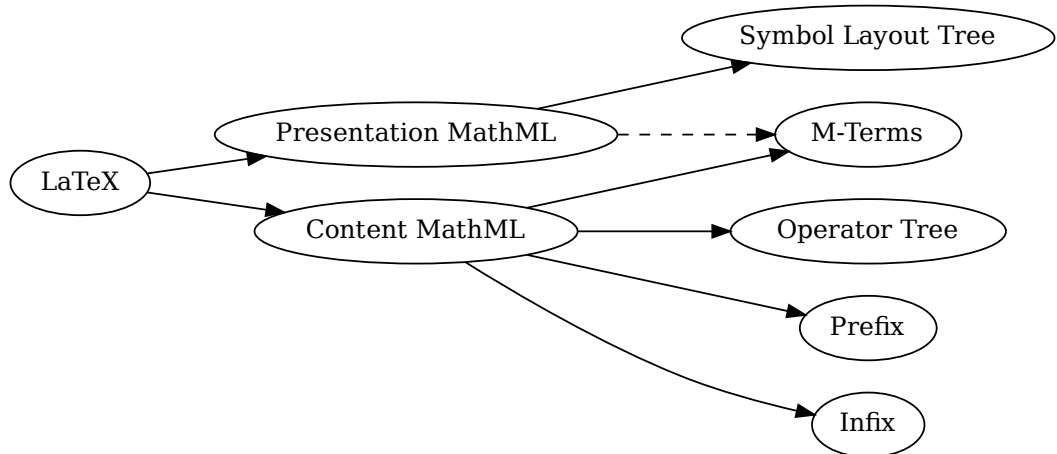
  latex [label = "LaTeX"];
  pmml [label = "Presentation MathML"];
  cmml [label = "Content MathML"];
  slt [label = "Symbol Layout Tree"];
  opt [label = "Operator Tree"];
  prefix [label = "Prefix"];
  infix [label = "Infix"];
  mterms [label = "M-Terms"];
}
...
```

```

\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.

The above example loads version `v1` of the theme, which is an alias for an earlier theme named `witiko/dot`. Future versions of the theme may have backwards-incompatible syntax and behavior. Therefore, you are encouraged to always specify the version `v1` to keep your documents from suddenly breaking.

**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}

```

```

\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**

Typesetting the above document produces the output shown in Figure 5. The theme requires the catchfile  $\LaTeX$  package and a Unix-like operating system with GNU Coreutils `md5sum` and either GNU Wget or cURL installed. The theme also requires shell access unless the `frozenCache` plain  $\TeX$  option is enabled.

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

```

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

```

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

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

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

## 2.2.4 Snippets

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

```
1381 \prop_new:N
1382   \g_@@_snippets_prop
1383 \cs_new:Nn
1384   \@@_setup_snippet:nn
1385   {
1386     \tl_if_empty:nT
1387       { #1 }
1388       {
1389         \msg_error:nnn
1390           { markdown }
1391           { empty-snippet-name }
1392           { #1 }
1393       }
1394     \tl_set:NV
1395       \l_tmpa_tl
1396       \g_@@_current_theme_tl
1397     \tl_put_right:Nn
1398       \l_tmpa_tl
1399       { #1 }
1400     \@@_if_snippet_exists:nT
1401       { #1 }
1402       {
1403         \msg_warning:nnV
1404           { markdown }
1405           { redefined-snippet }
1406           \l_tmpa_tl
1407       }
1408     \keys_precompile:nnN
1409       { markdown/options }
1410       { #2 }
1411       \l_tmpb_tl
1412     \prop_gput:NVV
1413       \g_@@_snippets_prop
1414       \l_tmpa_tl
1415       \l_tmpb_tl
1416   }
1417 \cs_gset_eq:NN
1418   \markdownSetupSnippet
1419   \@@_setup_snippet:nn
```

```

1420 \msg_new:nnnn
1421   { markdown }
1422   { empty-snippet-name }
1423   { Empty~snippet~name~#1 }
1424   { Pick~a~non~empty~name~for~your~snippet }
1425 \msg_new:nnn
1426   { markdown }
1427   { redefined-snippet }
1428   { Redefined~snippet~#1 }

```

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

```

1429 \tl_new:N
1430   \l_@@_current_snippet_tl
1431 \prg_new_conditional:Nnn
1432   \@@_if_snippet_exists:n
1433   { TF, T, F }
1434   {
1435     \tl_set:NV
1436       \l_@@_current_snippet_tl
1437       \g_@@_current_theme_tl
1438     \tl_put_right:Nn
1439       \l_@@_current_snippet_tl
1440       { #1 }
1441     \prop_if_in:NVTF
1442       \g_@@_snippets_prop
1443       \l_@@_current_snippet_tl
1444       { \prg_return_true: }
1445       { \prg_return_false: }
1446   }
1447 \cs_gset_eq:NN
1448   \markdownIfSnippetExists
1449   \@@_if_snippet_exists:nTF

```

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

```

1450 \keys_define:nn
1451   { markdown/options }
1452   {
1453     snippet .code:n = {
1454       \tl_set:NV
1455         \l_tmpa_tl
1456         \g_@@_current_theme_tl
1457       \tl_put_right:Nn
1458         \l_tmpa_tl
1459         { #1 }
1460       \@@_if_snippet_exists:nTF
1461         { #1 }
1462     }

```

```

1463     \prop_get:NVN
1464     \g_@@_snippets_prop
1465     \l_tmpa_tl
1466     \l_tmpb_tl
1467     \tl_use:N
1468     \l_tmpb_tl
1469   }
1470   {
1471     \msg_error:nnV
1472     { markdown }
1473     { undefined-snippet }
1474     \l_tmpa_tl
1475   }
1476 }
1477 }
1478 \msg_new:nnn
1479 { markdown }
1480 { undefined-snippet }
1481 { Can't~invoke~undefined~snippet~#1 }
1482 \ExplSyntaxOff

```

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

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```

\end{markdown}

```

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

```
\markdownSetup{import=jdooe/lists}
\begin{markdown}[snippet=jdooe/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 = {
    jdooe/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 `jdooe/lists` theme. For example, we can make the snippet `jdooe/lists/romanNumerals` available under the name `roman`.

```
\markdownSetup{
  import = {
    jdooe/lists = romanNumerals as roman,
  },
}
```

```
\begin{markdown}[snippet=roman]
```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

```
1483 \ExplSyntaxOn
1484 \tl_new:N
1485 \l_@@_import_current_theme_tl
1486 \keys_define:nn
1487 { markdown/options/import }
1488 {
```

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

```
1489   unknown .default:n = {},
1490   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.



```

1491     \tl_set_eq:NN
1492     \l_@@_import_current_theme_tl
1493     \l_keys_key_str
1494     \tl_replace_all:NVN
1495     \l_@@_import_current_theme_tl
1496     \c_backslash_str
1497     { / }

```

Here, we import the snippets.

```

1498     \clist_map_inline:nn
1499     { #1 }
1500     {
1501         \regex_extract_once:nnNTF
1502         { ^(.*)\s+as\s+(.*)$ }
1503         { ##1 }
1504         \l_tmpa_seq
1505         {
1506             \seq_pop:NN
1507             \l_tmpa_seq
1508             \l_tmpa_tl
1509             \seq_pop:NN
1510             \l_tmpa_seq
1511             \l_tmpa_tl
1512             \seq_pop:NN
1513             \l_tmpa_seq
1514             \l_tmpb_tl
1515         }
1516         {
1517             \tl_set:Nn
1518             \l_tmpa_tl
1519             { ##1 }
1520             \tl_set:Nn
1521             \l_tmpb_tl
1522             { ##1 }
1523         }
1524         \tl_put_left:Nn
1525         \l_tmpa_tl
1526         { / }
1527         \tl_put_left:NV
1528         \l_tmpa_tl
1529         \l_@@_import_current_theme_tl
1530         \@@_setup_snippet:Vx
1531         \l_tmpb_tl
1532         { snippet = { \l_tmpa_tl } }
1533     }

```

Here, we load the theme.

```

1534     \@@_set_theme:V

```

```

1535     \l_@@_import_current_theme_tl
1536   },
1537 }
1538 \cs_generate_variant:Nn
1539   \tl_replace_all:Nnn
1540   { NVn }
1541 \cs_generate_variant:Nn
1542   \@@_set_theme:n
1543   { V }
1544 \cs_generate_variant:Nn
1545   \@@_setup_snippet:nn
1546   { Vx }

```

## 2.2.5 Token Renderers

The following T<sub>E</sub>X macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.6).

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

```
1547 \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.

```
1548 \prop_new:N \g_@@_renderer_arities_prop
1549 \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  $\langle class name \rangle$ .

$\backslash\text{markdownRendererAttributeKeyValue}$  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.

```

1550 \ExplSyntaxOn
1551 \cs_gset_protected:Npn
1552   \markdownRendererAttributeIdentifier
1553   {
1554     \markdownRendererAttributeIdentifierPrototype
1555   }
1556 \seq_gput_right:Nn
1557   \g_@@_renderers_seq
1558   { attributeIdentifier }
1559 \prop_gput:Nnn
1560   \g_@@_renderer_arities_prop
1561   { attributeIdentifier }
1562   { 1 }
1563 \cs_gset_protected:Npn
1564   \markdownRendererAttributeClassName
1565   {
1566     \markdownRendererAttributeClassNamePrototype
1567   }
1568 \seq_gput_right:Nn
1569   \g_@@_renderers_seq
1570   { attributeClassName }
1571 \prop_gput:Nnn
1572   \g_@@_renderer_arities_prop
1573   { attributeClassName }
1574   { 1 }
1575 \cs_gset_protected:Npn
1576   \markdownRendererAttributeKeyValue
1577   {
1578     \markdownRendererAttributeKeyValuePrototype
1579   }
1580 \seq_gput_right:Nn
1581   \g_@@_renderers_seq
1582   { attributeKeyValue }
1583 \prop_gput:Nnn
1584   \g_@@_renderer_arities_prop
1585   { attributeKeyValue }
1586   { 2 }
1587 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```
1588 \ExplSyntaxOn
1589 \cs_gset_protected:Npn
1590   \markdownRendererBlockQuoteBegin
1591   {
1592     \markdownRendererBlockQuoteBeginPrototype
1593   }
1594 \seq_gput_right:Nn
1595   \g_@@_renderers_seq
1596   { blockQuoteBegin }
1597 \prop_gput:Nnn
1598   \g_@@_renderer_arities_prop
1599   { blockQuoteBegin }
1600   { 0 }
1601 \ExplSyntaxOff
```

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

```
1602 \ExplSyntaxOn
1603 \cs_gset_protected:Npn
1604   \markdownRendererBlockQuoteEnd
1605   {
1606     \markdownRendererBlockQuoteEndPrototype
1607   }
1608 \seq_gput_right:Nn
1609   \g_@@_renderers_seq
1610   { blockQuoteEnd }
1611 \prop_gput:Nnn
1612   \g_@@_renderer_arities_prop
1613   { blockQuoteEnd }
1614   { 0 }
1615 \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.

```
1616 \ExplSyntaxOn
1617 \cs_gset_protected:Npn
1618   \markdownRendererBracketedSpanAttributeContextBegin
1619   {
1620     \markdownRendererBracketedSpanAttributeContextBeginPrototype
```

```

1621 }
1622 \seq_gput_right:Nn
1623 \g_@@_renderers_seq
1624 { bracketedSpanAttributeContextBegin }
1625 \prop_gput:Nnn
1626 \g_@@_renderer_arities_prop
1627 { bracketedSpanAttributeContextBegin }
1628 { 0 }
1629 \cs_gset_protected:Npn
1630 \markdownRendererBracketedSpanAttributeContextEnd
1631 {
1632   \markdownRendererBracketedSpanAttributeContextEndPrototype
1633 }
1634 \seq_gput_right:Nn
1635 \g_@@_renderers_seq
1636 { bracketedSpanAttributeContextEnd }
1637 \prop_gput:Nnn
1638 \g_@@_renderer_arities_prop
1639 { bracketedSpanAttributeContextEnd }
1640 { 0 }
1641 \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.

```

1642 \ExplSyntaxOn
1643 \cs_gset_protected:Npn
1644 \markdownRendererUlBegin
1645 {
1646   \markdownRendererUlBeginPrototype
1647 }
1648 \seq_gput_right:Nn
1649 \g_@@_renderers_seq
1650 { ulBegin }
1651 \prop_gput:Nnn
1652 \g_@@_renderer_arities_prop
1653 { ulBegin }
1654 { 0 }
1655 \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.

```

1656 \ExplSyntaxOn
1657 \cs_gset_protected:Npn
1658   \markdownRendererUlBeginTight
1659   {
1660     \markdownRendererUlBeginTightPrototype
1661   }
1662 \seq_gput_right:Nn
1663   \g_@@_renderers_seq
1664   { ulBeginTight }
1665 \prop_gput:Nnn
1666   \g_@@_renderer_arities_prop
1667   { ulBeginTight }
1668   { 0 }
1669 \ExplSyntaxOff

```

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

```

1670 \ExplSyntaxOn
1671 \cs_gset_protected:Npn
1672   \markdownRendererUlItem
1673   {
1674     \markdownRendererUlItemPrototype
1675   }
1676 \seq_gput_right:Nn
1677   \g_@@_renderers_seq
1678   { ulItem }
1679 \prop_gput:Nnn
1680   \g_@@_renderer_arities_prop
1681   { ulItem }
1682   { 0 }
1683 \ExplSyntaxOff

```

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

```

1684 \ExplSyntaxOn
1685 \cs_gset_protected:Npn
1686   \markdownRendererUlItemEnd
1687   {
1688     \markdownRendererUlItemEndPrototype
1689   }
1690 \seq_gput_right:Nn
1691   \g_@@_renderers_seq
1692   { ulItemEnd }
1693 \prop_gput:Nnn
1694   \g_@@_renderer_arities_prop
1695   { ulItemEnd }
1696   { 0 }

```

```
1697 \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.

```
1698 \ExplSyntaxOn
1699 \cs_gset_protected:Npn
1700   \markdownRendererUEnd
1701   {
1702     \markdownRendererUEndPrototype
1703   }
1704 \seq_gput_right:Nn
1705   \g_@@_renderers_seq
1706   { ulEnd }
1707 \prop_gput:Nnn
1708   \g_@@_renderer_arities_prop
1709   { ulEnd }
1710   { 0 }
1711 \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.

```
1712 \ExplSyntaxOn
1713 \cs_gset_protected:Npn
1714   \markdownRendererUEndTight
1715   {
1716     \markdownRendererUEndTightPrototype
1717   }
1718 \seq_gput_right:Nn
1719   \g_@@_renderers_seq
1720   { ulEndTight }
1721 \prop_gput:Nnn
1722   \g_@@_renderer_arities_prop
1723   { ulEndTight }
1724   { 0 }
1725 \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  $\langle suppress author \rangle$  parameter is either the token `-`, when the author's name is to be suppressed, or `+` otherwise.

```

1726 \ExplSyntaxOn
1727 \cs_gset_protected:Npn
1728   \markdownRendererCite
1729   {
1730     \markdownRendererCitePrototype
1731   }
1732 \seq_gput_right:Nn
1733   \g_@@_renderers_seq
1734   { cite }
1735 \prop_gput:Nnn
1736   \g_@@_renderer_arities_prop
1737   { cite }
1738   { 1 }
1739 \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.

```

1740 \ExplSyntaxOn
1741 \cs_gset_protected:Npn
1742   \markdownRendererTextCite
1743   {
1744     \markdownRendererTextCitePrototype
1745   }
1746 \seq_gput_right:Nn
1747   \g_@@_renderers_seq
1748   { textCite }
1749 \prop_gput:Nnn
1750   \g_@@_renderer_arities_prop
1751   { textCite }
1752   { 1 }
1753 \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.

```

1754 \ExplSyntaxOn
1755 \cs_gset_protected:Npn
1756   \markdownRendererInputVerbatim
1757   {
1758     \markdownRendererInputVerbatimPrototype

```



```

1759 }
1760 \seq_gput_right:Nn
1761 \g_@@_renderers_seq
1762 { inputVerbatim }
1763 \prop_gput:Nnn
1764 \g_@@_renderer_arities_prop
1765 { inputVerbatim }
1766 { 1 }
1767 \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.

```

1768 \ExplSyntaxOn
1769 \cs_gset_protected:Npn
1770 \markdownRendererInputFencedCode
1771 {
1772   \markdownRendererInputFencedCodePrototype
1773 }
1774 \seq_gput_right:Nn
1775 \g_@@_renderers_seq
1776 { inputFencedCode }
1777 \prop_gput:Nnn
1778 \g_@@_renderer_arities_prop
1779 { inputFencedCode }
1780 { 3 }
1781 \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.

```

1782 \ExplSyntaxOn
1783 \cs_gset_protected:Npn
1784 \markdownRendererCodeSpan
1785 {
1786   \markdownRendererCodeSpanPrototype
1787 }
1788 \seq_gput_right:Nn
1789 \g_@@_renderers_seq
1790 { codeSpan }
1791 \prop_gput:Nnn
1792 \g_@@_renderer_arities_prop
1793 { codeSpan }

```

```

1794 { 1 }
1795 \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.

```

1796 \ExplSyntaxOn
1797 \cs_gset_protected:Npn
1798   \markdownRendererCodeSpanAttributeContextBegin
1799   {
1800     \markdownRendererCodeSpanAttributeContextBeginPrototype
1801   }
1802 \seq_gput_right:Nn
1803   \g_@@_renderers_seq
1804   { codeSpanAttributeContextBegin }
1805 \prop_gput:Nnn
1806   \g_@@_renderer_arities_prop
1807   { codeSpanAttributeContextBegin }
1808   { 0 }
1809 \cs_gset_protected:Npn
1810   \markdownRendererCodeSpanAttributeContextEnd
1811   {
1812     \markdownRendererCodeSpanAttributeContextEndPrototype
1813   }
1814 \seq_gput_right:Nn
1815   \g_@@_renderers_seq
1816   { codeSpanAttributeContextEnd }
1817 \prop_gput:Nnn
1818   \g_@@_renderer_arities_prop
1819   { codeSpanAttributeContextEnd }
1820   { 0 }
1821 \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.

```

1822 \ExplSyntaxOn
1823 \cs_gset_protected:Npn
1824   \markdownRendererContentBlock

```

```

1825 {
1826   \markdownRendererContentBlockPrototype
1827 }
1828 \seq_gput_right:Nn
1829   \g_@@_renderers_seq
1830   { contentBlock }
1831 \prop_gput:Nnn
1832   \g_@@_renderer_arities_prop
1833   { contentBlock }
1834   { 4 }
1835 \ExplSyntaxOff

```

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

```

1836 \ExplSyntaxOn
1837 \cs_gset_protected:Npn
1838   \markdownRendererContentBlockOnlineImage
1839   {
1840     \markdownRendererContentBlockOnlineImagePrototype
1841   }
1842 \seq_gput_right:Nn
1843   \g_@@_renderers_seq
1844   { contentBlockOnlineImage }
1845 \prop_gput:Nnn
1846   \g_@@_renderer_arities_prop
1847   { contentBlockOnlineImage }
1848   { 4 }
1849 \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>32</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s, s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local TeX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses

---

<sup>32</sup>Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

beside syntax highlighting. The `Languages.json` file provided by Sotkov [4] is a good starting point.

```
1850 \ExplSyntaxOn
1851 \cs_gset_protected:Npn
1852   \markdownRendererContentBlockCode
1853   {
1854     \markdownRendererContentBlockCodePrototype
1855   }
1856 \seq_gput_right:Nn
1857   \g_@@_renderers_seq
1858   { contentBlockCode }
1859 \prop_gput:Nnn
1860   \g_@@_renderer_arities_prop
1861   { contentBlockCode }
1862   { 5 }
1863 \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.

```
1864 \ExplSyntaxOn
1865 \cs_gset_protected:Npn
1866   \markdownRendererDlBegin
1867   {
1868     \markdownRendererDlBeginPrototype
1869   }
1870 \seq_gput_right:Nn
1871   \g_@@_renderers_seq
1872   { dlBegin }
1873 \prop_gput:Nnn
1874   \g_@@_renderer_arities_prop
1875   { dlBegin }
1876   { 0 }
1877 \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.

```
1878 \ExplSyntaxOn
1879 \cs_gset_protected:Npn
1880   \markdownRendererDlBeginTight
```

```

1881 {
1882   \markdownRendererDlBeginTightPrototype
1883 }
1884 \seq_gput_right:Nn
1885   \g_@@_renderers_seq
1886   { dlBeginTight }
1887 \prop_gput:Nnn
1888   \g_@@_renderer_arities_prop
1889   { dlBeginTight }
1890   { 0 }
1891 \ExplSyntaxOff

```

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

```

1892 \ExplSyntaxOn
1893 \cs_gset_protected:Npn
1894   \markdownRendererDlItem
1895   {
1896     \markdownRendererDlItemPrototype
1897   }
1898 \seq_gput_right:Nn
1899   \g_@@_renderers_seq
1900   { dlItem }
1901 \prop_gput:Nnn
1902   \g_@@_renderer_arities_prop
1903   { dlItem }
1904   { 1 }
1905 \ExplSyntaxOff

```

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

```

1906 \ExplSyntaxOn
1907 \cs_gset_protected:Npn
1908   \markdownRendererDlItemEnd
1909   {
1910     \markdownRendererDlItemEndPrototype
1911   }
1912 \seq_gput_right:Nn
1913   \g_@@_renderers_seq
1914   { dlItemEnd }
1915 \prop_gput:Nnn
1916   \g_@@_renderer_arities_prop
1917   { dlItemEnd }
1918   { 0 }
1919 \ExplSyntaxOff

```

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

```

1920 \ExplSyntaxOn
1921 \cs_gset_protected:Npn
1922   \markdownRendererDlDefinitionBegin
1923   {
1924     \markdownRendererDlDefinitionBeginPrototype
1925   }
1926 \seq_gput_right:Nn
1927   \g_@@_renderers_seq
1928   { dlDefinitionBegin }
1929 \prop_gput:Nnn
1930   \g_@@_renderer_arities_prop
1931   { dlDefinitionBegin }
1932   { 0 }
1933 \ExplSyntaxOff

```

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

```

1934 \ExplSyntaxOn
1935 \cs_gset_protected:Npn
1936   \markdownRendererDlDefinitionEnd
1937   {
1938     \markdownRendererDlDefinitionEndPrototype
1939   }
1940 \seq_gput_right:Nn
1941   \g_@@_renderers_seq
1942   { dlDefinitionEnd }
1943 \prop_gput:Nnn
1944   \g_@@_renderer_arities_prop
1945   { dlDefinitionEnd }
1946   { 0 }
1947 \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.

```

1948 \ExplSyntaxOn
1949 \cs_gset_protected:Npn
1950   \markdownRendererDlEnd
1951   {
1952     \markdownRendererDlEndPrototype
1953   }
1954 \seq_gput_right:Nn
1955   \g_@@_renderers_seq
1956   { dlEnd }

```

```

1957 \prop_gput:Nnn
1958   \g_@@_renderer_arities_prop
1959   { dlEnd }
1960   { 0 }
1961 \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.

```

1962 \ExplSyntaxOn
1963 \cs_gset_protected:Npn
1964   \markdownRendererDlEndTight
1965   {
1966     \markdownRendererDlEndTightPrototype
1967   }
1968 \seq_gput_right:Nn
1969   \g_@@_renderers_seq
1970   { dlEndTight }
1971 \prop_gput:Nnn
1972   \g_@@_renderer_arities_prop
1973   { dlEndTight }
1974   { 0 }
1975 \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.

```

1976 \ExplSyntaxOn
1977 \cs_gset_protected:Npn
1978   \markdownRendererEllipsis
1979   {
1980     \markdownRendererEllipsisPrototype
1981   }
1982 \seq_gput_right:Nn
1983   \g_@@_renderers_seq
1984   { ellipsis }
1985 \prop_gput:Nnn
1986   \g_@@_renderer_arities_prop
1987   { ellipsis }
1988   { 0 }
1989 \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.

```

1990 \ExplSyntaxOn
1991 \cs_gset_protected:Npn
1992   \markdownRendererEmphasis
1993   {
1994     \markdownRendererEmphasisPrototype
1995   }
1996 \seq_gput_right:Nn
1997   \g_@@_renderers_seq
1998   { emphasis }
1999 \prop_gput:Nnn
2000   \g_@@_renderer_arities_prop
2001   { emphasis }
2002   { 1 }
2003 \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.

```

2004 \ExplSyntaxOn
2005 \cs_gset_protected:Npn
2006   \markdownRendererStrongEmphasis
2007   {
2008     \markdownRendererStrongEmphasisPrototype
2009   }
2010 \seq_gput_right:Nn
2011   \g_@@_renderers_seq
2012   { strongEmphasis }
2013 \prop_gput:Nnn
2014   \g_@@_renderer_arities_prop
2015   { strongEmphasis }
2016   { 1 }
2017 \ExplSyntaxOff

```

### 2.2.5.13 Fenced Code Attribute Context Renderers

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

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

```

2018 \ExplSyntaxOn
2019 \cs_gset_protected:Npn
2020   \markdownRendererFencedCodeAttributeContextBegin

```



```

2021 {
2022   \markdownRendererFencedCodeAttributeContextBeginPrototype
2023 }
2024 \seq_gput_right:Nn
2025   \g_@@_renderers_seq
2026   { fencedCodeAttributeContextBegin }
2027 \prop_gput:Nnn
2028   \g_@@_renderer_arities_prop
2029   { fencedCodeAttributeContextBegin }
2030   { 0 }
2031 \cs_gset_protected:Npn
2032   \markdownRendererFencedCodeAttributeContextEnd
2033   {
2034     \markdownRendererFencedCodeAttributeContextEndPrototype
2035   }
2036 \seq_gput_right:Nn
2037   \g_@@_renderers_seq
2038   { fencedCodeAttributeContextEnd }
2039 \prop_gput:Nnn
2040   \g_@@_renderer_arities_prop
2041   { fencedCodeAttributeContextEnd }
2042   { 0 }
2043 \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.

```

2044 \ExplSyntaxOn
2045 \cs_gset_protected:Npn
2046   \markdownRendererFencedDivAttributeContextBegin
2047   {
2048     \markdownRendererFencedDivAttributeContextBeginPrototype
2049   }
2050 \seq_gput_right:Nn
2051   \g_@@_renderers_seq
2052   { fencedDivAttributeContextBegin }
2053 \prop_gput:Nnn
2054   \g_@@_renderer_arities_prop
2055   { fencedDivAttributeContextBegin }
2056   { 0 }
2057 \cs_gset_protected:Npn
2058   \markdownRendererFencedDivAttributeContextEnd
2059   {
2060     \markdownRendererFencedDivAttributeContextEndPrototype

```

```

2061 }
2062 \seq_gput_right:Nn
2063 \g_@@_renderers_seq
2064 { fencedDivAttributeContextEnd }
2065 \prop_gput:Nnn
2066 \g_@@_renderer_arities_prop
2067 { fencedDivAttributeContextEnd }
2068 { 0 }
2069 \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.

```

2070 \ExplSyntaxOn
2071 \cs_gset_protected:Npn
2072 \markdownRendererHeaderAttributeContextBegin
2073 {
2074 \markdownRendererHeaderAttributeContextBeginPrototype
2075 }
2076 \seq_gput_right:Nn
2077 \g_@@_renderers_seq
2078 { headerAttributeContextBegin }
2079 \prop_gput:Nnn
2080 \g_@@_renderer_arities_prop
2081 { headerAttributeContextBegin }
2082 { 0 }
2083 \cs_gset_protected:Npn
2084 \markdownRendererHeaderAttributeContextEnd
2085 {
2086 \markdownRendererHeaderAttributeContextEndPrototype
2087 }
2088 \seq_gput_right:Nn
2089 \g_@@_renderers_seq
2090 { headerAttributeContextEnd }
2091 \prop_gput:Nnn
2092 \g_@@_renderer_arities_prop
2093 { headerAttributeContextEnd }
2094 { 0 }
2095 \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.

```

2096 \ExplSyntaxOn
2097 \cs_gset_protected:Npn
2098   \markdownRendererHeadingOne
2099   {
2100     \markdownRendererHeadingOnePrototype
2101   }
2102 \seq_gput_right:Nn
2103   \g_@@_renderers_seq
2104   { headingOne }
2105 \prop_gput:Nnn
2106   \g_@@_renderer_arities_prop
2107   { headingOne }
2108   { 1 }
2109 \ExplSyntaxOff

```

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

```

2110 \ExplSyntaxOn
2111 \cs_gset_protected:Npn
2112   \markdownRendererHeadingTwo
2113   {
2114     \markdownRendererHeadingTwoPrototype
2115   }
2116 \seq_gput_right:Nn
2117   \g_@@_renderers_seq
2118   { headingTwo }
2119 \prop_gput:Nnn
2120   \g_@@_renderer_arities_prop
2121   { headingTwo }
2122   { 1 }
2123 \ExplSyntaxOff

```

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

```

2124 \ExplSyntaxOn
2125 \cs_gset_protected:Npn
2126   \markdownRendererHeadingThree
2127   {
2128     \markdownRendererHeadingThreePrototype
2129   }
2130 \seq_gput_right:Nn
2131   \g_@@_renderers_seq
2132   { headingThree }
2133 \prop_gput:Nnn

```

```

2134 \g_@@_renderer_arities_prop
2135 { headingThree }
2136 { 1 }
2137 \ExplSyntaxOff

```

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

```

2138 \ExplSyntaxOn
2139 \cs_gset_protected:Npn
2140 \markdownRendererHeadingFour
2141 {
2142   \markdownRendererHeadingFourPrototype
2143 }
2144 \seq_gput_right:Nn
2145 \g_@@_renderers_seq
2146 { headingFour }
2147 \prop_gput:Nnn
2148 \g_@@_renderer_arities_prop
2149 { headingFour }
2150 { 1 }
2151 \ExplSyntaxOff

```

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

```

2152 \ExplSyntaxOn
2153 \cs_gset_protected:Npn
2154 \markdownRendererHeadingFive
2155 {
2156   \markdownRendererHeadingFivePrototype
2157 }
2158 \seq_gput_right:Nn
2159 \g_@@_renderers_seq
2160 { headingFive }
2161 \prop_gput:Nnn
2162 \g_@@_renderer_arities_prop
2163 { headingFive }
2164 { 1 }
2165 \ExplSyntaxOff

```

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

```

2166 \ExplSyntaxOn
2167 \cs_gset_protected:Npn
2168 \markdownRendererHeadingSix
2169 {
2170   \markdownRendererHeadingSixPrototype
2171 }

```

```

2172 \seq_gput_right:Nn
2173   \g_@@_renderers_seq
2174   { headingSix }
2175 \prop_gput:Nnn
2176   \g_@@_renderer_arities_prop
2177   { headingSix }
2178   { 1 }
2179 \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.

```

2180 \ExplSyntaxOn
2181 \cs_gset_protected:Npn
2182   \markdownRendererInlineHtmlComment
2183   {
2184     \markdownRendererInlineHtmlCommentPrototype
2185   }
2186 \seq_gput_right:Nn
2187   \g_@@_renderers_seq
2188   { inlineHtmlComment }
2189 \prop_gput:Nnn
2190   \g_@@_renderer_arities_prop
2191   { inlineHtmlComment }
2192   { 1 }
2193 \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.

```

2194 \ExplSyntaxOn
2195 \cs_gset_protected:Npn
2196   \markdownRendererInlineHtmlTag
2197   {
2198     \markdownRendererInlineHtmlTagPrototype
2199   }

```

```

2200 \seq_gput_right:Nn
2201   \g_@@_renderers_seq
2202   { inlineHtmlTag }
2203 \prop_gput:Nnn
2204   \g_@@_renderer_arities_prop
2205   { inlineHtmlTag }
2206   { 1 }
2207 \cs_gset_protected:Npn
2208   \markdownRendererInputBlockHtmlElement
2209   {
2210     \markdownRendererInputBlockHtmlElementPrototype
2211   }
2212 \seq_gput_right:Nn
2213   \g_@@_renderers_seq
2214   { inputBlockHtmlElement }
2215 \prop_gput:Nnn
2216   \g_@@_renderer_arities_prop
2217   { inputBlockHtmlElement }
2218   { 1 }
2219 \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.

```

2220 \ExplSyntaxOn
2221 \cs_gset_protected:Npn
2222   \markdownRendererImage
2223   {
2224     \markdownRendererImagePrototype
2225   }
2226 \seq_gput_right:Nn
2227   \g_@@_renderers_seq
2228   { image }
2229 \prop_gput:Nnn
2230   \g_@@_renderer_arities_prop
2231   { image }
2232   { 4 }
2233 \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.

```

2234 \ExplSyntaxOn
2235 \cs_gset_protected:Npn
2236   \markdownRendererImageAttributeContextBegin
2237   {
2238     \markdownRendererImageAttributeContextBeginPrototype
2239   }
2240 \seq_gput_right:Nn
2241   \g_@@_renderers_seq
2242   { imageAttributeContextBegin }
2243 \prop_gput:Nnn
2244   \g_@@_renderer_arities_prop
2245   { imageAttributeContextBegin }
2246   { 0 }
2247 \cs_gset_protected:Npn
2248   \markdownRendererImageAttributeContextEnd
2249   {
2250     \markdownRendererImageAttributeContextEndPrototype
2251   }
2252 \seq_gput_right:Nn
2253   \g_@@_renderers_seq
2254   { imageAttributeContextEnd }
2255 \prop_gput:Nnn
2256   \g_@@_renderer_arities_prop
2257   { imageAttributeContextEnd }
2258   { 0 }
2259 \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.

```

2260 \ExplSyntaxOn
2261 \cs_gset_protected:Npn
2262   \markdownRendererInterblockSeparator
2263   {
2264     \markdownRendererInterblockSeparatorPrototype
2265   }
2266 \seq_gput_right:Nn
2267   \g_@@_renderers_seq
2268   { interblockSeparator }
2269 \prop_gput:Nnn
2270   \g_@@_renderer_arities_prop
2271   { interblockSeparator }

```

```

2272 { 0 }
2273 \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.

```

2274 \ExplSyntaxOn
2275 \cs_gset_protected:Npn
2276   \markdownRendererParagraphSeparator
2277   {
2278     \markdownRendererParagraphSeparatorPrototype
2279   }
2280 \seq_gput_right:Nn
2281   \g_@@_renderers_seq
2282   { paragraphSeparator }
2283 \prop_gput:Nnn
2284   \g_@@_renderer_arities_prop
2285   { paragraphSeparator }
2286   { 0 }
2287 \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.

```

2288 \ExplSyntaxOn
2289 \cs_gset_protected:Npn
2290   \markdownRendererLineBlockBegin
2291   {
2292     \markdownRendererLineBlockBeginPrototype
2293   }
2294 \seq_gput_right:Nn
2295   \g_@@_renderers_seq
2296   { lineBlockBegin }
2297 \prop_gput:Nnn
2298   \g_@@_renderer_arities_prop
2299   { lineBlockBegin }
2300   { 0 }
2301 \cs_gset_protected:Npn
2302   \markdownRendererLineBlockEnd
2303   {

```



```

2304     \markdownRendererLineBlockEndPrototype
2305   }
2306 \seq_gput_right:Nn
2307   \g_@@_renderers_seq
2308   { lineBlockEnd }
2309 \prop_gput:Nnn
2310   \g_@@_renderer_arities_prop
2311   { lineBlockEnd }
2312   { 0 }
2313 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2314 \ExplSyntaxOn
2315 \cs_gset_protected:Npn
2316   \markdownRendererSoftLineBreak
2317   {
2318     \markdownRendererSoftLineBreakPrototype
2319   }
2320 \seq_gput_right:Nn
2321   \g_@@_renderers_seq
2322   { softLineBreak }
2323 \prop_gput:Nnn
2324   \g_@@_renderer_arities_prop
2325   { softLineBreak }
2326   { 0 }
2327 \ExplSyntaxOff

```

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

```

2328 \ExplSyntaxOn
2329 \cs_gset_protected:Npn
2330   \markdownRendererHardLineBreak
2331   {
2332     \markdownRendererHardLineBreakPrototype
2333   }
2334 \seq_gput_right:Nn
2335   \g_@@_renderers_seq
2336   { hardLineBreak }
2337 \prop_gput:Nnn
2338   \g_@@_renderer_arities_prop
2339   { hardLineBreak }
2340   { 0 }
2341 \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.

```
2342 \ExplSyntaxOn
2343 \cs_gset_protected:Npn
2344   \markdownRendererLink
2345   {
2346     \markdownRendererLinkPrototype
2347   }
2348 \seq_gput_right:Nn
2349   \g_@@_renderers_seq
2350   { link }
2351 \prop_gput:Nnn
2352   \g_@@_renderer_arities_prop
2353   { link }
2354   { 4 }
2355 \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.

```
2356 \ExplSyntaxOn
2357 \cs_gset_protected:Npn
2358   \markdownRendererLinkAttributeContextBegin
2359   {
2360     \markdownRendererLinkAttributeContextBeginPrototype
2361   }
2362 \seq_gput_right:Nn
2363   \g_@@_renderers_seq
2364   { linkAttributeContextBegin }
2365 \prop_gput:Nnn
2366   \g_@@_renderer_arities_prop
2367   { linkAttributeContextBegin }
2368   { 0 }
2369 \cs_gset_protected:Npn
2370   \markdownRendererLinkAttributeContextEnd
2371   {
2372     \markdownRendererLinkAttributeContextEndPrototype
2373   }
2374 \seq_gput_right:Nn
2375   \g_@@_renderers_seq
```

```

2376 { linkAttributeContextEnd }
2377 \prop_gput:Nnn
2378 \g_@@_renderer_arities_prop
2379 { linkAttributeContextEnd }
2380 { 0 }
2381 \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.

```

2382 \ExplSyntaxOn
2383 \cs_gset_protected:Npn
2384 \markdownRendererMark
2385 {
2386 \markdownRendererMarkPrototype
2387 }
2388 \seq_gput_right:Nn
2389 \g_@@_renderers_seq
2390 { mark }
2391 \prop_gput:Nnn
2392 \g_@@_renderer_arities_prop
2393 { mark }
2394 { 1 }
2395 \ExplSyntaxOff

```

### 2.2.5.27 Markdown Document Renderers

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

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

```

2396 \ExplSyntaxOn
2397 \cs_gset_protected:Npn
2398 \markdownRendererDocumentBegin
2399 {
2400 \markdownRendererDocumentBeginPrototype
2401 }
2402 \seq_gput_right:Nn
2403 \g_@@_renderers_seq
2404 { documentBegin }
2405 \prop_gput:Nnn

```

```

2406 \g_@@_renderer_arities_prop
2407 { documentBegin }
2408 { 0 }
2409 \cs_gset_protected:Npn
2410 \markdownRendererDocumentEnd
2411 {
2412   \markdownRendererDocumentEndPrototype
2413 }
2414 \seq_gput_right:Nn
2415 \g_@@_renderers_seq
2416 { documentEnd }
2417 \prop_gput:Nnn
2418 \g_@@_renderer_arities_prop
2419 { documentEnd }
2420 { 0 }
2421 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2422 \ExplSyntaxOn
2423 \cs_gset_protected:Npn
2424 \markdownRendererNbsp
2425 {
2426   \markdownRendererNbspPrototype
2427 }
2428 \seq_gput_right:Nn
2429 \g_@@_renderers_seq
2430 { nbsp }
2431 \prop_gput:Nnn
2432 \g_@@_renderer_arities_prop
2433 { nbsp }
2434 { 0 }
2435 \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.

```

2436 \def\markdownRendererNote{%
2437   \markdownRendererNotePrototype}%
2438 \ExplSyntaxOn
2439 \seq_gput_right:Nn
2440 \g_@@_renderers_seq
2441 { note }
2442 \prop_gput:Nnn

```

```

2443 \g_@@_renderer_arities_prop
2444 { note }
2445 { 1 }
2446 \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.

```

2447 \ExplSyntaxOn
2448 \cs_gset_protected:Npn
2449 \markdownRendererOlBegin
2450 {
2451   \markdownRendererOlBeginPrototype
2452 }
2453 \seq_gput_right:Nn
2454 \g_@@_renderers_seq
2455 { olBegin }
2456 \prop_gput:Nnn
2457 \g_@@_renderer_arities_prop
2458 { olBegin }
2459 { 0 }
2460 \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.

```

2461 \ExplSyntaxOn
2462 \cs_gset_protected:Npn
2463 \markdownRendererOlBeginTight
2464 {
2465   \markdownRendererOlBeginTightPrototype
2466 }
2467 \seq_gput_right:Nn
2468 \g_@@_renderers_seq
2469 { olBeginTight }
2470 \prop_gput:Nnn
2471 \g_@@_renderer_arities_prop
2472 { olBeginTight }
2473 { 0 }
2474 \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`).

```

2475 \ExplSyntaxOn
2476 \cs_gset_protected:Npn
2477   \markdownRendererFancyOlBegin
2478   {
2479     \markdownRendererFancyOlBeginPrototype
2480   }
2481 \seq_gput_right:Nn
2482   \g_@@_renderers_seq
2483   { fancyOlBegin }
2484 \prop_gput:Nnn
2485   \g_@@_renderer_arities_prop
2486   { fancyOlBegin }
2487   { 2 }
2488 \ExplSyntaxOff

```

The `\markdownRendererFancyOlBeginTight` macro represents the beginning of a fancy ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `fancyLists` and `tightLists` options are enabled. The macro receives two arguments: the style of the list item labels, and the style of delimiters between list item labels and texts. See the `\markdownRendererFancyOlBegin` macro for the valid style values.

```

2489 \ExplSyntaxOn
2490 \cs_gset_protected:Npn
2491   \markdownRendererFancyOlBeginTight
2492   {
2493     \markdownRendererFancyOlBeginTightPrototype
2494   }
2495 \seq_gput_right:Nn
2496   \g_@@_renderers_seq
2497   { fancyOlBeginTight }
2498 \prop_gput:Nnn
2499   \g_@@_renderer_arities_prop
2500   { fancyOlBeginTight }
2501   { 2 }
2502 \ExplSyntaxOff

```

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

```

2503 \ExplSyntaxOn
2504 \cs_gset_protected:Npn

```

```

2505 \markdownRendererOlItem
2506 {
2507   \markdownRendererOlItemPrototype
2508 }
2509 \seq_gput_right:Nn
2510 \g_@@_renderers_seq
2511 { olItem }
2512 \prop_gput:Nnn
2513 \g_@@_renderer_arities_prop
2514 { olItem }
2515 { 0 }
2516 \ExplSyntaxOff

```

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

```

2517 \ExplSyntaxOn
2518 \cs_gset_protected:Npn
2519   \markdownRendererOlItemEnd
2520   {
2521     \markdownRendererOlItemEndPrototype
2522   }
2523 \seq_gput_right:Nn
2524 \g_@@_renderers_seq
2525 { olItemEnd }
2526 \prop_gput:Nnn
2527 \g_@@_renderer_arities_prop
2528 { olItemEnd }
2529 { 0 }
2530 \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.

```

2531 \ExplSyntaxOn
2532 \cs_gset_protected:Npn
2533   \markdownRendererOlItemWithNumber
2534   {
2535     \markdownRendererOlItemWithNumberPrototype
2536   }
2537 \seq_gput_right:Nn
2538 \g_@@_renderers_seq
2539 { olItemWithNumber }
2540 \prop_gput:Nnn
2541 \g_@@_renderer_arities_prop

```

```

2542 { olItemWithNumber }
2543 { 1 }
2544 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItem` 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.

```

2545 \ExplSyntaxOn
2546 \cs_gset_protected:Npn
2547 \markdownRendererFancyOliItem
2548 {
2549   \markdownRendererFancyOliItemPrototype
2550 }
2551 \seq_gput_right:Nn
2552 \g_@@_renderers_seq
2553 { fancyOliItem }
2554 \prop_gput:Nnn
2555 \g_@@_renderer_arities_prop
2556 { fancyOliItem }
2557 { 0 }
2558 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItemEnd` 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.

```

2559 \ExplSyntaxOn
2560 \cs_gset_protected:Npn
2561 \markdownRendererFancyOliItemEnd
2562 {
2563   \markdownRendererFancyOliItemEndPrototype
2564 }
2565 \seq_gput_right:Nn
2566 \g_@@_renderers_seq
2567 { fancyOliItemEnd }
2568 \prop_gput:Nnn
2569 \g_@@_renderer_arities_prop
2570 { fancyOliItemEnd }
2571 { 0 }
2572 \ExplSyntaxOff

```

The `\markdownRendererFancyOliItemWithNumber` 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.

```

2573 \ExplSyntaxOn
2574 \cs_gset_protected:Npn

```



```

2575 \markdownRendererFancyOListItemWithNumber
2576 {
2577   \markdownRendererFancyOListItemWithNumberPrototype
2578 }
2579 \seq_gput_right:Nn
2580 \g_@@_renderers_seq
2581 { fancyOListItemWithNumber }
2582 \prop_gput:Nnn
2583 \g_@@_renderer_arities_prop
2584 { fancyOListItemWithNumber }
2585 { 1 }
2586 \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.

```

2587 \ExplSyntaxOn
2588 \cs_gset_protected:Npn
2589   \markdownRendererO1End
2590   {
2591     \markdownRendererO1EndPrototype
2592   }
2593 \seq_gput_right:Nn
2594 \g_@@_renderers_seq
2595 { olEnd }
2596 \prop_gput:Nnn
2597 \g_@@_renderer_arities_prop
2598 { olEnd }
2599 { 0 }
2600 \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.

```

2601 \ExplSyntaxOn
2602 \cs_gset_protected:Npn
2603   \markdownRendererO1EndTight
2604   {
2605     \markdownRendererO1EndTightPrototype
2606   }
2607 \seq_gput_right:Nn
2608 \g_@@_renderers_seq
2609 { olEndTight }
2610 \prop_gput:Nnn

```

```

2611 \g_@@_renderer_arities_prop
2612 { olEndTight }
2613 { 0 }
2614 \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.

```

2615 \ExplSyntaxOn
2616 \cs_gset_protected:Npn
2617 \markdownRendererFancyOlEnd
2618 {
2619 \markdownRendererFancyOlEndPrototype
2620 }
2621 \seq_gput_right:Nn
2622 \g_@@_renderers_seq
2623 { fancyOlEnd }
2624 \prop_gput:Nnn
2625 \g_@@_renderer_arities_prop
2626 { fancyOlEnd }
2627 { 0 }
2628 \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.

```

2629 \ExplSyntaxOn
2630 \cs_gset_protected:Npn
2631 \markdownRendererFancyOlEndTight
2632 {
2633 \markdownRendererFancyOlEndTightPrototype
2634 }
2635 \seq_gput_right:Nn
2636 \g_@@_renderers_seq
2637 { fancyOlEndTight }
2638 \prop_gput:Nnn
2639 \g_@@_renderer_arities_prop
2640 { fancyOlEndTight }
2641 { 0 }
2642 \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.

```
2643 \ExplSyntaxOn
2644 \cs_gset_protected:Npn
2645   \markdownRendererInputRawInline
2646   {
2647     \markdownRendererInputRawInlinePrototype
2648   }
2649 \seq_gput_right:Nn
2650   \g_@@_renderers_seq
2651   { inputRawInline }
2652 \prop_gput:Nnn
2653   \g_@@_renderer_arities_prop
2654   { inputRawInline }
2655   { 2 }
2656 \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.

```
2657 \ExplSyntaxOn
2658 \cs_gset_protected:Npn
2659   \markdownRendererInputRawBlock
2660   {
2661     \markdownRendererInputRawBlockPrototype
2662   }
2663 \seq_gput_right:Nn
2664   \g_@@_renderers_seq
2665   { inputRawBlock }
2666 \prop_gput:Nnn
2667   \g_@@_renderer_arities_prop
2668   { inputRawBlock }
2669   { 2 }
2670 \ExplSyntaxOff
```

### 2.2.5.32 Section Renderers

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

```
2671 \ExplSyntaxOn
2672 \cs_gset_protected:Npn
2673   \markdownRendererSectionBegin
2674   {
2675     \markdownRendererSectionBeginPrototype
2676   }
```

```

2677 \seq_gput_right:Nn
2678   \g_@@_renderers_seq
2679   { sectionBegin }
2680 \prop_gput:Nnn
2681   \g_@@_renderer_arities_prop
2682   { sectionBegin }
2683   { 0 }
2684 \cs_gset_protected:Npn
2685   \markdownRendererSectionEnd
2686   {
2687     \markdownRendererSectionEndPrototype
2688   }
2689 \seq_gput_right:Nn
2690   \g_@@_renderers_seq
2691   { sectionEnd }
2692 \prop_gput:Nnn
2693   \g_@@_renderer_arities_prop
2694   { sectionEnd }
2695   { 0 }
2696 \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.

```

2697 \ExplSyntaxOn
2698 \cs_gset_protected:Npn
2699   \markdownRendererReplacementCharacter
2700   {
2701     \markdownRendererReplacementCharacterPrototype
2702   }
2703 \seq_gput_right:Nn
2704   \g_@@_renderers_seq
2705   { replacementCharacter }
2706 \prop_gput:Nnn
2707   \g_@@_renderer_arities_prop
2708   { replacementCharacter }
2709   { 0 }
2710 \ExplSyntaxOff

```

### 2.2.5.34 Special Character Renderers

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

```

2711 \ExplSyntaxOn
2712 \cs_gset_protected:Npn

```

```

2713 \markdownRendererLeftBrace
2714 {
2715     \markdownRendererLeftBracePrototype
2716 }
2717 \seq_gput_right:Nn
2718 \g_@@_renderers_seq
2719 { leftBrace }
2720 \prop_gput:Nnn
2721 \g_@@_renderer_arities_prop
2722 { leftBrace }
2723 { 0 }
2724 \cs_gset_protected:Npn
2725 \markdownRendererRightBrace
2726 {
2727     \markdownRendererRightBracePrototype
2728 }
2729 \seq_gput_right:Nn
2730 \g_@@_renderers_seq
2731 { rightBrace }
2732 \prop_gput:Nnn
2733 \g_@@_renderer_arities_prop
2734 { rightBrace }
2735 { 0 }
2736 \cs_gset_protected:Npn
2737 \markdownRendererDollarSign
2738 {
2739     \markdownRendererDollarSignPrototype
2740 }
2741 \seq_gput_right:Nn
2742 \g_@@_renderers_seq
2743 { dollarSign }
2744 \prop_gput:Nnn
2745 \g_@@_renderer_arities_prop
2746 { dollarSign }
2747 { 0 }
2748 \cs_gset_protected:Npn
2749 \markdownRendererPercentSign
2750 {
2751     \markdownRendererPercentSignPrototype
2752 }
2753 \seq_gput_right:Nn
2754 \g_@@_renderers_seq
2755 { percentSign }
2756 \prop_gput:Nnn
2757 \g_@@_renderer_arities_prop
2758 { percentSign }
2759 { 0 }

```

```

2760 \cs_gset_protected:Npn
2761   \markdownRendererAmpersand
2762   {
2763     \markdownRendererAmpersandPrototype
2764   }
2765 \seq_gput_right:Nn
2766   \g_@@_renderers_seq
2767   { ampersand }
2768 \prop_gput:Nnn
2769   \g_@@_renderer_arities_prop
2770   { ampersand }
2771   { 0 }
2772 \cs_gset_protected:Npn
2773   \markdownRendererUnderscore
2774   {
2775     \markdownRendererUnderscorePrototype
2776   }
2777 \seq_gput_right:Nn
2778   \g_@@_renderers_seq
2779   { underscore }
2780 \prop_gput:Nnn
2781   \g_@@_renderer_arities_prop
2782   { underscore }
2783   { 0 }
2784 \cs_gset_protected:Npn
2785   \markdownRendererHash
2786   {
2787     \markdownRendererHashPrototype
2788   }
2789 \seq_gput_right:Nn
2790   \g_@@_renderers_seq
2791   { hash }
2792 \prop_gput:Nnn
2793   \g_@@_renderer_arities_prop
2794   { hash }
2795   { 0 }
2796 \cs_gset_protected:Npn
2797   \markdownRendererCircumflex
2798   {
2799     \markdownRendererCircumflexPrototype
2800   }
2801 \seq_gput_right:Nn
2802   \g_@@_renderers_seq
2803   { circumflex }
2804 \prop_gput:Nnn
2805   \g_@@_renderer_arities_prop
2806   { circumflex }

```

```

2807 { 0 }
2808 \cs_gset_protected:Npn
2809 \markdownRendererBackslash
2810 {
2811   \markdownRendererBackslashPrototype
2812 }
2813 \seq_gput_right:Nn
2814 \g_@@_renderers_seq
2815 { backslash }
2816 \prop_gput:Nnn
2817 \g_@@_renderer_arities_prop
2818 { backslash }
2819 { 0 }
2820 \cs_gset_protected:Npn
2821 \markdownRendererTilde
2822 {
2823   \markdownRendererTildePrototype
2824 }
2825 \seq_gput_right:Nn
2826 \g_@@_renderers_seq
2827 { tilde }
2828 \prop_gput:Nnn
2829 \g_@@_renderer_arities_prop
2830 { tilde }
2831 { 0 }
2832 \cs_gset_protected:Npn
2833 \markdownRendererPipe
2834 {
2835   \markdownRendererPipePrototype
2836 }
2837 \seq_gput_right:Nn
2838 \g_@@_renderers_seq
2839 { pipe }
2840 \prop_gput:Nnn
2841 \g_@@_renderer_arities_prop
2842 { pipe }
2843 { 0 }
2844 \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.

```

2845 \ExplSyntaxOn
2846 \cs_gset_protected:Npn

```

```

2847 \markdownRendererStrikeThrough
2848 {
2849   \markdownRendererStrikeThroughPrototype
2850 }
2851 \seq_gput_right:Nn
2852 \g_@@_renderers_seq
2853 { strikeThrough }
2854 \prop_gput:Nnn
2855 \g_@@_renderer_arities_prop
2856 { strikeThrough }
2857 { 1 }
2858 \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.

```

2859 \ExplSyntaxOn
2860 \cs_gset_protected:Npn
2861   \markdownRendererSubscript
2862   {
2863     \markdownRendererSubscriptPrototype
2864   }
2865 \seq_gput_right:Nn
2866 \g_@@_renderers_seq
2867 { subscript }
2868 \prop_gput:Nnn
2869 \g_@@_renderer_arities_prop
2870 { subscript }
2871 { 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.

```

2872 \cs_gset_protected:Npn
2873   \markdownRendererSuperscript
2874   {
2875     \markdownRendererSuperscriptPrototype
2876   }
2877 \seq_gput_right:Nn
2878 \g_@@_renderers_seq
2879 { superscript }
2880 \prop_gput:Nnn
2881 \g_@@_renderer_arities_prop

```



```

2882 { superscript }
2883 { 1 }
2884 \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.

```

2885 \ExplSyntaxOn
2886 \cs_gset_protected:Npn
2887   \markdownRendererTableAttributeContextBegin
2888   {
2889     \markdownRendererTableAttributeContextBeginPrototype
2890   }
2891 \seq_gput_right:Nn
2892   \g_@@_renderers_seq
2893   { tableAttributeContextBegin }
2894 \prop_gput:Nnn
2895   \g_@@_renderer_arities_prop
2896   { tableAttributeContextBegin }
2897   { 0 }
2898 \cs_gset_protected:Npn
2899   \markdownRendererTableAttributeContextEnd
2900   {
2901     \markdownRendererTableAttributeContextEndPrototype
2902   }
2903 \seq_gput_right:Nn
2904   \g_@@_renderers_seq
2905   { tableAttributeContextEnd }
2906 \prop_gput:Nnn
2907   \g_@@_renderer_arities_prop
2908   { tableAttributeContextEnd }
2909   { 0 }
2910 \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.

```

2911 \ExplSyntaxOn
2912 \cs_gset_protected:Npn
2913   \markdownRendererTable
2914   {
2915     \markdownRendererTablePrototype
2916   }
2917 \seq_gput_right:Nn
2918   \g_@@_renderers_seq
2919   { table }
2920 \prop_gput:Nnn
2921   \g_@@_renderer_arities_prop
2922   { table }
2923   { 3 }
2924 \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.

```

2925 \ExplSyntaxOn
2926 \cs_gset_protected:Npn
2927   \markdownRendererInlineMath
2928   {
2929     \markdownRendererInlineMathPrototype
2930   }
2931 \seq_gput_right:Nn
2932   \g_@@_renderers_seq
2933   { inlineMath }
2934 \prop_gput:Nnn
2935   \g_@@_renderer_arities_prop
2936   { inlineMath }
2937   { 1 }
2938 \cs_gset_protected:Npn
2939   \markdownRendererDisplayMath
2940   {
2941     \markdownRendererDisplayMathPrototype
2942   }
2943 \seq_gput_right:Nn

```

```

2944 \g_@@_renderers_seq
2945 { displayMath }
2946 \prop_gput:Nnn
2947 \g_@@_renderer_arities_prop
2948 { displayMath }
2949 { 1 }
2950 \ExplSyntaxOff

```

### 2.2.5.41 Thematic Break Renderer

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

```

2951 \ExplSyntaxOn
2952 \cs_gset_protected:Npn
2953 \markdownRendererThematicBreak
2954 {
2955   \markdownRendererThematicBreakPrototype
2956 }
2957 \seq_gput_right:Nn
2958 \g_@@_renderers_seq
2959 { thematicBreak }
2960 \prop_gput:Nnn
2961 \g_@@_renderer_arities_prop
2962 { thematicBreak }
2963 { 0 }
2964 \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.

```

2965 \ExplSyntaxOn
2966 \cs_gset_protected:Npn
2967 \markdownRendererTickedBox
2968 {
2969   \markdownRendererTickedBoxPrototype
2970 }
2971 \seq_gput_right:Nn
2972 \g_@@_renderers_seq
2973 { tickedBox }
2974 \prop_gput:Nnn
2975 \g_@@_renderer_arities_prop
2976 { tickedBox }

```

```

2977 { 0 }
2978 \cs_gset_protected:Npn
2979 \markdownRendererHalfTickedBox
2980 {
2981   \markdownRendererHalfTickedBoxPrototype
2982 }
2983 \seq_gput_right:Nn
2984 \g_@@_renderers_seq
2985 { halfTickedBox }
2986 \prop_gput:Nnn
2987 \g_@@_renderer_arities_prop
2988 { halfTickedBox }
2989 { 0 }
2990 \cs_gset_protected:Npn
2991 \markdownRendererUntickedBox
2992 {
2993   \markdownRendererUntickedBoxPrototype
2994 }
2995 \seq_gput_right:Nn
2996 \g_@@_renderers_seq
2997 { untickedBox }
2998 \prop_gput:Nnn
2999 \g_@@_renderer_arities_prop
3000 { untickedBox }
3001 { 0 }
3002 \ExplSyntaxOff

```

#### 2.2.5.43 Warning and Error Renderers

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

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

```

3003 \ExplSyntaxOn
3004 \cs_gset_protected:Npn
3005 \markdownRendererWarning
3006 {
3007   \markdownRendererWarningPrototype

```

```

3008 }
3009 \cs_gset_protected:Npn
3010 \markdownRendererError
3011 {
3012   \markdownRendererErrorPrototype
3013 }
3014 \seq_gput_right:Nn
3015 \g_@@_renderers_seq
3016 { warning }
3017 \prop_gput:Nnn
3018 \g_@@_renderer_arities_prop
3019 { warning }
3020 { 4 }
3021 \seq_gput_right:Nn
3022 \g_@@_renderers_seq
3023 { error }
3024 \prop_gput:Nnn
3025 \g_@@_renderer_arities_prop
3026 { error }
3027 { 4 }
3028 \ExplSyntaxOff

```

#### 2.2.5.44 YAML Metadata Renderers

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

```

3029 \ExplSyntaxOn
3030 \cs_gset_protected:Npn
3031 \markdownRendererJekyllDataBegin
3032 {
3033   \markdownRendererJekyllDataBeginPrototype
3034 }
3035 \seq_gput_right:Nn
3036 \g_@@_renderers_seq
3037 { jekyllDataBegin }
3038 \prop_gput:Nnn
3039 \g_@@_renderer_arities_prop
3040 { jekyllDataBegin }
3041 { 0 }
3042 \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.

```

3043 \ExplSyntaxOn
3044 \cs_gset_protected:Npn

```

```

3045 \markdownRendererJekyllDataEnd
3046 {
3047   \markdownRendererJekyllDataEndPrototype
3048 }
3049 \seq_gput_right:Nn
3050 \g_@@_renderers_seq
3051 { jekyllDataEnd }
3052 \prop_gput:Nnn
3053 \g_@@_renderer_arities_prop
3054 { jekyllDataEnd }
3055 { 0 }
3056 \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.

```

3057 \ExplSyntaxOn
3058 \cs_gset_protected:Npn
3059 \markdownRendererJekyllDataMappingBegin
3060 {
3061   \markdownRendererJekyllDataMappingBeginPrototype
3062 }
3063 \seq_gput_right:Nn
3064 \g_@@_renderers_seq
3065 { jekyllDataMappingBegin }
3066 \prop_gput:Nnn
3067 \g_@@_renderer_arities_prop
3068 { jekyllDataMappingBegin }
3069 { 2 }
3070 \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.

```

3071 \ExplSyntaxOn
3072 \cs_gset_protected:Npn
3073 \markdownRendererJekyllDataMappingEnd
3074 {
3075   \markdownRendererJekyllDataMappingEndPrototype
3076 }
3077 \seq_gput_right:Nn
3078 \g_@@_renderers_seq
3079 { jekyllDataMappingEnd }
3080 \prop_gput:Nnn

```

```

3081 \g_@@_renderer_arities_prop
3082 { jekyllDataMappingEnd }
3083 { 0 }
3084 \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.

```

3085 \ExplSyntaxOn
3086 \cs_gset_protected:Npn
3087 \markdownRendererJekyllDataSequenceBegin
3088 {
3089   \markdownRendererJekyllDataSequenceBeginPrototype
3090 }
3091 \seq_gput_right:Nn
3092 \g_@@_renderers_seq
3093 { jekyllDataSequenceBegin }
3094 \prop_gput:Nnn
3095 \g_@@_renderer_arities_prop
3096 { jekyllDataSequenceBegin }
3097 { 2 }
3098 \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.

```

3099 \ExplSyntaxOn
3100 \cs_gset_protected:Npn
3101 \markdownRendererJekyllDataSequenceEnd
3102 {
3103   \markdownRendererJekyllDataSequenceEndPrototype
3104 }
3105 \seq_gput_right:Nn
3106 \g_@@_renderers_seq
3107 { jekyllDataSequenceEnd }
3108 \prop_gput:Nnn
3109 \g_@@_renderer_arities_prop
3110 { jekyllDataSequenceEnd }
3111 { 0 }
3112 \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.

```

3113 \ExplSyntaxOn
3114 \cs_gset_protected:Npn
3115   \markdownRendererJekyllDataBoolean
3116   {
3117     \markdownRendererJekyllDataBooleanPrototype
3118   }
3119 \seq_gput_right:Nn
3120   \g_@@_renderers_seq
3121   { jekyllDataBoolean }
3122 \prop_gput:Nnn
3123   \g_@@_renderer_arities_prop
3124   { jekyllDataBoolean }
3125   { 2 }
3126 \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.

```

3127 \ExplSyntaxOn
3128 \cs_gset_protected:Npn
3129   \markdownRendererJekyllDataNumber
3130   {
3131     \markdownRendererJekyllDataNumberPrototype
3132   }
3133 \seq_gput_right:Nn
3134   \g_@@_renderers_seq
3135   { jekyllDataNumber }
3136 \prop_gput:Nnn
3137   \g_@@_renderer_arities_prop
3138   { jekyllDataNumber }
3139   { 2 }
3140 \ExplSyntaxOff

```

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

For each string scalar value, both macros are produced. Whereas `\markdownRendererJekyllDataTypographicString` receives the scalar value after all markdown markup and special `TeX` characters in the string have been replaced by `TeX` macros, `\markdownRendererJekyllDataProgrammaticString`



receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\text{\TeX}$ , such as document titles, author names, or exam questions, the `\markdownRendererJekyllDataProgrammaticString` macro is more appropriate for identifiers and other programmatic text that won't be typeset by  $\text{\TeX}$ .

```

3141 \ExplSyntaxOn
3142 \cs_gset_protected:Npn
3143   \markdownRendererJekyllDataTypographicString
3144   {
3145     \markdownRendererJekyllDataTypographicStringPrototype
3146   }
3147 \cs_gset_protected:Npn
3148   \markdownRendererJekyllDataProgrammaticString
3149   {
3150     \markdownRendererJekyllDataProgrammaticStringPrototype
3151   }
3152 \seq_gput_right:Nn
3153   \g_@@_renderers_seq
3154   { jekyllDataTypographicString }
3155 \prop_gput:Nnn
3156   \g_@@_renderer_arities_prop
3157   { jekyllDataTypographicString }
3158   { 2 }
3159 \seq_gput_right:Nn
3160   \g_@@_renderers_seq
3161   { jekyllDataProgrammaticString }
3162 \prop_gput:Nnn
3163   \g_@@_renderer_arities_prop
3164   { jekyllDataProgrammaticString }
3165   { 2 }
3166 \ExplSyntaxOff

```

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

```

3167 \ExplSyntaxOn
3168 \cs_gset:Npn
3169   \markdownRendererJekyllDataTypographicString
3170   {
3171     \cs_if_exist:NTF
3172       \markdownRendererJekyllDataString
3173       {
3174         \@@_if_option:nTF
3175           { experimental }
3176           {

```

```

3177         \markdownError
3178         {
3179             The~jekyllDataString~renderer~has~been~deprecated,~
3180             to~be~removed~in~Markdown~4.0.0
3181         }
3182     }
3183     {
3184         \markdownWarning
3185         {
3186             The~jekyllDataString~renderer~has~been~deprecated,~
3187             to~be~removed~in~Markdown~4.0.0
3188         }
3189         \markdownRendererJekyllDataString
3190     }
3191 }
3192 {
3193     \cs_if_exist:NTF
3194     \markdownRendererJekyllDataStringPrototype
3195     {
3196         \@@_if_option:nTF
3197         { experimental }
3198         {
3199             \markdownError
3200             {
3201                 The~jekyllDataString~renderer~prototype~
3202                 has~been~deprecated,~
3203                 to~be~removed~in~Markdown~4.0.0
3204             }
3205         }
3206         {
3207             \markdownWarning
3208             {
3209                 The~jekyllDataString~renderer~prototype~
3210                 has~been~deprecated,~
3211                 to~be~removed~in~Markdown~4.0.0
3212             }
3213             \markdownRendererJekyllDataStringPrototype
3214         }
3215     }
3216     {
3217         \markdownRendererJekyllDataTypographicStringPrototype
3218     }
3219 }
3220 }
3221 \seq_gput_right:Nn
3222 \g_@@_renderers_seq
3223 { jekyllDataString }

```

```

3224 \prop_gput:Nnn
3225   \g_@@_renderer_arities_prop
3226   { jekyllDataString }
3227   { 2 }
3228 \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.

```

3229 \ExplSyntaxOn
3230 \cs_gset_protected:Npn
3231   \markdownRendererJekyllDataEmpty
3232   {
3233     \markdownRendererJekyllDataEmptyPrototype
3234   }
3235 \seq_gput_right:Nn
3236   \g_@@_renderers_seq
3237   { jekyllDataEmpty }
3238 \prop_gput:Nnn
3239   \g_@@_renderer_arities_prop
3240   { jekyllDataEmpty }
3241   { 1 }
3242 \ExplSyntaxOff

```

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

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

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

```

3243 \ExplSyntaxOn
3244 \cs_new:Nn \@@_define_renderers:
3245   {
3246     \seq_map_inline:Nn
3247       \g_@@_renderers_seq
3248       {
3249         \@@_define_renderer:n
3250         { ##1 }
3251       }

```

```

3252 }
3253 \cs_new:Nn \@@_define_renderer:n
3254 {
3255   \@@_renderer_tl_to_csname:nN
3256   { #1 }
3257   \l_tmpa_tl
3258   \prop_get:NnN
3259   \g_@@_renderer_arities_prop
3260   { #1 }
3261   \l_tmpb_tl
3262   \@@_define_renderer:ncV
3263   { #1 }
3264   { \l_tmpa_tl }
3265   \l_tmpb_tl
3266 }
3267 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3268 {
3269   \tl_set:Nn
3270   \l_tmpa_tl
3271   { \str_uppercase:n { #1 } }
3272   \tl_set:Nx
3273   #2
3274   {
3275     markdownRenderer
3276     \tl_head:f { \l_tmpa_tl }
3277     \tl_tail:n { #1 }
3278   }
3279 }
3280 \tl_new:N
3281 \l_@@_renderer_definition_tl
3282 \bool_new:N
3283 \g_@@_appending_renderer_bool
3284 \bool_new:N
3285 \g_@@_unprotected_renderer_bool
3286 \cs_new:Nn \@@_define_renderer:nNn
3287 {
3288   \keys_define:nn
3289   { markdown/options/renderers }
3290   {
3291     #1 .code:n = {
3292       \tl_set:Nn
3293       \l_@@_renderer_definition_tl
3294       { ##1 }
3295       \regex_replace_all:nnN
3296       { \cP\#0 }
3297       { #1 }
3298       \l_@@_renderer_definition_tl

```

```

3299     \bool_if:NT
3300         \g_@@_appending_renderer_bool
3301     {
3302         \@@_tl_set_from_cs:NNn
3303         \l_tmpa_tl
3304         #2
3305         { #3 }
3306         \tl_put_left:NV
3307         \l_@@_renderer_definition_tl
3308         \l_tmpa_tl
3309     }
3310     \bool_if:NTF
3311         \g_@@_unprotected_renderer_bool
3312     {
3313         \tl_set:Nn
3314         \l_tmpa_tl
3315         { \cs_set:Npn }
3316     }
3317     {
3318         \tl_set:Nn
3319         \l_tmpa_tl
3320         { \cs_set_protected:Npn }
3321     }
3322     \exp_last_unbraced:NNV
3323     \cs_generate_from_arg_count:NNnV
3324     #2
3325     \l_tmpa_tl
3326     { #3 }
3327     \l_@@_renderer_definition_tl
3328 },
3329 }

```

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

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

```

3330     \str_if_eq:nnT
3331     { #1 }
3332     { jekyllDataString }
3333     {
3334         \cs_undefine:N
3335         #2
3336     }
3337 }

```

We define the function `\@@_tl_set_from_cs:NNn` [11]. 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.

```

3338 \cs_new_protected:Nn
3339   \@@_tl_set_from_cs:NNn
3340   {
3341     \tl_set:Nn
3342       \l_tmpa_tl
3343       { #2 }
3344     \int_step_inline:nn
3345       { #3 }
3346     {
3347       \exp_args:Nnc
3348         \tl_put_right:Nn
3349         \l_tmpa_tl
3350         { @@_tl_set_from_cs_parameter_ ##1 }
3351     }
3352     \exp_args:NNV
3353     \tl_set:No
3354     \l_tmpb_tl
3355     \l_tmpa_tl
3356     \regex_replace_all:nnN
3357     { \cP. }
3358     { \0\0 }
3359     \l_tmpb_tl
3360     \int_step_inline:nn
3361     { #3 }
3362     {
3363       \regex_replace_all:nnN
3364       { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3365       { \cP\# ##1 }
3366       \l_tmpb_tl
3367     }
3368     \tl_set:NV
3369     #1
3370     \l_tmpb_tl
3371   }
3372 \cs_generate_variant:Nn
3373   \@@_define_renderer:nNn
3374   { ncV }
3375 \cs_generate_variant:Nn
3376   \cs_generate_from_arg_count:NNnn
3377   { NNnV }
3378 \cs_generate_variant:Nn
3379   \tl_put_left:Nn
3380   { Nv }
3381 \keys_define:nn
3382   { markdown/options }

```

```

3383 {
3384   renderers .code:n = {
3385     \bool_gset_false:N
3386     \g_@@_unprotected_renderer_bool
3387     \keys_set:nn
3388     { markdown/options/renderers }
3389     { #1 }
3390   },
3391   unprotectedRenderers .code:n = {
3392     \bool_gset_true:N
3393     \g_@@_unprotected_renderer_bool
3394     \keys_set:nn
3395     { markdown/options/renderers }
3396     { #1 }
3397   },
3398 }

```

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.
  }
}

```

```

3399 \tl_new:N
3400 \l_@@_renderer_glob_definition_tl
3401 \seq_new:N
3402 \l_@@_renderer_glob_results_seq
3403 \regex_const:Nn
3404 \c_@@_appending_key_regex
3405 { \s*+$ }
3406 \keys_define:nn
3407 { markdown/options/renderers }
3408 {
3409   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 += {...},
        },
    }
},
}

```

```

3410     \regex_match:NVTF
3411     \c_@@_appending_key_regex
3412     \l_keys_key_str
3413     {
3414         \bool_gset_true:N
3415         \g_@@_appending_renderer_bool
3416         \tl_set:NV
3417         \l_tmpa_tl
3418         \l_keys_key_str
3419         \regex_replace_once:NnN
3420         \c_@@_appending_key_regex
3421         { }
3422         \l_tmpa_tl
3423         \tl_set:Nx
3424         \l_tmpb_tl
3425         { { \l_tmpa_tl } = }
3426         \tl_put_right:Nn
3427         \l_tmpb_tl
3428         { { #1 } }
3429         \keys_set:nV
3430         { markdown/options/renderers }
3431         \l_tmpb_tl

```



```

3432         \bool_gset_false:N
3433         \g_@@_appending_renderer_bool
3434     }

```

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.
  }
}

```

```

3435     {
3436         \@@_glob_seq:VnN
3437         \l_keys_key_str
3438         { g_@@_renderers_seq }
3439         \l_@@_renderer_glob_results_seq
3440     \seq_if_empty:NTF
3441         \l_@@_renderer_glob_results_seq
3442     {
3443         \msg_error:nnV
3444         { markdown }
3445         { undefined-renderer }

```

```

3446         \l_keys_key_str
3447     }
3448     {
3449         \tl_set:Nn
3450         \l_@@_renderer_glob_definition_tl
3451         { \exp_not:n { #1 } }
3452         \seq_map_inline:Nn
3453         \l_@@_renderer_glob_results_seq
3454         {
3455             \tl_set:Nn
3456             \l_tmpa_tl
3457             { { ##1 } = }
3458             \tl_put_right:Nx
3459             \l_tmpa_tl
3460             { { \l_@@_renderer_glob_definition_tl } }
3461             \keys_set:nV
3462             { markdown/options/renderers }
3463             \l_tmpa_tl
3464         }
3465     }
3466 }
3467 },
3468 }
3469 \msg_new:nnn
3470 { markdown }
3471 { undefined-renderer }
3472 {
3473     Renderer~#1~is~undefined.
3474 }
3475 \cs_generate_variant:Nn
3476 \@@_glob_seq:nnN
3477 { VnN }
3478 \cs_generate_variant:Nn
3479 \cs_generate_from_arg_count:NNnn
3480 { cNVV }
3481 \cs_generate_variant:Nn
3482 \msg_error:nnn
3483 { nnV }
3484 \prg_generate_conditional_variant:Nnn
3485 \regex_match:Nn
3486 { NV }
3487 { TF }
3488 \prop_new:N
3489 \g_@@_glob_cache_prop
3490 \tl_new:N
3491 \l_@@_current_glob_tl
3492 \cs_new:Nn

```

```

3493 \@@_glob_seq:nnN
3494 {
3495   \tl_set:Nn
3496     \l_@@_current_glob_tl
3497     { ^ #1 $ }
3498   \prop_get:NeNTF
3499     \g_@@_glob_cache_prop
3500     { #2 / \l_@@_current_glob_tl }
3501   \l_tmpa_clist
3502   {
3503     \seq_set_from_clist:NN
3504       #3
3505       \l_tmpa_clist
3506   }
3507   {
3508     \seq_clear:N
3509       #3
3510     \regex_replace_all:nnN
3511       { \* }
3512       { .* }
3513       \l_@@_current_glob_tl
3514     \regex_set:NV
3515       \l_tmpa_regex
3516       \l_@@_current_glob_tl
3517     \seq_map_inline:cn
3518       { #2 }
3519     {
3520       \regex_match:NnT
3521         \l_tmpa_regex
3522         { ##1 }
3523       {
3524         \seq_put_right:Nn
3525           #3
3526           { ##1 }
3527       }
3528     }
3529     \clist_set_from_seq:NN
3530       \l_tmpa_clist
3531       #3
3532     \prop_gput:NeV
3533       \g_@@_glob_cache_prop
3534       { #2 / \l_@@_current_glob_tl }
3535     \l_tmpa_clist
3536   }
3537 }
3538 \cs_generate_variant:Nn
3539   \regex_set:Nn

```

```

3540 { NV }
3541 \cs_generate_variant:Nn
3542 \prop_gput:Nnn
3543 { NeV }

```

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

```

3544 \str_if_eq:VVT
3545 \c_@@_top_layer_tl
3546 \c_@@_option_layer_plain_tex_tl
3547 {
3548   \@@_define_renderers:
3549 }
3550 \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  $\LaTeX$ 3 kernel.

```

3551 \ExplSyntaxOn
3552 \keys_define:nn
3553 { markdown/jekyllData }
3554 { }
3555 \ExplSyntaxOff

```

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

```

3556 \ExplSyntaxOn
3557 \@@_with_various_cases:nn
3558 { jekyllDataRenderers }
3559 {
3560   \keys_define:nn
3561     { markdown/options }
3562     {
3563       #1 .code:n = {
3564         \tl_set:Nn
3565           \l_tmpa_tl
3566           { ##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.

```

3567         \tl_replace_all:NnV
3568         \l_tmpa_tl
3569         { / }
3570         \c_backslash_str
3571         \keys_set:nV
3572         { markdown/options/jekyll-data-renderers }
3573         \l_tmpa_tl
3574     },
3575 }
3576 }
3577 \keys_define:nn
3578 { markdown/options/jekyll-data-renderers }
3579 {
3580     unknown .code:n = {
3581         \tl_set_eq:NN
3582         \l_tmpa_tl
3583         \l_keys_key_str
3584         \tl_replace_all:NVn
3585         \l_tmpa_tl
3586         \c_backslash_str
3587         { / }
3588         \tl_put_right:Nn
3589         \l_tmpa_tl
3590         {
3591             .code:n = { #1 }
3592         }
3593         \keys_define:nV
3594         { markdown/jekyllData }
3595         \l_tmpa_tl
3596     }
3597 }
3598 \cs_generate_variant:Nn
3599 \keys_define:nn
3600 { nV }
3601 \ExplSyntaxOff

```

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

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

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

```

3602 \ExplSyntaxOn
3603 \cs_new:Nn \@@_define_renderer_prototypes:
3604 {
3605   \seq_map_inline:Nn
3606     \g_@@_renderers_seq
3607     {
3608       \@@_define_renderer_prototype:n
3609         { ##1 }
3610     }
3611 }
3612 \cs_new:Nn \@@_define_renderer_prototype:n
3613 {
3614   \@@_renderer_prototype_tl_to_csname:nN
3615     { #1 }
3616     \l_tmpa_tl
3617   \prop_get:NnN
3618     \g_@@_renderer_arities_prop
3619     { #1 }
3620     \l_tmpb_tl
3621   \@@_define_renderer_prototype:ncV
3622     { #1 }
3623     { \l_tmpa_tl }
3624     \l_tmpb_tl
3625 }
3626 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3627 {
3628   \tl_set:Nn
3629     \l_tmpa_tl
3630     { \str_uppercase:n { #1 } }
3631   \tl_set:Nx
3632     #2
3633     {
3634       markdownRenderer
3635       \tl_head:f { \l_tmpa_tl }
3636       \tl_tail:n { #1 }
3637       Prototype
3638     }
3639 }
3640 \tl_new:N
3641   \l_@@_renderer_prototype_definition_tl
3642 \bool_new:N
3643   \g_@@_appending_renderer_prototype_bool

```

```

3644 \bool_new:N
3645   \g_@@_unprotected_renderer_prototype_bool
3646 \cs_new:Nn \@@_define_renderer_prototype:nNn
3647   {
3648     \keys_define:nn
3649       { markdown/options/renderer-prototypes }
3650     {
3651       #1 .code:n = {
3652         \tl_set:Nn
3653           \l_@@_renderer_prototype_definition_tl
3654           { ##1 }
3655         \regex_replace_all:nnN
3656           { \cP\#0 }
3657           { #1 }
3658         \l_@@_renderer_prototype_definition_tl
3659         \bool_if:NT
3660           \g_@@_appending_renderer_prototype_bool
3661           {
3662             \@@_tl_set_from_cs:NNn
3663             \l_tmpa_tl
3664             #2
3665             { #3 }
3666             \tl_put_left:NV
3667               \l_@@_renderer_prototype_definition_tl
3668               \l_tmpa_tl
3669           }
3670         \bool_if:NTF
3671           \g_@@_unprotected_renderer_prototype_bool
3672           {
3673             \tl_set:Nn
3674               \l_tmpa_tl
3675               { \cs_set:Npn }
3676           }
3677           {
3678             \tl_set:Nn
3679               \l_tmpa_tl
3680               { \cs_set_protected:Npn }
3681           }
3682         \exp_last_unbraced:NNV
3683           \cs_generate_from_arg_count:NNnV
3684           #2
3685           \l_tmpa_tl
3686           { #3 }
3687           \l_@@_renderer_prototype_definition_tl
3688       },
3689     }

```

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

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

```

3690   \str_if_eq:nnF
3691     { #1 }
3692     { jekyllDataString }
3693     {
3694       \cs_if_free:NT
3695         #2
3696         {
3697           \cs_generate_from_arg_count:NNnn
3698             #2
3699             \cs_gset_protected:Npn
3700               { #3 }
3701               { }
3702           }
3703         }
3704     }
3705 \cs_generate_variant:Nn
3706   \@@_define_renderer_prototype:nNn
3707   { 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.
  }
}

```

```

3708 \keys_define:nn
3709   { markdown/options/renderer-prototypes }
3710   {
3711     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 += {...},
        },
    }
},
}

```

```

3712     \regex_match:NVTF
3713     \c_@@_appending_key_regex
3714     \l_keys_key_str
3715     {
3716         \bool_gset_true:N
3717         \g_@@_appending_renderer_prototype_bool
3718         \tl_set:NV
3719         \l_tmpa_tl
3720         \l_keys_key_str
3721         \regex_replace_once:NnN
3722         \c_@@_appending_key_regex
3723         { }
3724         \l_tmpa_tl
3725         \tl_set:Nx
3726         \l_tmpb_tl
3727         { { \l_tmpa_tl } = }
3728         \tl_put_right:Nn
3729         \l_tmpb_tl
3730         { { #1 } }
3731         \keys_set:nV
3732         { markdown/options/renderer-prototypes }
3733         \l_tmpb_tl

```

```

3734         \bool_gset_false:N
3735         \g_@@_appending_renderer_prototype_bool
3736     }

```

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.
}

```

```

3737     {
3738         \@@_glob_seq:VnN
3739         \l_keys_key_str
3740         { g_@@_renderers_seq }
3741         \l_@@_renderer_glob_results_seq
3742         \seq_if_empty:NTF
3743         \l_@@_renderer_glob_results_seq
3744         {
3745             \msg_error:nnV
3746             { markdown }

```

```

3747         { undefined-renderer-prototype }
3748         \l_keys_key_str
3749     }
3750     {
3751         \tl_set:Nn
3752         \l_@@_renderer_glob_definition_tl
3753         { \exp_not:n { #1 } }
3754         \seq_map_inline:Nn
3755         \l_@@_renderer_glob_results_seq
3756         {
3757             \tl_set:Nn
3758             \l_tmpa_tl
3759             { { ##1 } = }
3760             \tl_put_right:Nx
3761             \l_tmpa_tl
3762             { { \l_@@_renderer_glob_definition_tl } }
3763             \keys_set:nV
3764             { markdown/options/renderer-prototypes }
3765             \l_tmpa_tl
3766         }
3767     }
3768 }
3769 },
3770 }
3771 \msg_new:nnn
3772 { markdown }
3773 { undefined-renderer-prototype }
3774 {
3775     Renderer~prototype~#1~is~undefined.
3776 }
3777 \@@_with_various_cases:nn
3778 { rendererPrototypes }
3779 {
3780     \keys_define:nn
3781     { markdown/options }
3782     {
3783         #1 .code:n = {
3784             \bool_gset_false:N
3785             \g_@@_unprotected_renderer_prototype_bool
3786             \keys_set:nn
3787             { markdown/options/renderer-prototypes }
3788             { ##1 }
3789         },
3790     }
3791 }
3792 \@@_with_various_cases:nn
3793 { unprotectedRendererPrototypes }

```

```

3794 {
3795   \keys_define:nn
3796     { markdown/options }
3797     {
3798       #1 .code:n = {
3799         \bool_gset_true:N
3800           \g_@@_unprotected_renderer_prototype_bool
3801         \keys_set:nn
3802           { markdown/options/renderer-prototypes }
3803           { ##1 }
3804       },
3805     }
3806 }

```

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.

```

3807 \str_if_eq:VVT
3808   \c_@@_top_layer_tl
3809   \c_@@_option_layer_plain_tex_tl
3810   {
3811     \@@_define_renderer_prototypes:
3812   }
3813 \ExplSyntaxOff

```

### 2.2.7 Logging Facilities

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

The `\markdownInfo`, `\markdownWarning`, and `\markdownError` macros have been deprecated and will be removed in the next major version of the Markdown package.

### 2.2.8 Miscellanea

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

```

3814 \let\markdownMakeOther\relax

```

The `\markdownReadAndConvert` macro implements the `\markdownBegin` and `\yamlBegin` macros. The first argument specifies the token sequence that will

terminate the markdown input when the plain  $\TeX$  special characters have had their category changed to *other*: `\markdownEnd` for the `\markdownBegin` macro and `\yamlEnd` for the `\yamlBegin` macro. The second argument specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
3815 \let\markdownReadAndConvert\relax
3816 \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*.

```
3817 \catcode`\|=0\catcode`\=12%
3818 |gdef|markdownBegin{%
3819   |markdownReadAndConvert{\markdownEnd}%
3820                               {\|markdownEnd}}%
3821 |gdef|yamlBegin{%
3822   |begingroup
3823   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3824   |markdownReadAndConvert{\yamlEnd}%
3825                               {\|yamlEnd}}%
3826 |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.

```
3827 \ExplSyntaxOn
3828 \keys_define:nn
3829   { markdown/options }
3830   {
3831     code .code:n = { #1 },
3832   }
3833 \ExplSyntaxOff
```

This can be especially useful in snippets.

## 2.3 $\LaTeX$ Interface

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

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

```

3834 \ExplSyntaxOn
3835 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3836 \cs_generate_variant:Nn
3837   \tl_const:Nn
3838   { NV }
3839 \tl_if_exist:NF
3840   \c_@@_top_layer_tl
3841   {
3842     \tl_const:NV
3843       \c_@@_top_layer_tl
3844       \c_@@_option_layer_latex_tl
3845   }
3846 \ExplSyntaxOff
3847 \input markdown/markdown

```

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

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

where *<options>* are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.45) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single L<sup>A</sup>T<sub>E</sub>X theme (see Section 2.3.4) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

#### 2.3.1.1 Typesetting Markdown and YAML directly

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

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

```

3848 \newenvironment{markdown}\relax\relax
3849 \newenvironment{markdown*}[1]\relax\relax

```

Furthermore, both environments accept L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.3) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

The `markdown` and `markdown*` environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```

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

```

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

```

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

```

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

```

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

```

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

Furthermore, using the `\markdownEnd` macro in of after the replacement after-text is optional and only makes a difference if you redefined it to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment.

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

```

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

```

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

```

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

```

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

```
3850 \newenvironment{yaml}\relax\relax
```

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

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

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

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

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

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

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



The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain  $\TeX$ . Unlike the `\markinline` macro provided by the plain  $\TeX$  interface, this macro also accepts  $\LaTeX$  interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown content.

### 2.3.1.2 Typesetting Markdown and YAML from external documents

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain  $\TeX$ . Unlike the `\markdownInput` macro provided by the plain  $\TeX$  interface, this macro also accepts  $\LaTeX$  interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown document.

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

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

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain  $\TeX$ . Unlike the `\yamlInput` macro provided by the plain  $\TeX$  interface, this macro also accepts  $\LaTeX$  interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

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

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
```

```

    jekyllData,
    expectJekyllData,
    ensureJekyllData,
    smartEllipses,
] {hello.yml}
\end{document}

```

### 2.3.2 Using $\LaTeX$ hooks with the Markdown package

$\LaTeX$  provides an intricate hook management system that allows users to insert extra material before and after certain  $\TeX$  macros and  $\LaTeX$  environments, among other things. [12, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before  $\TeX$  commands and before/after  $\LaTeX$  environments without restriction:

```

\documentclass{article}
\usepackage{markdown}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{emphasis: }
\AddToHook{env/markdown/before}{<markdown>}
\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with  $\LaTeX$  will produce the text “`markdownfoo emphasis: _bar_ baz!</markdown>`”, as expected.

However, using hooks to insert extra material after  $\TeX$  commands only works for commands with a fixed number of parameters that don’t use currying.

If, in the above example, you explicitly defined the renderer for emphasis using `\markdownSetup` or another method that does not use currying, then you would be able to insert extra material even after the renderer:

```

\documentclass{article}
\usepackage{markdown}
\markdownSetup{renderers={emphasis={\emph{#1}}}}
\begin{document}
\AddToHook{cmd/markdownRendererEmphasis/before}{<emphasis>}
\AddToHook{cmd/markdownRendererEmphasis/after}{</emphasis>}
\AddToHook{env/markdown/before}{<markdown>}

```

```

\AddToHook{env/markdown/after}{</markdown>}
\begin{markdown}
foo _bar_ baz!
\end{markdown}
\end{document}

```

Processing the above example with  $\LaTeX$  will produce the text “`markdownfoo emphasis_bar_/emphasis baz!/markdown`”, as expected.

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

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

### 2.3.3 Options

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

$\LaTeX$  options map directly to the options recognized by the plain  $\TeX$  interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain  $\TeX$  interface (see Sections 2.2.5 and 2.2.6).

The  $\LaTeX$  options may be specified when loading the  $\LaTeX$  package, when using the `markdown*`  $\LaTeX$  environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro.

#### 2.3.3.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [13, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain  $\TeX$  options, the Markdown package also recognizes these as aliases and accepts them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

```

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

```

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

```
3851 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3852 \DeclareOption{frozenscache}{\markdownSetup{frozensCache}}
```

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

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

```
3853 \ExplSyntaxOn
3854 \str_if_eq:VVT
3855   \c_@@_top_layer_tl
3856   \c_@@_option_layer_latex_tl
3857   {
3858     \@@_define_option_commands_and_keyvals:
3859     \@@_define_renderers:
3860     \@@_define_renderer_prototypes:
3861   }
3862 \ExplSyntaxOff
```

The following example L<sup>A</sup>T<sub>E</sub>X code showcases a possible configuration of plain T<sub>E</sub>X interface options `hybrid`, `smartEllipses`, and `cacheDir`.

```
\markdownSetup{
  hybrid,
  smartEllipses,
  cacheDir = /tmp,
}
```

### 2.3.4 Themes

In Section 2.2.3, we described the concept of themes. In L<sup>A</sup>T<sub>E</sub>X, we expand on the concept of themes by allowing a theme to be a full-blown L<sup>A</sup>T<sub>E</sub>X package. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a L<sup>A</sup>T<sub>E</sub>X package named `markdowntheme<munged theme name>.sty` if it exists and a T<sub>E</sub>X document named `markdowntheme<munged theme name>.tex` otherwise.

Having the Markdown package automatically load either the generic `.tex theme file` or the L<sup>A</sup>T<sub>E</sub>X-specific `.sty theme file` allows developers to have a single *theme file*, when the theme is small or the difference between T<sub>E</sub>X formats is unimportant, and

scale up to separate theme files native to different  $\text{T}_{\text{E}}\text{X}$  formats for large multi-format themes, where different code is needed for different  $\text{T}_{\text{E}}\text{X}$  formats. To enable code reuse, developers can load the `.tex` theme file from the `.sty` theme file using the `\markdownLoadPlainTeXTheme` macro.

If the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  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  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, the following code would first load the Markdown package, then the theme `witiko/example/foo`, and finally the theme `witiko/example/bar`:

```
\usepackage[
  import=witiko/example/foo,
  import=witiko/example/bar,
]{markdown}
```

```
3863 \newif\ifmarkdownLaTeXLoaded
3864 \markdownLaTeXLoadedfalse
```

Due to limitations of  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ , themes may not be loaded after the beginning of a  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  document.

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

```
3865 \ExplSyntaxOn
3866 \prop_new:N
3867 \g_@@_latex_built_in_themes_prop
3868 \ExplSyntaxOff
```

Built-in  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  themes provided with the Markdown package include:

**witiko/markdown/defaults** A  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  theme with the default definitions of token renderer prototypes for plain  $\text{T}_{\text{E}}\text{X}$ . This theme is loaded automatically together with the package and explicitly loading it has no effect.

```
3869 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

At the end of the  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  module, we load the `witiko/markdown/defaults`  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  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).

```
3870 \ExplSyntaxOn
3871 \str_if_eq:VVT
3872 \c_@@_top_layer_tl
3873 \c_@@_option_layer_latex_tl
3874 {
```

```

3875 \ExplSyntaxOff
3876 \AtEndOfPackage
3877 {
3878   \@@_if_option:nF
3879   { noDefaults }
3880   {
3881     \@@_if_option:nTF
3882     { experimental }
3883     {
3884       \@@_setup:n
3885       { theme = witiko/markdown/defaults@experimental }
3886     }
3887     {
3888       \@@_setup:n
3889       { theme = witiko/markdown/defaults }
3890     }
3891   }
3892 }
3893 \ExplSyntaxOn
3894 }
3895 \ExplSyntaxOff

```

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

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

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

```

3896 \ExplSyntaxOn
3897 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3898 \cs_generate_variant:Nn
3899   \tl_const:Nn
3900   { NV }
3901 \tl_if_exist:NF
3902   \c_@@_top_layer_tl
3903   {
3904     \tl_const:NV
3905       \c_@@_top_layer_tl
3906       \c_@@_option_layer_context_tl
3907   }
3908 \ExplSyntaxOff

```

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

```

3909 \writestatus{loading}{ConTeXt User Module / markdown}%
3910 \startmodule[markdown]
3911 \def\dospecials{\do\ \do\\\do{\do\}\do\$\do\&%
3912 \do#\do\^\do\_do\%do\~}%
3913 \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 and YAML

The interface exposes the `\startmarkdown`, `\stopmarkdown`, `\startyaml`, `\stopyaml`, `\inputmarkdown`, and `\inputyaml` macros.

### 2.4.1.1 Typesetting Markdown and YAML directly

The `\startmarkdown` and `\stopmarkdown` macros are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain TeX interface.

```

3914 \let\startmarkdown\relax
3915 \let\stopmarkdown\relax

```

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

The macros `\startmarkdown` and `\stopmarkdown` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

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

```

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

```

The `\startyaml` and `\stopyaml` macros are aliases for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain TeX interface.

```

3916 \let\startyaml\relax
3917 \let\stopyaml\relax

```

You may prepend your own code to the `\startyaml` macro and append your own code to the `\stopyaml` macro to produce special effects before and after the YAML document.

The macros `\startyaml` and `\stopyaml` are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros.

The following example ConTeXt code showcases the usage of the `\startyaml` and `\stopyaml` macros:

```
\usemodule[t][markdown]
\starttext
\startyaml
title: _Hello_ world ...
author: John Doe
\stopyaml
\stoptext
```

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

```
\usemodule[t][markdown]
\starttext
\setupyaml[jekyllData, expectJekyllData, ensureJekyllData]
\startyaml
title: _Hello_ world ...
author: John Doe
\stopyaml
\stoptext
```

#### 2.4.1.2 Typesetting Markdown and YAML from external documents

The `\inputmarkdown` macro aliases the macro `\markdownInput` exposed by the plain TeX interface.

```
3918 \let\inputmarkdown\relax
```

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

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

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



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

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

The `\inputyaml` macro aliases the macro `\yamlInput` exposed by the plain `TEX` interface.

```
3919 \let\inputyaml\relax
```

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

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

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

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

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

## 2.4.2 Options

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

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

The `ConTeXt` options may be specified when using the `\inputmarkdown` macro (see Section 2.4), via the `\markdownSetup` macro, or via the `\setupmarkdown[#1]` macro, which is an alias for `\markdownSetup{#1}`.

```
3920 \ExplSyntaxOn
```

```

3921 \cs_new:Npn
3922   \setupmarkdown
3923   [ #1 ]
3924   {
3925     \@@_setup:n
3926     { #1 }
3927   }

```

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

```

3928 \cs_gset_eq:NN
3929   \setupyaml
3930   \setupmarkdown

```

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

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

```

3931 \cs_new:Nn \@@_caseless:N
3932   {
3933     \regex_replace_all:nnN
3934     { ([a-z])([A-Z]) }
3935     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3936     #1
3937     \tl_set:Nx
3938     #1
3939     { #1 }
3940   }
3941 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3942 \str_if_eq:VVT
3943   \c_@@_top_layer_tl
3944   \c_@@_option_layer_context_tl
3945   {
3946     \@@_define_option_commands_and_keyvals:
3947     \@@_define_renderers:
3948     \@@_define_renderer_prototypes:
3949   }

```

### 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]
```

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

```
3950 \prop_new:N
3951   \g_@@_context_built_in_themes_prop
3952 \ExplSyntaxOff
```

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

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

```
3953 \startmodule[markdownthemewitiko_markdown_defaults]
3954 \unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConTeXt themes.

## 3 Implementation

This part of the documentation describes the implementation of the interfaces exposed by the package (see Section 2) and is aimed at the developers of the package, as well as the curious users.

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TeX *token renderers* is performed by the Lua layer. The plain

TeX layer provides default definitions for the token renderers. The L<sup>A</sup>TeX 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 TeX, 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 TeX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3955 local upper, format, length =
3956   string.upper, string.format, string.len
3957 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3958   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3959   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 TeX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
3960 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.

```
3961 function util.err(msg, exit_code)
3962   io.stderr:write("markdown.lua: " .. msg .. "\n")
3963   os.exit(exit_code or 1)
3964 end
```

The `util.cache` method used `dir`, `string`, `salt`, and `suffix` to determine a pathname. If a file with such a pathname does not exist, it gets created with `transform(string)` as its content. Regardless, the pathname is then returned.

```
3965 function util.cache(dir, string, salt, transform, suffix)
3966   local digest = md5.sumhexa(string .. (salt or ""))
3967   local name = util.pathname(dir, digest .. suffix)
3968   local file = io.open(name, "r")
3969   if file == nil then -- If no cache entry exists, create a new one.
3970     file = assert(io.open(name, "w"),
3971       [[Could not open file ]] .. name .. [[ for writing]])
3972     local result = string
```

```

3973     if transform ~= nil then
3974         result = transform(result)
3975     end
3976     assert(file:write(result))
3977     assert(file:close())
3978 end
3979 return name
3980 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.

```

3981 function util.cache_verbatim(dir, string)
3982     local name = util.cache(dir, string, nil, nil, ".verbatim")
3983     return name
3984 end

```

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

```

3985 function util.table_copy(t)
3986     local u = { }
3987     for k, v in pairs(t) do u[k] = v end
3988     return setmetatable(u, getmetatable(t))
3989 end

```

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

```

3990 function util.encode_json_string(s)
3991     s = s:gsub([[\\]], [[\\]])
3992     s = s:gsub([[\"]], [[\"]])
3993     return [[\"]] .. s .. [[\"]]
3994 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 [14, Chapter 21].

```

3995 function util.expand_tabs_in_line(s, tabstop)
3996     local tab = tabstop or 4
3997     local corr = 0
3998     return (s:gsub(")\t", function(p)
3999         local sp = tab - (p - 1 + corr) % tab
4000         corr = corr - 1 + sp
4001         return string.rep(" ", sp)
4002     end))
4003 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.

```
4004 function util.walk(t, f)
4005   local typ = type(t)
4006   if typ == "string" then
4007     f(t)
4008   elseif typ == "table" then
4009     local i = 1
4010     local n
4011     n = t[i]
4012     while n do
4013       util.walk(n, f)
4014       i = i + 1
4015       n = t[i]
4016     end
4017   elseif typ == "function" then
4018     local ok, val = pcall(t)
4019     if ok then
4020       util.walk(val, f)
4021     end
4022   else
4023     f(tostring(t))
4024   end
4025 end
```

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

```
4026 function util.flatten(ary)
4027   local new = {}
4028   for _,v in ipairs(ary) do
4029     if type(v) == "table" then
4030       for _,w in ipairs(util.flatten(v)) do
4031         new[#new + 1] = w
4032       end
4033     else
4034       new[#new + 1] = v
4035     end
4036   end
4037   return new
4038 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.

```
4039 function util.rope_to_string(rope)
4040   local buffer = {}
4041   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4042   return table.concat(buffer)
```

```
4043 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.

```
4044 function util.rope_last(rope)
4045   if #rope == 0 then
4046     return nil
4047   else
4048     local l = rope[#rope]
4049     if type(l) == "table" then
4050       return util.rope_last(l)
4051     else
4052       return l
4053     end
4054   end
4055 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  $1 \leq i \leq \#ary$ .

```
4056 function util.intersperse(ary, x)
4057   local new = {}
4058   local l = #ary
4059   for i,v in ipairs(ary) do
4060     local n = #new
4061     new[n + 1] = v
4062     if i ~= l then
4063       new[n + 2] = x
4064     end
4065   end
4066   return new
4067 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  $1 \leq i \leq \#ary$ .

```
4068 function util.map(ary, f)
4069   local new = {}
4070   for i,v in ipairs(ary) do
4071     new[i] = f(v)
4072   end
4073   return new
4074 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.

```
4075 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
4076 local char_escapes_list = ""
4077 for i,_ in pairs(char_escapes) do
4078     char_escapes_list = char_escapes_list .. i
4079 end
```

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

```
4080 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.

```
4081 if string_escapes then
4082     for k,v in pairs(string_escapes) do
4083         escapable = P(k) / v + escapable
4084     end
4085 end
```

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

```
4086 local escape_string = Cs((escapable + any)^0)
```

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

```
4087 return function(s)
4088     return lpeg.match(escape_string, s)
4089 end
4090 end
```

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

```
4091 function util.pathname(dir, file)
4092     if #dir == 0 then
4093         return file
4094     else
4095         return dir .. "/" .. file
4096     end
4097 end
```



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

```
4098 function util.salt(options)
4099   local opt_string = {}
4100   for k, _ in pairs(defaultOptions) do
4101     local v = options[k]
4102     if type(v) == "table" then
4103       for _, i in ipairs(v) do
4104         opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4105       end

```

The `cacheDir` option is disregarded.

```
4106     elseif k ~= "cacheDir" then
4107       opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4108     end
4109   end
4110   table.sort(opt_string)
4111   local salt = table.concat(opt_string, ",")
4112     .. "," .. metadata.version
4113   return salt
4114 end

```

The `util.warning` method produces a warning `s` that is unrelated to any specific markdown text being processed. For warnings that are specific to a markdown text, use `writer->warning` function.

```
4115 function util.warning(s)
4116   io.stderr:write("Warning: " .. s .. "\n")
4117 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.

```
4118 local entities = {}
4119
4120 local character_entities = {
4121   ["Tab"] = 9,
4122   ["NewLine"] = 10,
4123   ["excl"] = 33,
4124   ["QUOT"] = 34,
4125   ["quot"] = 34,
4126   ["num"] = 35,
4127   ["dollar"] = 36,
4128   ["percnt"] = 37,
4129   ["AMP"] = 38,
4130   ["amp"] = 38,
4131   ["apos"] = 39,

```

```

4132 ["lpar"] = 40,
4133 ["rpar"] = 41,
4134 ["ast"] = 42,
4135 ["midast"] = 42,
4136 ["plus"] = 43,
4137 ["comma"] = 44,
4138 ["period"] = 46,
4139 ["sol"] = 47,
4140 ["colon"] = 58,
4141 ["semi"] = 59,
4142 ["LT"] = 60,
4143 ["lt"] = 60,
4144 ["nvlt"] = {60, 8402},
4145 ["bne"] = {61, 8421},
4146 ["equals"] = 61,
4147 ["GT"] = 62,
4148 ["gt"] = 62,
4149 ["nvgt"] = {62, 8402},
4150 ["quest"] = 63,
4151 ["commat"] = 64,
4152 ["lbrack"] = 91,
4153 ["lsqb"] = 91,
4154 ["bsol"] = 92,
4155 ["rbrack"] = 93,
4156 ["rsqb"] = 93,
4157 ["Hat"] = 94,
4158 ["UnderBar"] = 95,
4159 ["lowbar"] = 95,
4160 ["DiacriticalGrave"] = 96,
4161 ["grave"] = 96,
4162 ["fjlig"] = {102, 106},
4163 ["lbrace"] = 123,
4164 ["lcub"] = 123,
4165 ["VerticalLine"] = 124,
4166 ["verbar"] = 124,
4167 ["vert"] = 124,
4168 ["rbrace"] = 125,
4169 ["rcub"] = 125,
4170 ["NonBreakingSpace"] = 160,
4171 ["nbsp"] = 160,
4172 ["iexcl"] = 161,
4173 ["cent"] = 162,
4174 ["pound"] = 163,
4175 ["curren"] = 164,
4176 ["yen"] = 165,
4177 ["brvbar"] = 166,
4178 ["sect"] = 167,

```

4179 ["Dot"] = 168,  
4180 ["DoubleDot"] = 168,  
4181 ["die"] = 168,  
4182 ["uml"] = 168,  
4183 ["COPY"] = 169,  
4184 ["copy"] = 169,  
4185 ["ordf"] = 170,  
4186 ["laquo"] = 171,  
4187 ["not"] = 172,  
4188 ["shy"] = 173,  
4189 ["REG"] = 174,  
4190 ["circledR"] = 174,  
4191 ["reg"] = 174,  
4192 ["macr"] = 175,  
4193 ["strns"] = 175,  
4194 ["deg"] = 176,  
4195 ["PlusMinus"] = 177,  
4196 ["plusmn"] = 177,  
4197 ["pm"] = 177,  
4198 ["sup2"] = 178,  
4199 ["sup3"] = 179,  
4200 ["DiacriticalAcute"] = 180,  
4201 ["acute"] = 180,  
4202 ["micro"] = 181,  
4203 ["para"] = 182,  
4204 ["CenterDot"] = 183,  
4205 ["centerdot"] = 183,  
4206 ["middot"] = 183,  
4207 ["Cedilla"] = 184,  
4208 ["cedil"] = 184,  
4209 ["sup1"] = 185,  
4210 ["ordm"] = 186,  
4211 ["raquo"] = 187,  
4212 ["frac14"] = 188,  
4213 ["frac12"] = 189,  
4214 ["half"] = 189,  
4215 ["frac34"] = 190,  
4216 ["iquest"] = 191,  
4217 ["Agrave"] = 192,  
4218 ["Aacute"] = 193,  
4219 ["Acirc"] = 194,  
4220 ["Atilde"] = 195,  
4221 ["Auml"] = 196,  
4222 ["Aring"] = 197,  
4223 ["angst"] = 197,  
4224 ["AElig"] = 198,  
4225 ["Ccedil"] = 199,

4226 ["Egrave"] = 200,  
4227 ["Eacute"] = 201,  
4228 ["Ecirc"] = 202,  
4229 ["Euml"] = 203,  
4230 ["Igrave"] = 204,  
4231 ["Iacute"] = 205,  
4232 ["Icirc"] = 206,  
4233 ["Iuml"] = 207,  
4234 ["ETH"] = 208,  
4235 ["Ntilde"] = 209,  
4236 ["Ograve"] = 210,  
4237 ["Oacute"] = 211,  
4238 ["Ocirc"] = 212,  
4239 ["Otilde"] = 213,  
4240 ["Ouml"] = 214,  
4241 ["times"] = 215,  
4242 ["Oslash"] = 216,  
4243 ["Ugrave"] = 217,  
4244 ["Uacute"] = 218,  
4245 ["Ucirc"] = 219,  
4246 ["Uuml"] = 220,  
4247 ["Yacute"] = 221,  
4248 ["THORN"] = 222,  
4249 ["szlig"] = 223,  
4250 ["agrave"] = 224,  
4251 ["aacute"] = 225,  
4252 ["acirc"] = 226,  
4253 ["atilde"] = 227,  
4254 ["auml"] = 228,  
4255 ["aring"] = 229,  
4256 ["aelig"] = 230,  
4257 ["ccedil"] = 231,  
4258 ["egrave"] = 232,  
4259 ["eacute"] = 233,  
4260 ["ecirc"] = 234,  
4261 ["euml"] = 235,  
4262 ["igrave"] = 236,  
4263 ["iacute"] = 237,  
4264 ["icirc"] = 238,  
4265 ["iuml"] = 239,  
4266 ["eth"] = 240,  
4267 ["ntilde"] = 241,  
4268 ["ograve"] = 242,  
4269 ["oacute"] = 243,  
4270 ["ocirc"] = 244,  
4271 ["otilde"] = 245,  
4272 ["ouml"] = 246,

4273 ["div"] = 247,  
4274 ["divide"] = 247,  
4275 ["oslash"] = 248,  
4276 ["ugrave"] = 249,  
4277 ["uacute"] = 250,  
4278 ["ucirc"] = 251,  
4279 ["uuml"] = 252,  
4280 ["yacute"] = 253,  
4281 ["thorn"] = 254,  
4282 ["yuml"] = 255,  
4283 ["Amacr"] = 256,  
4284 ["amacr"] = 257,  
4285 ["Abreve"] = 258,  
4286 ["abreve"] = 259,  
4287 ["Aogon"] = 260,  
4288 ["aogon"] = 261,  
4289 ["Cacute"] = 262,  
4290 ["cacute"] = 263,  
4291 ["Ccirc"] = 264,  
4292 ["ccirc"] = 265,  
4293 ["Cdot"] = 266,  
4294 ["cdot"] = 267,  
4295 ["Ccaron"] = 268,  
4296 ["ccaron"] = 269,  
4297 ["Dcaron"] = 270,  
4298 ["dcaron"] = 271,  
4299 ["Dstrok"] = 272,  
4300 ["dstrok"] = 273,  
4301 ["Emacr"] = 274,  
4302 ["emacr"] = 275,  
4303 ["Edot"] = 278,  
4304 ["edot"] = 279,  
4305 ["Eogon"] = 280,  
4306 ["eogon"] = 281,  
4307 ["Ecaron"] = 282,  
4308 ["ecaron"] = 283,  
4309 ["Gcirc"] = 284,  
4310 ["gcirc"] = 285,  
4311 ["Gbreve"] = 286,  
4312 ["gbreve"] = 287,  
4313 ["Gdot"] = 288,  
4314 ["gdot"] = 289,  
4315 ["Gcedil"] = 290,  
4316 ["Hcirc"] = 292,  
4317 ["hcirc"] = 293,  
4318 ["Hstrok"] = 294,  
4319 ["hstrok"] = 295,

4320 ["Itilde"] = 296,  
4321 ["itilde"] = 297,  
4322 ["Imacr"] = 298,  
4323 ["imacr"] = 299,  
4324 ["Iogon"] = 302,  
4325 ["iogon"] = 303,  
4326 ["Idot"] = 304,  
4327 ["imath"] = 305,  
4328 ["inodot"] = 305,  
4329 ["IJlig"] = 306,  
4330 ["ijlig"] = 307,  
4331 ["Jcirc"] = 308,  
4332 ["jcirc"] = 309,  
4333 ["Kcedil"] = 310,  
4334 ["kcedil"] = 311,  
4335 ["kgreen"] = 312,  
4336 ["Lacute"] = 313,  
4337 ["lacute"] = 314,  
4338 ["Lcedil"] = 315,  
4339 ["lcedil"] = 316,  
4340 ["Lcaron"] = 317,  
4341 ["lcaron"] = 318,  
4342 ["Lmidot"] = 319,  
4343 ["lmidot"] = 320,  
4344 ["Lstrok"] = 321,  
4345 ["lstrok"] = 322,  
4346 ["Nacute"] = 323,  
4347 ["nacute"] = 324,  
4348 ["Ncedil"] = 325,  
4349 ["ncedil"] = 326,  
4350 ["Ncaron"] = 327,  
4351 ["ncaron"] = 328,  
4352 ["napos"] = 329,  
4353 ["ENG"] = 330,  
4354 ["eng"] = 331,  
4355 ["Omacr"] = 332,  
4356 ["omacr"] = 333,  
4357 ["Odblac"] = 336,  
4358 ["odblac"] = 337,  
4359 ["OElig"] = 338,  
4360 ["oelig"] = 339,  
4361 ["Racute"] = 340,  
4362 ["racute"] = 341,  
4363 ["Rcedil"] = 342,  
4364 ["rcedil"] = 343,  
4365 ["Rcaron"] = 344,  
4366 ["rcaron"] = 345,

4367 ["Sacute"] = 346,  
4368 ["sacute"] = 347,  
4369 ["Scirc"] = 348,  
4370 ["scirc"] = 349,  
4371 ["Scedil"] = 350,  
4372 ["scedil"] = 351,  
4373 ["Scaron"] = 352,  
4374 ["scaron"] = 353,  
4375 ["Tcedil"] = 354,  
4376 ["tcedil"] = 355,  
4377 ["Tcaron"] = 356,  
4378 ["tcaron"] = 357,  
4379 ["Tstrok"] = 358,  
4380 ["tstrok"] = 359,  
4381 ["Utilde"] = 360,  
4382 ["utilde"] = 361,  
4383 ["Umacr"] = 362,  
4384 ["umacr"] = 363,  
4385 ["Ubreve"] = 364,  
4386 ["ubreve"] = 365,  
4387 ["Uring"] = 366,  
4388 ["uring"] = 367,  
4389 ["Udblac"] = 368,  
4390 ["udblac"] = 369,  
4391 ["Uogon"] = 370,  
4392 ["uogon"] = 371,  
4393 ["Wcirc"] = 372,  
4394 ["wcirc"] = 373,  
4395 ["Ycirc"] = 374,  
4396 ["ycirc"] = 375,  
4397 ["Yuml"] = 376,  
4398 ["Zacute"] = 377,  
4399 ["zacute"] = 378,  
4400 ["Zdot"] = 379,  
4401 ["zdot"] = 380,  
4402 ["Zcaron"] = 381,  
4403 ["zcaron"] = 382,  
4404 ["fnof"] = 402,  
4405 ["imped"] = 437,  
4406 ["gacute"] = 501,  
4407 ["jmath"] = 567,  
4408 ["circ"] = 710,  
4409 ["Hacek"] = 711,  
4410 ["caron"] = 711,  
4411 ["Breve"] = 728,  
4412 ["breve"] = 728,  
4413 ["DiacriticalDot"] = 729,

4414 ["dot"] = 729,  
4415 ["ring"] = 730,  
4416 ["ogon"] = 731,  
4417 ["DiacriticalTilde"] = 732,  
4418 ["tilde"] = 732,  
4419 ["DiacriticalDoubleAcute"] = 733,  
4420 ["dblac"] = 733,  
4421 ["DownBreve"] = 785,  
4422 ["Alpha"] = 913,  
4423 ["Beta"] = 914,  
4424 ["Gamma"] = 915,  
4425 ["Delta"] = 916,  
4426 ["Epsilon"] = 917,  
4427 ["Zeta"] = 918,  
4428 ["Eta"] = 919,  
4429 ["Theta"] = 920,  
4430 ["Iota"] = 921,  
4431 ["Kappa"] = 922,  
4432 ["Lambda"] = 923,  
4433 ["Mu"] = 924,  
4434 ["Nu"] = 925,  
4435 ["Xi"] = 926,  
4436 ["Omicron"] = 927,  
4437 ["Pi"] = 928,  
4438 ["Rho"] = 929,  
4439 ["Sigma"] = 931,  
4440 ["Tau"] = 932,  
4441 ["Upsilon"] = 933,  
4442 ["Phi"] = 934,  
4443 ["Chi"] = 935,  
4444 ["Psi"] = 936,  
4445 ["Omega"] = 937,  
4446 ["ohm"] = 937,  
4447 ["alpha"] = 945,  
4448 ["beta"] = 946,  
4449 ["gamma"] = 947,  
4450 ["delta"] = 948,  
4451 ["epsi"] = 949,  
4452 ["epsilon"] = 949,  
4453 ["zeta"] = 950,  
4454 ["eta"] = 951,  
4455 ["theta"] = 952,  
4456 ["iota"] = 953,  
4457 ["kappa"] = 954,  
4458 ["lambda"] = 955,  
4459 ["mu"] = 956,  
4460 ["nu"] = 957,



4461 ["xi"] = 958,  
4462 ["omicron"] = 959,  
4463 ["pi"] = 960,  
4464 ["rho"] = 961,  
4465 ["sigmaf"] = 962,  
4466 ["sigmav"] = 962,  
4467 ["varsigma"] = 962,  
4468 ["sigma"] = 963,  
4469 ["tau"] = 964,  
4470 ["upsilon"] = 965,  
4471 ["upsilon"] = 965,  
4472 ["phi"] = 966,  
4473 ["chi"] = 967,  
4474 ["psi"] = 968,  
4475 ["omega"] = 969,  
4476 ["thetasym"] = 977,  
4477 ["thetav"] = 977,  
4478 ["vartheta"] = 977,  
4479 ["Upsilon"] = 978,  
4480 ["upsih"] = 978,  
4481 ["phiv"] = 981,  
4482 ["straightphi"] = 981,  
4483 ["varphi"] = 981,  
4484 ["piv"] = 982,  
4485 ["varpi"] = 982,  
4486 ["Gammad"] = 988,  
4487 ["digamma"] = 989,  
4488 ["gammad"] = 989,  
4489 ["kappav"] = 1008,  
4490 ["varkappa"] = 1008,  
4491 ["rhov"] = 1009,  
4492 ["varrho"] = 1009,  
4493 ["epsiv"] = 1013,  
4494 ["straightepsilon"] = 1013,  
4495 ["varepsilon"] = 1013,  
4496 ["backepsilon"] = 1014,  
4497 ["bepsi"] = 1014,  
4498 ["IOcy"] = 1025,  
4499 ["DJcy"] = 1026,  
4500 ["GJcy"] = 1027,  
4501 ["Jukcy"] = 1028,  
4502 ["DScy"] = 1029,  
4503 ["Iukcy"] = 1030,  
4504 ["YIcy"] = 1031,  
4505 ["Jsercy"] = 1032,  
4506 ["LJcy"] = 1033,  
4507 ["NJcy"] = 1034,

4508 ["TSHcy"] = 1035,  
4509 ["KJcy"] = 1036,  
4510 ["Ubrcy"] = 1038,  
4511 ["DZcy"] = 1039,  
4512 ["Acy"] = 1040,  
4513 ["Bcy"] = 1041,  
4514 ["Vcy"] = 1042,  
4515 ["Gcy"] = 1043,  
4516 ["Dcy"] = 1044,  
4517 ["IEcy"] = 1045,  
4518 ["ZHcy"] = 1046,  
4519 ["Zcy"] = 1047,  
4520 ["Icy"] = 1048,  
4521 ["Jcy"] = 1049,  
4522 ["Kcy"] = 1050,  
4523 ["Lcy"] = 1051,  
4524 ["Mcy"] = 1052,  
4525 ["Ncy"] = 1053,  
4526 ["Ocy"] = 1054,  
4527 ["Pcy"] = 1055,  
4528 ["Rcy"] = 1056,  
4529 ["Scy"] = 1057,  
4530 ["Tcy"] = 1058,  
4531 ["Ucy"] = 1059,  
4532 ["Fcy"] = 1060,  
4533 ["KHcy"] = 1061,  
4534 ["TScy"] = 1062,  
4535 ["CHcy"] = 1063,  
4536 ["SHcy"] = 1064,  
4537 ["SHCHcy"] = 1065,  
4538 ["HARDcy"] = 1066,  
4539 ["Ycy"] = 1067,  
4540 ["SOFTcy"] = 1068,  
4541 ["Ecy"] = 1069,  
4542 ["YUcy"] = 1070,  
4543 ["YAcy"] = 1071,  
4544 ["acy"] = 1072,  
4545 ["bcy"] = 1073,  
4546 ["vcy"] = 1074,  
4547 ["gcy"] = 1075,  
4548 ["dcy"] = 1076,  
4549 ["iecy"] = 1077,  
4550 ["zhcy"] = 1078,  
4551 ["zcy"] = 1079,  
4552 ["icy"] = 1080,  
4553 ["jcy"] = 1081,  
4554 ["kcy"] = 1082,

4555 ["lcy"] = 1083,  
4556 ["mcy"] = 1084,  
4557 ["ncy"] = 1085,  
4558 ["ocy"] = 1086,  
4559 ["pcy"] = 1087,  
4560 ["rcy"] = 1088,  
4561 ["scy"] = 1089,  
4562 ["tcy"] = 1090,  
4563 ["ucy"] = 1091,  
4564 ["fcy"] = 1092,  
4565 ["khcy"] = 1093,  
4566 ["tscy"] = 1094,  
4567 ["chcy"] = 1095,  
4568 ["shcy"] = 1096,  
4569 ["shchcy"] = 1097,  
4570 ["hardcy"] = 1098,  
4571 ["ycy"] = 1099,  
4572 ["softcy"] = 1100,  
4573 ["ecy"] = 1101,  
4574 ["yucy"] = 1102,  
4575 ["yacy"] = 1103,  
4576 ["iocy"] = 1105,  
4577 ["djcy"] = 1106,  
4578 ["gjcy"] = 1107,  
4579 ["jukcy"] = 1108,  
4580 ["dscy"] = 1109,  
4581 ["iukcy"] = 1110,  
4582 ["yicy"] = 1111,  
4583 ["jsercy"] = 1112,  
4584 ["ljcy"] = 1113,  
4585 ["njcy"] = 1114,  
4586 ["tshcy"] = 1115,  
4587 ["kjcy"] = 1116,  
4588 ["ubrky"] = 1118,  
4589 ["dzcy"] = 1119,  
4590 ["ensp"] = 8194,  
4591 ["emsp"] = 8195,  
4592 ["emsp13"] = 8196,  
4593 ["emsp14"] = 8197,  
4594 ["numsp"] = 8199,  
4595 ["puncsp"] = 8200,  
4596 ["ThinSpace"] = 8201,  
4597 ["thinsp"] = 8201,  
4598 ["VeryThinSpace"] = 8202,  
4599 ["hairsp"] = 8202,  
4600 ["NegativeMediumSpace"] = 8203,  
4601 ["NegativeThickSpace"] = 8203,

4602 ["NegativeThinSpace"] = 8203,  
4603 ["NegativeVeryThinSpace"] = 8203,  
4604 ["ZeroWidthSpace"] = 8203,  
4605 ["zwnj"] = 8204,  
4606 ["zwj"] = 8205,  
4607 ["lrm"] = 8206,  
4608 ["rlm"] = 8207,  
4609 ["dash"] = 8208,  
4610 ["hyphen"] = 8208,  
4611 ["ndash"] = 8211,  
4612 ["mdash"] = 8212,  
4613 ["horbar"] = 8213,  
4614 ["Verbar"] = 8214,  
4615 ["Vert"] = 8214,  
4616 ["OpenCurlyQuote"] = 8216,  
4617 ["lsquo"] = 8216,  
4618 ["CloseCurlyQuote"] = 8217,  
4619 ["rsquo"] = 8217,  
4620 ["rsquor"] = 8217,  
4621 ["lsquor"] = 8218,  
4622 ["sbquo"] = 8218,  
4623 ["OpenCurlyDoubleQuote"] = 8220,  
4624 ["ldquo"] = 8220,  
4625 ["CloseCurlyDoubleQuote"] = 8221,  
4626 ["rdquo"] = 8221,  
4627 ["rdquor"] = 8221,  
4628 ["bdquo"] = 8222,  
4629 ["ldquor"] = 8222,  
4630 ["dagger"] = 8224,  
4631 ["Dagger"] = 8225,  
4632 ["ddagger"] = 8225,  
4633 ["bull"] = 8226,  
4634 ["bullet"] = 8226,  
4635 ["nldr"] = 8229,  
4636 ["hellip"] = 8230,  
4637 ["mldr"] = 8230,  
4638 ["permil"] = 8240,  
4639 ["pertenk"] = 8241,  
4640 ["prime"] = 8242,  
4641 ["Prime"] = 8243,  
4642 ["tprime"] = 8244,  
4643 ["backprime"] = 8245,  
4644 ["bprime"] = 8245,  
4645 ["lsaquo"] = 8249,  
4646 ["rsaquo"] = 8250,  
4647 ["OverBar"] = 8254,  
4648 ["oline"] = 8254,

4649 ["caret"] = 8257,  
 4650 ["hybull"] = 8259,  
 4651 ["frasl"] = 8260,  
 4652 ["bsemi"] = 8271,  
 4653 ["qprime"] = 8279,  
 4654 ["MediumSpace"] = 8287,  
 4655 ["ThickSpace"] = {8287, 8202},  
 4656 ["NoBreak"] = 8288,  
 4657 ["ApplyFunction"] = 8289,  
 4658 ["af"] = 8289,  
 4659 ["InvisibleTimes"] = 8290,  
 4660 ["it"] = 8290,  
 4661 ["InvisibleComma"] = 8291,  
 4662 ["ic"] = 8291,  
 4663 ["euro"] = 8364,  
 4664 ["TripleDot"] = 8411,  
 4665 ["tdot"] = 8411,  
 4666 ["DotDot"] = 8412,  
 4667 ["Copf"] = 8450,  
 4668 ["complexes"] = 8450,  
 4669 ["incare"] = 8453,  
 4670 ["gscr"] = 8458,  
 4671 ["HilbertSpace"] = 8459,  
 4672 ["Hscr"] = 8459,  
 4673 ["hamilt"] = 8459,  
 4674 ["Hfr"] = 8460,  
 4675 ["Poincareplane"] = 8460,  
 4676 ["Hopf"] = 8461,  
 4677 ["quaternions"] = 8461,  
 4678 ["planckh"] = 8462,  
 4679 ["hbar"] = 8463,  
 4680 ["hslash"] = 8463,  
 4681 ["planck"] = 8463,  
 4682 ["plankv"] = 8463,  
 4683 ["Iscr"] = 8464,  
 4684 ["imagline"] = 8464,  
 4685 ["Ifr"] = 8465,  
 4686 ["Im"] = 8465,  
 4687 ["image"] = 8465,  
 4688 ["imagpart"] = 8465,  
 4689 ["Laplacetrif"] = 8466,  
 4690 ["Lscr"] = 8466,  
 4691 ["lagran"] = 8466,  
 4692 ["ell"] = 8467,  
 4693 ["Nopf"] = 8469,  
 4694 ["naturals"] = 8469,  
 4695 ["numero"] = 8470,

4696 ["copysr"] = 8471,  
4697 ["weierp"] = 8472,  
4698 ["wp"] = 8472,  
4699 ["Popf"] = 8473,  
4700 ["primes"] = 8473,  
4701 ["Qopf"] = 8474,  
4702 ["rationals"] = 8474,  
4703 ["Rscr"] = 8475,  
4704 ["realine"] = 8475,  
4705 ["Re"] = 8476,  
4706 ["Rfr"] = 8476,  
4707 ["real"] = 8476,  
4708 ["realpart"] = 8476,  
4709 ["Ropf"] = 8477,  
4710 ["reals"] = 8477,  
4711 ["rx"] = 8478,  
4712 ["TRADE"] = 8482,  
4713 ["trade"] = 8482,  
4714 ["Zopf"] = 8484,  
4715 ["integers"] = 8484,  
4716 ["mho"] = 8487,  
4717 ["Zfr"] = 8488,  
4718 ["zeetrf"] = 8488,  
4719 ["iiota"] = 8489,  
4720 ["Bernoullis"] = 8492,  
4721 ["Bscr"] = 8492,  
4722 ["bernou"] = 8492,  
4723 ["Cayleys"] = 8493,  
4724 ["Cfr"] = 8493,  
4725 ["escr"] = 8495,  
4726 ["Escr"] = 8496,  
4727 ["expectation"] = 8496,  
4728 ["Fouriertrf"] = 8497,  
4729 ["Fscr"] = 8497,  
4730 ["Mellintrf"] = 8499,  
4731 ["Mscr"] = 8499,  
4732 ["phmmat"] = 8499,  
4733 ["order"] = 8500,  
4734 ["orderof"] = 8500,  
4735 ["oscr"] = 8500,  
4736 ["alefsym"] = 8501,  
4737 ["aleph"] = 8501,  
4738 ["beth"] = 8502,  
4739 ["gimel"] = 8503,  
4740 ["daleth"] = 8504,  
4741 ["CapitalDifferentialD"] = 8517,  
4742 ["DD"] = 8517,

4743 ["DifferentialD"] = 8518,  
 4744 ["dd"] = 8518,  
 4745 ["ExponentialE"] = 8519,  
 4746 ["ee"] = 8519,  
 4747 ["exponentiale"] = 8519,  
 4748 ["ImaginaryI"] = 8520,  
 4749 ["ii"] = 8520,  
 4750 ["frac13"] = 8531,  
 4751 ["frac23"] = 8532,  
 4752 ["frac15"] = 8533,  
 4753 ["frac25"] = 8534,  
 4754 ["frac35"] = 8535,  
 4755 ["frac45"] = 8536,  
 4756 ["frac16"] = 8537,  
 4757 ["frac56"] = 8538,  
 4758 ["frac18"] = 8539,  
 4759 ["frac38"] = 8540,  
 4760 ["frac58"] = 8541,  
 4761 ["frac78"] = 8542,  
 4762 ["LeftArrow"] = 8592,  
 4763 ["ShortLeftArrow"] = 8592,  
 4764 ["larr"] = 8592,  
 4765 ["leftarrow"] = 8592,  
 4766 ["slarr"] = 8592,  
 4767 ["ShortUpArrow"] = 8593,  
 4768 ["UpArrow"] = 8593,  
 4769 ["uarr"] = 8593,  
 4770 ["uparrow"] = 8593,  
 4771 ["RightArrow"] = 8594,  
 4772 ["ShortRightArrow"] = 8594,  
 4773 ["rarr"] = 8594,  
 4774 ["rightarrow"] = 8594,  
 4775 ["srarr"] = 8594,  
 4776 ["DownArrow"] = 8595,  
 4777 ["ShortDownArrow"] = 8595,  
 4778 ["darr"] = 8595,  
 4779 ["downarrow"] = 8595,  
 4780 ["LeftRightArrow"] = 8596,  
 4781 ["harr"] = 8596,  
 4782 ["leftrightarrow"] = 8596,  
 4783 ["UpDownArrow"] = 8597,  
 4784 ["updownarrow"] = 8597,  
 4785 ["varr"] = 8597,  
 4786 ["UpperLeftArrow"] = 8598,  
 4787 ["nwarr"] = 8598,  
 4788 ["nwarrow"] = 8598,  
 4789 ["UpperRightArrow"] = 8599,

```

4790 ["nearr"] = 8599,
4791 ["nearrow"] = 8599,
4792 ["LowerRightArrow"] = 8600,
4793 ["searr"] = 8600,
4794 ["searrow"] = 8600,
4795 ["LowerLeftArrow"] = 8601,
4796 ["swarr"] = 8601,
4797 ["swarrow"] = 8601,
4798 ["nlarr"] = 8602,
4799 ["nleftarrow"] = 8602,
4800 ["nrarr"] = 8603,
4801 ["nrightarrow"] = 8603,
4802 ["nrarrw"] = {8605, 824},
4803 ["rarrw"] = 8605,
4804 ["rightsquigarrow"] = 8605,
4805 ["Larr"] = 8606,
4806 ["twoheadleftarrow"] = 8606,
4807 ["Uarr"] = 8607,
4808 ["Rarr"] = 8608,
4809 ["twoheadrightarrow"] = 8608,
4810 ["Darr"] = 8609,
4811 ["larrtl"] = 8610,
4812 ["leftarrowtail"] = 8610,
4813 ["rarrtl"] = 8611,
4814 ["rightarrowtail"] = 8611,
4815 ["LeftTeeArrow"] = 8612,
4816 ["mapstoleft"] = 8612,
4817 ["UpTeeArrow"] = 8613,
4818 ["mapstoup"] = 8613,
4819 ["RightTeeArrow"] = 8614,
4820 ["map"] = 8614,
4821 ["mapsto"] = 8614,
4822 ["DownTeeArrow"] = 8615,
4823 ["mapstodown"] = 8615,
4824 ["hookleftarrow"] = 8617,
4825 ["larrhk"] = 8617,
4826 ["hookrightarrow"] = 8618,
4827 ["rarrhk"] = 8618,
4828 ["larrlp"] = 8619,
4829 ["looparrowleft"] = 8619,
4830 ["looparrowright"] = 8620,
4831 ["rarrlp"] = 8620,
4832 ["harrw"] = 8621,
4833 ["leftrightsquigarrow"] = 8621,
4834 ["nharr"] = 8622,
4835 ["nletrightarrow"] = 8622,
4836 ["Lsh"] = 8624,

```



4837 ["lsh"] = 8624,  
 4838 ["Rsh"] = 8625,  
 4839 ["rsh"] = 8625,  
 4840 ["ldsh"] = 8626,  
 4841 ["rdsh"] = 8627,  
 4842 ["crarr"] = 8629,  
 4843 ["cularr"] = 8630,  
 4844 ["curvearrowleft"] = 8630,  
 4845 ["curarr"] = 8631,  
 4846 ["curvearrowright"] = 8631,  
 4847 ["circlearrowleft"] = 8634,  
 4848 ["olarr"] = 8634,  
 4849 ["circlearrowright"] = 8635,  
 4850 ["orarr"] = 8635,  
 4851 ["LeftVector"] = 8636,  
 4852 ["leftharpoonup"] = 8636,  
 4853 ["lharu"] = 8636,  
 4854 ["DownLeftVector"] = 8637,  
 4855 ["leftharpoondown"] = 8637,  
 4856 ["lhard"] = 8637,  
 4857 ["RightUpVector"] = 8638,  
 4858 ["uharr"] = 8638,  
 4859 ["upharpoonright"] = 8638,  
 4860 ["LeftUpVector"] = 8639,  
 4861 ["uharl"] = 8639,  
 4862 ["upharpoonleft"] = 8639,  
 4863 ["RightVector"] = 8640,  
 4864 ["rharu"] = 8640,  
 4865 ["rightharpoonup"] = 8640,  
 4866 ["DownRightVector"] = 8641,  
 4867 ["rhard"] = 8641,  
 4868 ["rightharpoondown"] = 8641,  
 4869 ["RightDownVector"] = 8642,  
 4870 ["dharr"] = 8642,  
 4871 ["downharpoonright"] = 8642,  
 4872 ["LeftDownVector"] = 8643,  
 4873 ["dharl"] = 8643,  
 4874 ["downharpoonleft"] = 8643,  
 4875 ["RightArrowLeftArrow"] = 8644,  
 4876 ["rightleftarrows"] = 8644,  
 4877 ["rlarr"] = 8644,  
 4878 ["UpArrowDownArrow"] = 8645,  
 4879 ["udarr"] = 8645,  
 4880 ["LeftArrowRightArrow"] = 8646,  
 4881 ["leftrightarrows"] = 8646,  
 4882 ["lrarr"] = 8646,  
 4883 ["leftleftarrows"] = 8647,

4884 ["llarr"] = 8647,  
 4885 ["uparrows"] = 8648,  
 4886 ["uuarr"] = 8648,  
 4887 ["rightrightarrows"] = 8649,  
 4888 ["rrarr"] = 8649,  
 4889 ["ddarr"] = 8650,  
 4890 ["downdownarrows"] = 8650,  
 4891 ["ReverseEquilibrium"] = 8651,  
 4892 ["leftrightharpoons"] = 8651,  
 4893 ["lrhar"] = 8651,  
 4894 ["Equilibrium"] = 8652,  
 4895 ["rightleftharpoons"] = 8652,  
 4896 ["rlhar"] = 8652,  
 4897 ["nLeftarrow"] = 8653,  
 4898 ["nlArr"] = 8653,  
 4899 ["nLeftrightarrow"] = 8654,  
 4900 ["nhArr"] = 8654,  
 4901 ["nRightarrow"] = 8655,  
 4902 ["nrArr"] = 8655,  
 4903 ["DoubleLeftArrow"] = 8656,  
 4904 ["Leftarrow"] = 8656,  
 4905 ["lArr"] = 8656,  
 4906 ["DoubleUpArrow"] = 8657,  
 4907 ["Uparrow"] = 8657,  
 4908 ["uArr"] = 8657,  
 4909 ["DoubleRightArrow"] = 8658,  
 4910 ["Implies"] = 8658,  
 4911 ["Rightarrow"] = 8658,  
 4912 ["rArr"] = 8658,  
 4913 ["DoubleDownArrow"] = 8659,  
 4914 ["Downarrow"] = 8659,  
 4915 ["dArr"] = 8659,  
 4916 ["DoubleLeftRightArrow"] = 8660,  
 4917 ["Leftrightarrow"] = 8660,  
 4918 ["hArr"] = 8660,  
 4919 ["iff"] = 8660,  
 4920 ["DoubleUpDownArrow"] = 8661,  
 4921 ["Updownarrow"] = 8661,  
 4922 ["vArr"] = 8661,  
 4923 ["nwArr"] = 8662,  
 4924 ["neArr"] = 8663,  
 4925 ["seArr"] = 8664,  
 4926 ["swArr"] = 8665,  
 4927 ["Lleftarrow"] = 8666,  
 4928 ["lAarr"] = 8666,  
 4929 ["Rrightarrow"] = 8667,  
 4930 ["rAarr"] = 8667,

4931 ["zigrarr"] = 8669,  
 4932 ["LeftArrowBar"] = 8676,  
 4933 ["larrb"] = 8676,  
 4934 ["RightArrowBar"] = 8677,  
 4935 ["rarrb"] = 8677,  
 4936 ["DownArrowUpArrow"] = 8693,  
 4937 ["duarr"] = 8693,  
 4938 ["loarr"] = 8701,  
 4939 ["roarr"] = 8702,  
 4940 ["hoarr"] = 8703,  
 4941 ["ForAll"] = 8704,  
 4942 ["forall"] = 8704,  
 4943 ["comp"] = 8705,  
 4944 ["complement"] = 8705,  
 4945 ["PartialD"] = 8706,  
 4946 ["npart"] = {8706, 824},  
 4947 ["part"] = 8706,  
 4948 ["Exists"] = 8707,  
 4949 ["exist"] = 8707,  
 4950 ["NotExists"] = 8708,  
 4951 ["nexist"] = 8708,  
 4952 ["nexists"] = 8708,  
 4953 ["empty"] = 8709,  
 4954 ["emptyset"] = 8709,  
 4955 ["emptyv"] = 8709,  
 4956 ["varnothing"] = 8709,  
 4957 ["Del"] = 8711,  
 4958 ["nabla"] = 8711,  
 4959 ["Element"] = 8712,  
 4960 ["in"] = 8712,  
 4961 ["isin"] = 8712,  
 4962 ["isinv"] = 8712,  
 4963 ["NotElement"] = 8713,  
 4964 ["notin"] = 8713,  
 4965 ["notinva"] = 8713,  
 4966 ["ReverseElement"] = 8715,  
 4967 ["SuchThat"] = 8715,  
 4968 ["ni"] = 8715,  
 4969 ["niv"] = 8715,  
 4970 ["NotReverseElement"] = 8716,  
 4971 ["notni"] = 8716,  
 4972 ["notniva"] = 8716,  
 4973 ["Product"] = 8719,  
 4974 ["prod"] = 8719,  
 4975 ["Coproduct"] = 8720,  
 4976 ["coprod"] = 8720,  
 4977 ["Sum"] = 8721,

```

4978 ["sum"] = 8721,
4979 ["minus"] = 8722,
4980 ["MinusPlus"] = 8723,
4981 ["mnplus"] = 8723,
4982 ["mp"] = 8723,
4983 ["dotplus"] = 8724,
4984 ["plusdo"] = 8724,
4985 ["Backslash"] = 8726,
4986 ["setminus"] = 8726,
4987 ["setmn"] = 8726,
4988 ["smallsetminus"] = 8726,
4989 ["ssetmn"] = 8726,
4990 ["lowast"] = 8727,
4991 ["SmallCircle"] = 8728,
4992 ["compfn"] = 8728,
4993 ["Sqrt"] = 8730,
4994 ["radic"] = 8730,
4995 ["Proportional"] = 8733,
4996 ["prop"] = 8733,
4997 ["propto"] = 8733,
4998 ["varpropto"] = 8733,
4999 ["vprop"] = 8733,
5000 ["infin"] = 8734,
5001 ["angrt"] = 8735,
5002 ["ang"] = 8736,
5003 ["angle"] = 8736,
5004 ["nang"] = {8736, 8402},
5005 ["angmsd"] = 8737,
5006 ["measuredangle"] = 8737,
5007 ["angsph"] = 8738,
5008 ["VerticalBar"] = 8739,
5009 ["mid"] = 8739,
5010 ["shortmid"] = 8739,
5011 ["smid"] = 8739,
5012 ["NotVerticalBar"] = 8740,
5013 ["nmid"] = 8740,
5014 ["nshortmid"] = 8740,
5015 ["nsmid"] = 8740,
5016 ["DoubleVerticalBar"] = 8741,
5017 ["par"] = 8741,
5018 ["parallel"] = 8741,
5019 ["shortparallel"] = 8741,
5020 ["spar"] = 8741,
5021 ["NotDoubleVerticalBar"] = 8742,
5022 ["npar"] = 8742,
5023 ["nparallel"] = 8742,
5024 ["nshortparallel"] = 8742,

```

5025 ["nspar"] = 8742,  
5026 ["and"] = 8743,  
5027 ["wedge"] = 8743,  
5028 ["or"] = 8744,  
5029 ["vee"] = 8744,  
5030 ["cap"] = 8745,  
5031 ["caps"] = {8745, 65024},  
5032 ["cup"] = 8746,  
5033 ["cups"] = {8746, 65024},  
5034 ["Integral"] = 8747,  
5035 ["int"] = 8747,  
5036 ["Int"] = 8748,  
5037 ["iiint"] = 8749,  
5038 ["tint"] = 8749,  
5039 ["ContourIntegral"] = 8750,  
5040 ["conint"] = 8750,  
5041 ["oint"] = 8750,  
5042 ["Conint"] = 8751,  
5043 ["DoubleContourIntegral"] = 8751,  
5044 ["Cconint"] = 8752,  
5045 ["cwint"] = 8753,  
5046 ["ClockwiseContourIntegral"] = 8754,  
5047 ["cwconint"] = 8754,  
5048 ["CounterClockwiseContourIntegral"] = 8755,  
5049 ["awconint"] = 8755,  
5050 ["Therefore"] = 8756,  
5051 ["there4"] = 8756,  
5052 ["therefore"] = 8756,  
5053 ["Because"] = 8757,  
5054 ["because"] = 8757,  
5055 ["because"] = 8757,  
5056 ["ratio"] = 8758,  
5057 ["Colon"] = 8759,  
5058 ["Proportion"] = 8759,  
5059 ["dotminus"] = 8760,  
5060 ["minusd"] = 8760,  
5061 ["mDDot"] = 8762,  
5062 ["homtht"] = 8763,  
5063 ["Tilde"] = 8764,  
5064 ["nvsim"] = {8764, 8402},  
5065 ["sim"] = 8764,  
5066 ["thicksim"] = 8764,  
5067 ["thksim"] = 8764,  
5068 ["backsim"] = 8765,  
5069 ["bsim"] = 8765,  
5070 ["race"] = {8765, 817},  
5071 ["ac"] = 8766,

5072 ["acE"] = {8766, 819},  
5073 ["mstpos"] = 8766,  
5074 ["acd"] = 8767,  
5075 ["VerticalTilde"] = 8768,  
5076 ["wr"] = 8768,  
5077 ["wreath"] = 8768,  
5078 ["NotTilde"] = 8769,  
5079 ["nsim"] = 8769,  
5080 ["EqualTilde"] = 8770,  
5081 ["NotEqualTilde"] = {8770, 824},  
5082 ["eqsim"] = 8770,  
5083 ["esim"] = 8770,  
5084 ["nesim"] = {8770, 824},  
5085 ["TildeEqual"] = 8771,  
5086 ["sime"] = 8771,  
5087 ["simeq"] = 8771,  
5088 ["NotTildeEqual"] = 8772,  
5089 ["nsime"] = 8772,  
5090 ["nsimeq"] = 8772,  
5091 ["TildeFullEqual"] = 8773,  
5092 ["cong"] = 8773,  
5093 ["simne"] = 8774,  
5094 ["NotTildeFullEqual"] = 8775,  
5095 ["ncong"] = 8775,  
5096 ["TildeTilde"] = 8776,  
5097 ["ap"] = 8776,  
5098 ["approx"] = 8776,  
5099 ["asympt"] = 8776,  
5100 ["thickapprox"] = 8776,  
5101 ["thkap"] = 8776,  
5102 ["NotTildeTilde"] = 8777,  
5103 ["nap"] = 8777,  
5104 ["napprox"] = 8777,  
5105 ["ape"] = 8778,  
5106 ["approxpeq"] = 8778,  
5107 ["apid"] = 8779,  
5108 ["napid"] = {8779, 824},  
5109 ["backcong"] = 8780,  
5110 ["bcong"] = 8780,  
5111 ["CupCap"] = 8781,  
5112 ["asympeq"] = 8781,  
5113 ["nvap"] = {8781, 8402},  
5114 ["Bumpeq"] = 8782,  
5115 ["HumpDownHump"] = 8782,  
5116 ["NotHumpDownHump"] = {8782, 824},  
5117 ["bump"] = 8782,  
5118 ["nbump"] = {8782, 824},

5119 ["HumpEqual"] = 8783,  
5120 ["NotHumpEqual"] = {8783, 824},  
5121 ["bumpe"] = 8783,  
5122 ["bumpeq"] = 8783,  
5123 ["nbumpe"] = {8783, 824},  
5124 ["DotEqual"] = 8784,  
5125 ["doteq"] = 8784,  
5126 ["esdot"] = 8784,  
5127 ["nedot"] = {8784, 824},  
5128 ["doteqdot"] = 8785,  
5129 ["eDot"] = 8785,  
5130 ["efDot"] = 8786,  
5131 ["fallingdotseq"] = 8786,  
5132 ["erDot"] = 8787,  
5133 ["risingdotseq"] = 8787,  
5134 ["Assign"] = 8788,  
5135 ["colone"] = 8788,  
5136 ["coloneq"] = 8788,  
5137 ["ecolon"] = 8789,  
5138 ["eqcolon"] = 8789,  
5139 ["ecir"] = 8790,  
5140 ["eqcirc"] = 8790,  
5141 ["circeq"] = 8791,  
5142 ["cire"] = 8791,  
5143 ["wedgeq"] = 8793,  
5144 ["veeeq"] = 8794,  
5145 ["triangleq"] = 8796,  
5146 ["trie"] = 8796,  
5147 ["equest"] = 8799,  
5148 ["questeq"] = 8799,  
5149 ["NotEqual"] = 8800,  
5150 ["ne"] = 8800,  
5151 ["Congruent"] = 8801,  
5152 ["bnequiv"] = {8801, 8421},  
5153 ["equiv"] = 8801,  
5154 ["NotCongruent"] = 8802,  
5155 ["nequiv"] = 8802,  
5156 ["le"] = 8804,  
5157 ["leq"] = 8804,  
5158 ["nvle"] = {8804, 8402},  
5159 ["GreaterEqual"] = 8805,  
5160 ["ge"] = 8805,  
5161 ["geq"] = 8805,  
5162 ["nvge"] = {8805, 8402},  
5163 ["LessFullEqual"] = 8806,  
5164 ["lE"] = 8806,  
5165 ["leqq"] = 8806,

5166 ["nlE"] = {8806, 824},  
5167 ["nleqq"] = {8806, 824},  
5168 ["GreaterFullEqual"] = 8807,  
5169 ["NotGreaterFullEqual"] = {8807, 824},  
5170 ["gE"] = 8807,  
5171 ["geqq"] = 8807,  
5172 ["ngE"] = {8807, 824},  
5173 ["ngeqq"] = {8807, 824},  
5174 ["lnE"] = 8808,  
5175 ["lneqq"] = 8808,  
5176 ["lvertneqq"] = {8808, 65024},  
5177 ["lvnE"] = {8808, 65024},  
5178 ["gnE"] = 8809,  
5179 ["gneqq"] = 8809,  
5180 ["gvertneqq"] = {8809, 65024},  
5181 ["gvnE"] = {8809, 65024},  
5182 ["Lt"] = 8810,  
5183 ["NestedLessLess"] = 8810,  
5184 ["NotLessLess"] = {8810, 824},  
5185 ["ll"] = 8810,  
5186 ["nLt"] = {8810, 8402},  
5187 ["nLtv"] = {8810, 824},  
5188 ["Gt"] = 8811,  
5189 ["NestedGreaterGreater"] = 8811,  
5190 ["NotGreaterGreater"] = {8811, 824},  
5191 ["gg"] = 8811,  
5192 ["nGt"] = {8811, 8402},  
5193 ["nGtv"] = {8811, 824},  
5194 ["between"] = 8812,  
5195 ["twixt"] = 8812,  
5196 ["NotCupCap"] = 8813,  
5197 ["NotLess"] = 8814,  
5198 ["nless"] = 8814,  
5199 ["nlt"] = 8814,  
5200 ["NotGreater"] = 8815,  
5201 ["ngt"] = 8815,  
5202 ["ngtr"] = 8815,  
5203 ["NotLessEqual"] = 8816,  
5204 ["nle"] = 8816,  
5205 ["nleq"] = 8816,  
5206 ["NotGreaterEqual"] = 8817,  
5207 ["nge"] = 8817,  
5208 ["ngeq"] = 8817,  
5209 ["LessTilde"] = 8818,  
5210 ["lesssim"] = 8818,  
5211 ["lsim"] = 8818,  
5212 ["GreaterTilde"] = 8819,



5213 ["gsim"] = 8819,  
5214 ["gtrsim"] = 8819,  
5215 ["NotLessTilde"] = 8820,  
5216 ["nlsim"] = 8820,  
5217 ["NotGreaterTilde"] = 8821,  
5218 ["ngsim"] = 8821,  
5219 ["LessGreater"] = 8822,  
5220 ["lessgtr"] = 8822,  
5221 ["lg"] = 8822,  
5222 ["GreaterLess"] = 8823,  
5223 ["gl"] = 8823,  
5224 ["gtrless"] = 8823,  
5225 ["NotLessGreater"] = 8824,  
5226 ["ntlg"] = 8824,  
5227 ["NotGreaterLess"] = 8825,  
5228 ["ntgl"] = 8825,  
5229 ["Precedes"] = 8826,  
5230 ["pr"] = 8826,  
5231 ["prec"] = 8826,  
5232 ["Succeeds"] = 8827,  
5233 ["sc"] = 8827,  
5234 ["succ"] = 8827,  
5235 ["PrecedesSlantEqual"] = 8828,  
5236 ["prcue"] = 8828,  
5237 ["preccurlyeq"] = 8828,  
5238 ["SucceedsSlantEqual"] = 8829,  
5239 ["sccue"] = 8829,  
5240 ["succcurlyeq"] = 8829,  
5241 ["PrecedesTilde"] = 8830,  
5242 ["precsim"] = 8830,  
5243 ["prsim"] = 8830,  
5244 ["NotSucceedsTilde"] = {8831, 824},  
5245 ["SucceedsTilde"] = 8831,  
5246 ["scsim"] = 8831,  
5247 ["succsim"] = 8831,  
5248 ["NotPrecedes"] = 8832,  
5249 ["npr"] = 8832,  
5250 ["nprec"] = 8832,  
5251 ["NotSucceeds"] = 8833,  
5252 ["nsc"] = 8833,  
5253 ["nsucc"] = 8833,  
5254 ["NotSubset"] = {8834, 8402},  
5255 ["nsubset"] = {8834, 8402},  
5256 ["sub"] = 8834,  
5257 ["subset"] = 8834,  
5258 ["vnsup"] = {8834, 8402},  
5259 ["NotSuperset"] = {8835, 8402},

5260 ["Superset"] = 8835,  
5261 ["nsupset"] = {8835, 8402},  
5262 ["sup"] = 8835,  
5263 ["supset"] = 8835,  
5264 ["vnsup"] = {8835, 8402},  
5265 ["nsub"] = 8836,  
5266 ["nsup"] = 8837,  
5267 ["SubsetEqual"] = 8838,  
5268 ["sube"] = 8838,  
5269 ["subseteq"] = 8838,  
5270 ["SupersetEqual"] = 8839,  
5271 ["supe"] = 8839,  
5272 ["supseteq"] = 8839,  
5273 ["NotSubsetEqual"] = 8840,  
5274 ["nsube"] = 8840,  
5275 ["nsubseteq"] = 8840,  
5276 ["NotSupersetEqual"] = 8841,  
5277 ["nsupe"] = 8841,  
5278 ["nsupseteq"] = 8841,  
5279 ["subne"] = 8842,  
5280 ["subsetneq"] = 8842,  
5281 ["varsubsetneq"] = {8842, 65024},  
5282 ["vsubne"] = {8842, 65024},  
5283 ["supne"] = 8843,  
5284 ["supsetneq"] = 8843,  
5285 ["varsupsetneq"] = {8843, 65024},  
5286 ["vsupne"] = {8843, 65024},  
5287 ["cupdot"] = 8845,  
5288 ["UnionPlus"] = 8846,  
5289 ["uplus"] = 8846,  
5290 ["NotSquareSubset"] = {8847, 824},  
5291 ["SquareSubset"] = 8847,  
5292 ["sqsub"] = 8847,  
5293 ["sqsubset"] = 8847,  
5294 ["NotSquareSuperset"] = {8848, 824},  
5295 ["SquareSuperset"] = 8848,  
5296 ["sqsup"] = 8848,  
5297 ["sqsupset"] = 8848,  
5298 ["SquareSubsetEqual"] = 8849,  
5299 ["sqsube"] = 8849,  
5300 ["sqsubseteq"] = 8849,  
5301 ["SquareSupersetEqual"] = 8850,  
5302 ["sqsupe"] = 8850,  
5303 ["sqsupseteq"] = 8850,  
5304 ["SquareIntersection"] = 8851,  
5305 ["sqcap"] = 8851,  
5306 ["sqcaps"] = {8851, 65024},

```

5307 ["SquareUnion"] = 8852,
5308 ["sqcup"] = 8852,
5309 ["sqcups"] = {8852, 65024},
5310 ["CirclePlus"] = 8853,
5311 ["oplus"] = 8853,
5312 ["CircleMinus"] = 8854,
5313 ["ominus"] = 8854,
5314 ["CircleTimes"] = 8855,
5315 ["otimes"] = 8855,
5316 ["osol"] = 8856,
5317 ["CircleDot"] = 8857,
5318 ["odot"] = 8857,
5319 ["circledcirc"] = 8858,
5320 ["ocir"] = 8858,
5321 ["circledast"] = 8859,
5322 ["oast"] = 8859,
5323 ["circleddash"] = 8861,
5324 ["odash"] = 8861,
5325 ["boxplus"] = 8862,
5326 ["plusb"] = 8862,
5327 ["boxminus"] = 8863,
5328 ["minusb"] = 8863,
5329 ["boxtimes"] = 8864,
5330 ["timesb"] = 8864,
5331 ["dotsquare"] = 8865,
5332 ["sdotb"] = 8865,
5333 ["RightTee"] = 8866,
5334 ["vdash"] = 8866,
5335 ["LeftTee"] = 8867,
5336 ["dashv"] = 8867,
5337 ["DownTee"] = 8868,
5338 ["top"] = 8868,
5339 ["UpTee"] = 8869,
5340 ["bot"] = 8869,
5341 ["bottom"] = 8869,
5342 ["perp"] = 8869,
5343 ["models"] = 8871,
5344 ["DoubleRightTee"] = 8872,
5345 ["vDash"] = 8872,
5346 ["VDash"] = 8873,
5347 ["Vvdash"] = 8874,
5348 ["VDash"] = 8875,
5349 ["nvdash"] = 8876,
5350 ["nvDash"] = 8877,
5351 ["nVdash"] = 8878,
5352 ["nVDash"] = 8879,
5353 ["prurel"] = 8880,

```

5354 ["LeftTriangle"] = 8882,  
5355 ["vartriangleleft"] = 8882,  
5356 ["vltri"] = 8882,  
5357 ["RightTriangle"] = 8883,  
5358 ["vartriangleright"] = 8883,  
5359 ["vrtri"] = 8883,  
5360 ["LeftTriangleEqual"] = 8884,  
5361 ["ltrie"] = 8884,  
5362 ["nvltrie"] = {8884, 8402},  
5363 ["trianglelefteq"] = 8884,  
5364 ["RightTriangleEqual"] = 8885,  
5365 ["nvrtrie"] = {8885, 8402},  
5366 ["rtrie"] = 8885,  
5367 ["trianglerighteq"] = 8885,  
5368 ["origof"] = 8886,  
5369 ["imof"] = 8887,  
5370 ["multimap"] = 8888,  
5371 ["mumap"] = 8888,  
5372 ["hercon"] = 8889,  
5373 ["intcal"] = 8890,  
5374 ["intercal"] = 8890,  
5375 ["veebar"] = 8891,  
5376 ["barvee"] = 8893,  
5377 ["angrtvb"] = 8894,  
5378 ["ltri"] = 8895,  
5379 ["Wedge"] = 8896,  
5380 ["bigwedge"] = 8896,  
5381 ["xwedge"] = 8896,  
5382 ["Vee"] = 8897,  
5383 ["bigvee"] = 8897,  
5384 ["xvee"] = 8897,  
5385 ["Intersection"] = 8898,  
5386 ["bigcap"] = 8898,  
5387 ["xcap"] = 8898,  
5388 ["Union"] = 8899,  
5389 ["bigcup"] = 8899,  
5390 ["xcup"] = 8899,  
5391 ["Diamond"] = 8900,  
5392 ["diam"] = 8900,  
5393 ["diamond"] = 8900,  
5394 ["sdot"] = 8901,  
5395 ["Star"] = 8902,  
5396 ["sstarf"] = 8902,  
5397 ["divideontimes"] = 8903,  
5398 ["divonx"] = 8903,  
5399 ["bowtie"] = 8904,  
5400 ["ltimes"] = 8905,

5401 ["rtimes"] = 8906,  
5402 ["leftthreetimes"] = 8907,  
5403 ["lthree"] = 8907,  
5404 ["rightthreetimes"] = 8908,  
5405 ["rthree"] = 8908,  
5406 ["backsimeq"] = 8909,  
5407 ["bsime"] = 8909,  
5408 ["curlyvee"] = 8910,  
5409 ["cuvee"] = 8910,  
5410 ["curlywedge"] = 8911,  
5411 ["cuwed"] = 8911,  
5412 ["Sub"] = 8912,  
5413 ["Subset"] = 8912,  
5414 ["Sup"] = 8913,  
5415 ["Supset"] = 8913,  
5416 ["Cap"] = 8914,  
5417 ["Cup"] = 8915,  
5418 ["fork"] = 8916,  
5419 ["pitchfork"] = 8916,  
5420 ["epar"] = 8917,  
5421 ["lessdot"] = 8918,  
5422 ["ltdot"] = 8918,  
5423 ["gtdot"] = 8919,  
5424 ["gtrdot"] = 8919,  
5425 ["L1"] = 8920,  
5426 ["nL1"] = {8920, 824},  
5427 ["Gg"] = 8921,  
5428 ["ggg"] = 8921,  
5429 ["nGg"] = {8921, 824},  
5430 ["LessEqualGreater"] = 8922,  
5431 ["leg"] = 8922,  
5432 ["lesg"] = {8922, 65024},  
5433 ["lesseqgtr"] = 8922,  
5434 ["GreaterEqualLess"] = 8923,  
5435 ["gel"] = 8923,  
5436 ["gesl"] = {8923, 65024},  
5437 ["gtreqless"] = 8923,  
5438 ["cuepr"] = 8926,  
5439 ["curlyeqprec"] = 8926,  
5440 ["cuesc"] = 8927,  
5441 ["curlyeqsucc"] = 8927,  
5442 ["NotPrecedesSlantEqual"] = 8928,  
5443 ["nprcue"] = 8928,  
5444 ["NotSucceedsSlantEqual"] = 8929,  
5445 ["nsccue"] = 8929,  
5446 ["NotSquareSubsetEqual"] = 8930,  
5447 ["nsqsube"] = 8930,

5448 ["NotSquareSupersetEqual"] = 8931,  
5449 ["nsqsupe"] = 8931,  
5450 ["lnsim"] = 8934,  
5451 ["gnsim"] = 8935,  
5452 ["precnsim"] = 8936,  
5453 ["prnsim"] = 8936,  
5454 ["scnsim"] = 8937,  
5455 ["succnsim"] = 8937,  
5456 ["NotLeftTriangle"] = 8938,  
5457 ["nltri"] = 8938,  
5458 ["ntriangleleft"] = 8938,  
5459 ["NotRightTriangle"] = 8939,  
5460 ["nrtri"] = 8939,  
5461 ["ntriangleright"] = 8939,  
5462 ["NotLeftTriangleEqual"] = 8940,  
5463 ["nltrie"] = 8940,  
5464 ["ntrianglelefteq"] = 8940,  
5465 ["NotRightTriangleEqual"] = 8941,  
5466 ["nrtrie"] = 8941,  
5467 ["ntrianglerighteq"] = 8941,  
5468 ["vellip"] = 8942,  
5469 ["ctdot"] = 8943,  
5470 ["utdot"] = 8944,  
5471 ["dtdot"] = 8945,  
5472 ["disin"] = 8946,  
5473 ["isinsv"] = 8947,  
5474 ["isins"] = 8948,  
5475 ["isindot"] = 8949,  
5476 ["notinvc"] = {8949, 824},  
5477 ["notinvc"] = 8950,  
5478 ["notinvb"] = 8951,  
5479 ["isinE"] = 8953,  
5480 ["notinE"] = {8953, 824},  
5481 ["nisd"] = 8954,  
5482 ["xnis"] = 8955,  
5483 ["nis"] = 8956,  
5484 ["notnivc"] = 8957,  
5485 ["notnivb"] = 8958,  
5486 ["barwed"] = 8965,  
5487 ["barwedge"] = 8965,  
5488 ["Barwed"] = 8966,  
5489 ["doublebarwedge"] = 8966,  
5490 ["LeftCeiling"] = 8968,  
5491 ["lceil"] = 8968,  
5492 ["RightCeiling"] = 8969,  
5493 ["rceil"] = 8969,  
5494 ["LeftFloor"] = 8970,

5495 ["lfloor"] = 8970,  
5496 ["RightFloor"] = 8971,  
5497 ["rfloor"] = 8971,  
5498 ["drcrop"] = 8972,  
5499 ["dlcrop"] = 8973,  
5500 ["urcrop"] = 8974,  
5501 ["ulcrop"] = 8975,  
5502 ["bnot"] = 8976,  
5503 ["proflines"] = 8978,  
5504 ["profsurf"] = 8979,  
5505 ["telrec"] = 8981,  
5506 ["target"] = 8982,  
5507 ["ulcorn"] = 8988,  
5508 ["ulcorner"] = 8988,  
5509 ["urcorn"] = 8989,  
5510 ["urcorner"] = 8989,  
5511 ["dlcorn"] = 8990,  
5512 ["llcorner"] = 8990,  
5513 ["drcorn"] = 8991,  
5514 ["lrcorn"] = 8991,  
5515 ["frown"] = 8994,  
5516 ["sfrown"] = 8994,  
5517 ["smile"] = 8995,  
5518 ["ssmile"] = 8995,  
5519 ["cylcty"] = 9005,  
5520 ["profalar"] = 9006,  
5521 ["topbot"] = 9014,  
5522 ["ovbar"] = 9021,  
5523 ["solbar"] = 9023,  
5524 ["angzarr"] = 9084,  
5525 ["lmoust"] = 9136,  
5526 ["lmoustache"] = 9136,  
5527 ["rmoust"] = 9137,  
5528 ["rmoustache"] = 9137,  
5529 ["OverBracket"] = 9140,  
5530 ["tbrk"] = 9140,  
5531 ["UnderBracket"] = 9141,  
5532 ["bbrk"] = 9141,  
5533 ["bbrktbrk"] = 9142,  
5534 ["OverParenthesis"] = 9180,  
5535 ["UnderParenthesis"] = 9181,  
5536 ["OverBrace"] = 9182,  
5537 ["UnderBrace"] = 9183,  
5538 ["trpezium"] = 9186,  
5539 ["elinters"] = 9191,  
5540 ["blank"] = 9251,  
5541 ["circledS"] = 9416,

```
5542 ["oS"] = 9416,  
5543 ["HorizontalLine"] = 9472,  
5544 ["boxh"] = 9472,  
5545 ["boxv"] = 9474,  
5546 ["boxdr"] = 9484,  
5547 ["boxdl"] = 9488,  
5548 ["boxur"] = 9492,  
5549 ["boxul"] = 9496,  
5550 ["boxvr"] = 9500,  
5551 ["boxvl"] = 9508,  
5552 ["boxhd"] = 9516,  
5553 ["boxhu"] = 9524,  
5554 ["boxvh"] = 9532,  
5555 ["boxH"] = 9552,  
5556 ["boxV"] = 9553,  
5557 ["boxdR"] = 9554,  
5558 ["boxDr"] = 9555,  
5559 ["boxDR"] = 9556,  
5560 ["boxdL"] = 9557,  
5561 ["boxDL"] = 9558,  
5562 ["boxDL"] = 9559,  
5563 ["boxuR"] = 9560,  
5564 ["boxUr"] = 9561,  
5565 ["boxUR"] = 9562,  
5566 ["boxuL"] = 9563,  
5567 ["boxUL"] = 9564,  
5568 ["boxUL"] = 9565,  
5569 ["boxvR"] = 9566,  
5570 ["boxVr"] = 9567,  
5571 ["boxVR"] = 9568,  
5572 ["boxvL"] = 9569,  
5573 ["boxVL"] = 9570,  
5574 ["boxVL"] = 9571,  
5575 ["boxHd"] = 9572,  
5576 ["boxhD"] = 9573,  
5577 ["boxHD"] = 9574,  
5578 ["boxHu"] = 9575,  
5579 ["boxhU"] = 9576,  
5580 ["boxHU"] = 9577,  
5581 ["boxvH"] = 9578,  
5582 ["boxVh"] = 9579,  
5583 ["boxVH"] = 9580,  
5584 ["uhblk"] = 9600,  
5585 ["lhblk"] = 9604,  
5586 ["block"] = 9608,  
5587 ["blk14"] = 9617,  
5588 ["blk12"] = 9618,
```



5589 ["blk34"] = 9619,  
5590 ["Square"] = 9633,  
5591 ["squ"] = 9633,  
5592 ["square"] = 9633,  
5593 ["FilledVerySmallSquare"] = 9642,  
5594 ["blacksquare"] = 9642,  
5595 ["squarf"] = 9642,  
5596 ["squf"] = 9642,  
5597 ["EmptyVerySmallSquare"] = 9643,  
5598 ["rect"] = 9645,  
5599 ["marker"] = 9646,  
5600 ["fltns"] = 9649,  
5601 ["bigtriangleup"] = 9651,  
5602 ["xutri"] = 9651,  
5603 ["blacktriangle"] = 9652,  
5604 ["utrif"] = 9652,  
5605 ["triangle"] = 9653,  
5606 ["utri"] = 9653,  
5607 ["blacktriangleright"] = 9656,  
5608 ["rtrif"] = 9656,  
5609 ["rtri"] = 9657,  
5610 ["triangleright"] = 9657,  
5611 ["bigtriangledown"] = 9661,  
5612 ["xdtri"] = 9661,  
5613 ["blacktriangledown"] = 9662,  
5614 ["dtrif"] = 9662,  
5615 ["dtri"] = 9663,  
5616 ["triangledown"] = 9663,  
5617 ["blacktriangleleft"] = 9666,  
5618 ["ltrif"] = 9666,  
5619 ["ltri"] = 9667,  
5620 ["triangleleft"] = 9667,  
5621 ["loz"] = 9674,  
5622 ["lozenge"] = 9674,  
5623 ["cir"] = 9675,  
5624 ["tridot"] = 9708,  
5625 ["bigcirc"] = 9711,  
5626 ["xcirc"] = 9711,  
5627 ["ultri"] = 9720,  
5628 ["urtri"] = 9721,  
5629 ["lltri"] = 9722,  
5630 ["EmptySmallSquare"] = 9723,  
5631 ["FilledSmallSquare"] = 9724,  
5632 ["bigstar"] = 9733,  
5633 ["starf"] = 9733,  
5634 ["star"] = 9734,  
5635 ["phone"] = 9742,

5636 ["female"] = 9792,  
5637 ["male"] = 9794,  
5638 ["spades"] = 9824,  
5639 ["spadesuit"] = 9824,  
5640 ["clubs"] = 9827,  
5641 ["clubsuit"] = 9827,  
5642 ["hearts"] = 9829,  
5643 ["heartsuit"] = 9829,  
5644 ["diamondsuit"] = 9830,  
5645 ["diams"] = 9830,  
5646 ["sung"] = 9834,  
5647 ["flat"] = 9837,  
5648 ["natur"] = 9838,  
5649 ["natural"] = 9838,  
5650 ["sharp"] = 9839,  
5651 ["check"] = 10003,  
5652 ["checkmark"] = 10003,  
5653 ["cross"] = 10007,  
5654 ["malt"] = 10016,  
5655 ["maltese"] = 10016,  
5656 ["sext"] = 10038,  
5657 ["VerticalSeparator"] = 10072,  
5658 ["lbrk"] = 10098,  
5659 ["rbrk"] = 10099,  
5660 ["bsolhsub"] = 10184,  
5661 ["suphsol"] = 10185,  
5662 ["LeftDoubleBracket"] = 10214,  
5663 ["lobrk"] = 10214,  
5664 ["RightDoubleBracket"] = 10215,  
5665 ["robrk"] = 10215,  
5666 ["LeftAngleBracket"] = 10216,  
5667 ["lang"] = 10216,  
5668 ["langle"] = 10216,  
5669 ["RightAngleBracket"] = 10217,  
5670 ["rang"] = 10217,  
5671 ["rangle"] = 10217,  
5672 ["Lang"] = 10218,  
5673 ["Rang"] = 10219,  
5674 ["loang"] = 10220,  
5675 ["roang"] = 10221,  
5676 ["LongLeftArrow"] = 10229,  
5677 ["longleftarrow"] = 10229,  
5678 ["xlarr"] = 10229,  
5679 ["LongRightArrow"] = 10230,  
5680 ["longrightarrow"] = 10230,  
5681 ["xrarr"] = 10230,  
5682 ["LongLeftRightArrow"] = 10231,

5683 ["longleftarrow"] = 10231,  
5684 ["xharr"] = 10231,  
5685 ["DoubleLongLeftArrow"] = 10232,  
5686 ["Longleftarrow"] = 10232,  
5687 ["xlArr"] = 10232,  
5688 ["DoubleLongRightArrow"] = 10233,  
5689 ["Longrightarrow"] = 10233,  
5690 ["xrArr"] = 10233,  
5691 ["DoubleLongLeftRightArrow"] = 10234,  
5692 ["Longleftarrow"] = 10234,  
5693 ["xhArr"] = 10234,  
5694 ["longmapsto"] = 10236,  
5695 ["xmap"] = 10236,  
5696 ["dzigrarr"] = 10239,  
5697 ["nvlArr"] = 10498,  
5698 ["nvrArr"] = 10499,  
5699 ["nvHarr"] = 10500,  
5700 ["Map"] = 10501,  
5701 ["lbarr"] = 10508,  
5702 ["bkarow"] = 10509,  
5703 ["rbarr"] = 10509,  
5704 ["lBarr"] = 10510,  
5705 ["dbkarow"] = 10511,  
5706 ["rBarr"] = 10511,  
5707 ["RBarr"] = 10512,  
5708 ["drbkarow"] = 10512,  
5709 ["DDottrahd"] = 10513,  
5710 ["UpArrowBar"] = 10514,  
5711 ["DownArrowBar"] = 10515,  
5712 ["Rarrtl"] = 10518,  
5713 ["latail"] = 10521,  
5714 ["ratail"] = 10522,  
5715 ["lAtail"] = 10523,  
5716 ["rAtail"] = 10524,  
5717 ["larrfs"] = 10525,  
5718 ["rarrfs"] = 10526,  
5719 ["larrbfs"] = 10527,  
5720 ["rarrbfs"] = 10528,  
5721 ["nwarhk"] = 10531,  
5722 ["nearhk"] = 10532,  
5723 ["hksearrow"] = 10533,  
5724 ["searhk"] = 10533,  
5725 ["hkswarrow"] = 10534,  
5726 ["swarhk"] = 10534,  
5727 ["nwnear"] = 10535,  
5728 ["nesear"] = 10536,  
5729 ["toea"] = 10536,

5730 ["seswar"] = 10537,  
5731 ["tosa"] = 10537,  
5732 ["swnwar"] = 10538,  
5733 ["nrarrc"] = {10547, 824},  
5734 ["rarrc"] = 10547,  
5735 ["cudarrrr"] = 10549,  
5736 ["ldca"] = 10550,  
5737 ["rdca"] = 10551,  
5738 ["cudarrl"] = 10552,  
5739 ["larrpl"] = 10553,  
5740 ["curarrm"] = 10556,  
5741 ["cularrp"] = 10557,  
5742 ["rarrpl"] = 10565,  
5743 ["harrcir"] = 10568,  
5744 ["Uarrocir"] = 10569,  
5745 ["lurdshar"] = 10570,  
5746 ["ldrushar"] = 10571,  
5747 ["LeftRightVector"] = 10574,  
5748 ["RightUpDownVector"] = 10575,  
5749 ["DownLeftRightVector"] = 10576,  
5750 ["LeftUpDownVector"] = 10577,  
5751 ["LeftVectorBar"] = 10578,  
5752 ["RightVectorBar"] = 10579,  
5753 ["RightUpVectorBar"] = 10580,  
5754 ["RightDownVectorBar"] = 10581,  
5755 ["DownLeftVectorBar"] = 10582,  
5756 ["DownRightVectorBar"] = 10583,  
5757 ["LeftUpVectorBar"] = 10584,  
5758 ["LeftDownVectorBar"] = 10585,  
5759 ["LeftTeeVector"] = 10586,  
5760 ["RightTeeVector"] = 10587,  
5761 ["RightUpTeeVector"] = 10588,  
5762 ["RightDownTeeVector"] = 10589,  
5763 ["DownLeftTeeVector"] = 10590,  
5764 ["DownRightTeeVector"] = 10591,  
5765 ["LeftUpTeeVector"] = 10592,  
5766 ["LeftDownTeeVector"] = 10593,  
5767 ["lHar"] = 10594,  
5768 ["uHar"] = 10595,  
5769 ["rHar"] = 10596,  
5770 ["dHar"] = 10597,  
5771 ["luruhar"] = 10598,  
5772 ["ldrdhar"] = 10599,  
5773 ["ruluhar"] = 10600,  
5774 ["rdldhar"] = 10601,  
5775 ["lharul"] = 10602,  
5776 ["llhard"] = 10603,

5777 ["rharul"] = 10604,  
5778 ["lrhard"] = 10605,  
5779 ["UpEquilibrium"] = 10606,  
5780 ["udhar"] = 10606,  
5781 ["ReverseUpEquilibrium"] = 10607,  
5782 ["duhar"] = 10607,  
5783 ["RoundImplies"] = 10608,  
5784 ["erarr"] = 10609,  
5785 ["simrarr"] = 10610,  
5786 ["larrsim"] = 10611,  
5787 ["rarrsim"] = 10612,  
5788 ["rarrap"] = 10613,  
5789 ["ltlarr"] = 10614,  
5790 ["gtrarr"] = 10616,  
5791 ["subrarr"] = 10617,  
5792 ["suplarr"] = 10619,  
5793 ["lfisht"] = 10620,  
5794 ["rfisht"] = 10621,  
5795 ["ufisht"] = 10622,  
5796 ["dfisht"] = 10623,  
5797 ["lopar"] = 10629,  
5798 ["ropar"] = 10630,  
5799 ["lbrke"] = 10635,  
5800 ["rbrke"] = 10636,  
5801 ["lbrkslu"] = 10637,  
5802 ["rbrksld"] = 10638,  
5803 ["lbrksld"] = 10639,  
5804 ["rbrkslu"] = 10640,  
5805 ["langd"] = 10641,  
5806 ["rangd"] = 10642,  
5807 ["lparlt"] = 10643,  
5808 ["rpargt"] = 10644,  
5809 ["gtlPar"] = 10645,  
5810 ["ltrPar"] = 10646,  
5811 ["vzigzag"] = 10650,  
5812 ["vangrt"] = 10652,  
5813 ["angrtvbd"] = 10653,  
5814 ["ange"] = 10660,  
5815 ["range"] = 10661,  
5816 ["dwangle"] = 10662,  
5817 ["uwangle"] = 10663,  
5818 ["angmsdaa"] = 10664,  
5819 ["angmsdab"] = 10665,  
5820 ["angmsdac"] = 10666,  
5821 ["angmsdad"] = 10667,  
5822 ["angmsdae"] = 10668,  
5823 ["angmsdaf"] = 10669,

5824 ["angmsdag"] = 10670,  
5825 ["angmsdah"] = 10671,  
5826 ["bemptyv"] = 10672,  
5827 ["demptyv"] = 10673,  
5828 ["cemptyv"] = 10674,  
5829 ["raemptyv"] = 10675,  
5830 ["laemptyv"] = 10676,  
5831 ["ohbar"] = 10677,  
5832 ["omid"] = 10678,  
5833 ["opar"] = 10679,  
5834 ["operp"] = 10681,  
5835 ["olcross"] = 10683,  
5836 ["odsold"] = 10684,  
5837 ["olcir"] = 10686,  
5838 ["ofcir"] = 10687,  
5839 ["olt"] = 10688,  
5840 ["ogt"] = 10689,  
5841 ["cirscir"] = 10690,  
5842 ["cirE"] = 10691,  
5843 ["solb"] = 10692,  
5844 ["bsolb"] = 10693,  
5845 ["boxbox"] = 10697,  
5846 ["trisb"] = 10701,  
5847 ["rtriltri"] = 10702,  
5848 ["LeftTriangleBar"] = 10703,  
5849 ["NotLeftTriangleBar"] = {10703, 824},  
5850 ["NotRightTriangleBar"] = {10704, 824},  
5851 ["RightTriangleBar"] = 10704,  
5852 ["iinfin"] = 10716,  
5853 ["infintie"] = 10717,  
5854 ["nvinfin"] = 10718,  
5855 ["eparsl"] = 10723,  
5856 ["smeparsl"] = 10724,  
5857 ["eqvparsl"] = 10725,  
5858 ["blacklozenge"] = 10731,  
5859 ["lozf"] = 10731,  
5860 ["RuleDelayed"] = 10740,  
5861 ["dsol"] = 10742,  
5862 ["bigodot"] = 10752,  
5863 ["xodot"] = 10752,  
5864 ["bigoplus"] = 10753,  
5865 ["xoplus"] = 10753,  
5866 ["bigotimes"] = 10754,  
5867 ["xotime"] = 10754,  
5868 ["biguplus"] = 10756,  
5869 ["xuplus"] = 10756,  
5870 ["bigscup"] = 10758,

5871 ["xscup"] = 10758,  
5872 ["iiiint"] = 10764,  
5873 ["qint"] = 10764,  
5874 ["fpartint"] = 10765,  
5875 ["cirfnint"] = 10768,  
5876 ["awint"] = 10769,  
5877 ["rppolint"] = 10770,  
5878 ["scpolint"] = 10771,  
5879 ["npolint"] = 10772,  
5880 ["pointint"] = 10773,  
5881 ["quatint"] = 10774,  
5882 ["intlarhk"] = 10775,  
5883 ["pluscir"] = 10786,  
5884 ["plusacir"] = 10787,  
5885 ["simplus"] = 10788,  
5886 ["plusdu"] = 10789,  
5887 ["plussim"] = 10790,  
5888 ["plustwo"] = 10791,  
5889 ["mcomma"] = 10793,  
5890 ["minusdu"] = 10794,  
5891 ["loplus"] = 10797,  
5892 ["roplus"] = 10798,  
5893 ["Cross"] = 10799,  
5894 ["timesd"] = 10800,  
5895 ["timesbar"] = 10801,  
5896 ["smashp"] = 10803,  
5897 ["lotimes"] = 10804,  
5898 ["rotimes"] = 10805,  
5899 ["otimesas"] = 10806,  
5900 ["Otimes"] = 10807,  
5901 ["odiv"] = 10808,  
5902 ["triplus"] = 10809,  
5903 ["triminus"] = 10810,  
5904 ["tritime"] = 10811,  
5905 ["intprod"] = 10812,  
5906 ["iproduct"] = 10812,  
5907 ["amalg"] = 10815,  
5908 ["capdot"] = 10816,  
5909 ["ncup"] = 10818,  
5910 ["ncap"] = 10819,  
5911 ["capand"] = 10820,  
5912 ["cupor"] = 10821,  
5913 ["cupcap"] = 10822,  
5914 ["capcup"] = 10823,  
5915 ["cupbrcap"] = 10824,  
5916 ["capbrcup"] = 10825,  
5917 ["cupcup"] = 10826,

5918 ["capcap"] = 10827,  
5919 ["ccups"] = 10828,  
5920 ["ccaps"] = 10829,  
5921 ["ccupssm"] = 10832,  
5922 ["And"] = 10835,  
5923 ["Or"] = 10836,  
5924 ["andand"] = 10837,  
5925 ["oror"] = 10838,  
5926 ["orslope"] = 10839,  
5927 ["andslope"] = 10840,  
5928 ["andv"] = 10842,  
5929 ["orv"] = 10843,  
5930 ["andd"] = 10844,  
5931 ["ord"] = 10845,  
5932 ["wedbar"] = 10847,  
5933 ["sdote"] = 10854,  
5934 ["simdot"] = 10858,  
5935 ["congdot"] = 10861,  
5936 ["ncongdot"] = {10861, 824},  
5937 ["easter"] = 10862,  
5938 ["apacir"] = 10863,  
5939 ["apE"] = 10864,  
5940 ["napE"] = {10864, 824},  
5941 ["eplus"] = 10865,  
5942 ["pluse"] = 10866,  
5943 ["Esim"] = 10867,  
5944 ["Colone"] = 10868,  
5945 ["Equal"] = 10869,  
5946 ["ddotseq"] = 10871,  
5947 ["eDDot"] = 10871,  
5948 ["equivDD"] = 10872,  
5949 ["ltcir"] = 10873,  
5950 ["gtcir"] = 10874,  
5951 ["ltquest"] = 10875,  
5952 ["gtquest"] = 10876,  
5953 ["LessSlantEqual"] = 10877,  
5954 ["NotLessSlantEqual"] = {10877, 824},  
5955 ["leqslant"] = 10877,  
5956 ["les"] = 10877,  
5957 ["nleqslant"] = {10877, 824},  
5958 ["nles"] = {10877, 824},  
5959 ["GreaterSlantEqual"] = 10878,  
5960 ["NotGreaterSlantEqual"] = {10878, 824},  
5961 ["geqslant"] = 10878,  
5962 ["ges"] = 10878,  
5963 ["ngeqslant"] = {10878, 824},  
5964 ["nges"] = {10878, 824},



5965 ["lesdot"] = 10879,  
5966 ["gesdot"] = 10880,  
5967 ["lesdoto"] = 10881,  
5968 ["gesdoto"] = 10882,  
5969 ["lesdotor"] = 10883,  
5970 ["gesdoto1"] = 10884,  
5971 ["lap"] = 10885,  
5972 ["lessapprox"] = 10885,  
5973 ["gap"] = 10886,  
5974 ["gtrapprox"] = 10886,  
5975 ["lne"] = 10887,  
5976 ["lneq"] = 10887,  
5977 ["gne"] = 10888,  
5978 ["gneq"] = 10888,  
5979 ["lnap"] = 10889,  
5980 ["lnapprox"] = 10889,  
5981 ["gnap"] = 10890,  
5982 ["gnapprox"] = 10890,  
5983 ["lEg"] = 10891,  
5984 ["lesseqqgtr"] = 10891,  
5985 ["gEl"] = 10892,  
5986 ["gtreqqless"] = 10892,  
5987 ["lsime"] = 10893,  
5988 ["gsime"] = 10894,  
5989 ["lsimg"] = 10895,  
5990 ["gsiml"] = 10896,  
5991 ["lgE"] = 10897,  
5992 ["glE"] = 10898,  
5993 ["lesges"] = 10899,  
5994 ["gesles"] = 10900,  
5995 ["els"] = 10901,  
5996 ["eqslantless"] = 10901,  
5997 ["egs"] = 10902,  
5998 ["eqslantgtr"] = 10902,  
5999 ["elsdot"] = 10903,  
6000 ["egsdot"] = 10904,  
6001 ["el"] = 10905,  
6002 ["eg"] = 10906,  
6003 ["siml"] = 10909,  
6004 ["simg"] = 10910,  
6005 ["simlE"] = 10911,  
6006 ["simgE"] = 10912,  
6007 ["LessLess"] = 10913,  
6008 ["NotNestedLessLess"] = {10913, 824},  
6009 ["GreaterGreater"] = 10914,  
6010 ["NotNestedGreaterGreater"] = {10914, 824},  
6011 ["glj"] = 10916,

6012 ["gla"] = 10917,  
6013 ["ltcc"] = 10918,  
6014 ["gtcc"] = 10919,  
6015 ["lescc"] = 10920,  
6016 ["gescc"] = 10921,  
6017 ["smt"] = 10922,  
6018 ["lat"] = 10923,  
6019 ["smte"] = 10924,  
6020 ["smtes"] = {10924, 65024},  
6021 ["late"] = 10925,  
6022 ["lates"] = {10925, 65024},  
6023 ["bumpE"] = 10926,  
6024 ["NotPrecedesEqual"] = {10927, 824},  
6025 ["PrecedesEqual"] = 10927,  
6026 ["npre"] = {10927, 824},  
6027 ["npreceq"] = {10927, 824},  
6028 ["pre"] = 10927,  
6029 ["preceq"] = 10927,  
6030 ["NotSucceedsEqual"] = {10928, 824},  
6031 ["SucceedsEqual"] = 10928,  
6032 ["nsce"] = {10928, 824},  
6033 ["nsucceq"] = {10928, 824},  
6034 ["sce"] = 10928,  
6035 ["succeq"] = 10928,  
6036 ["prE"] = 10931,  
6037 ["scE"] = 10932,  
6038 ["precneqq"] = 10933,  
6039 ["prnE"] = 10933,  
6040 ["scnE"] = 10934,  
6041 ["succneqq"] = 10934,  
6042 ["prap"] = 10935,  
6043 ["precapprox"] = 10935,  
6044 ["scap"] = 10936,  
6045 ["succapprox"] = 10936,  
6046 ["precnapprox"] = 10937,  
6047 ["prnap"] = 10937,  
6048 ["scnap"] = 10938,  
6049 ["succnapprox"] = 10938,  
6050 ["Pr"] = 10939,  
6051 ["Sc"] = 10940,  
6052 ["subdot"] = 10941,  
6053 ["supdot"] = 10942,  
6054 ["subplus"] = 10943,  
6055 ["supplus"] = 10944,  
6056 ["submult"] = 10945,  
6057 ["supmult"] = 10946,  
6058 ["subedot"] = 10947,

6059 ["supedot"] = 10948,  
6060 ["nsubE"] = {10949, 824},  
6061 ["nsubseteqq"] = {10949, 824},  
6062 ["subE"] = 10949,  
6063 ["subseteqq"] = 10949,  
6064 ["nsupE"] = {10950, 824},  
6065 ["nsupseteqq"] = {10950, 824},  
6066 ["supE"] = 10950,  
6067 ["supseteqq"] = 10950,  
6068 ["subsim"] = 10951,  
6069 ["supsim"] = 10952,  
6070 ["subnE"] = 10955,  
6071 ["subsetneqq"] = 10955,  
6072 ["varsubsetneqq"] = {10955, 65024},  
6073 ["vsubnE"] = {10955, 65024},  
6074 ["supnE"] = 10956,  
6075 ["supsetneqq"] = 10956,  
6076 ["varsupsetneqq"] = {10956, 65024},  
6077 ["vsupnE"] = {10956, 65024},  
6078 ["csub"] = 10959,  
6079 ["csup"] = 10960,  
6080 ["csube"] = 10961,  
6081 ["csupe"] = 10962,  
6082 ["subsup"] = 10963,  
6083 ["supsub"] = 10964,  
6084 ["subsub"] = 10965,  
6085 ["supsup"] = 10966,  
6086 ["suphsub"] = 10967,  
6087 ["supdsub"] = 10968,  
6088 ["forkv"] = 10969,  
6089 ["topfork"] = 10970,  
6090 ["mlcp"] = 10971,  
6091 ["Dashv"] = 10980,  
6092 ["DoubleLeftTee"] = 10980,  
6093 ["Vdashl"] = 10982,  
6094 ["Barv"] = 10983,  
6095 ["vBar"] = 10984,  
6096 ["vBarv"] = 10985,  
6097 ["Vbar"] = 10987,  
6098 ["Not"] = 10988,  
6099 ["bNot"] = 10989,  
6100 ["rnmid"] = 10990,  
6101 ["cirmid"] = 10991,  
6102 ["midcir"] = 10992,  
6103 ["topcir"] = 10993,  
6104 ["nhpar"] = 10994,  
6105 ["parsim"] = 10995,

6106 ["npars1"] = {11005, 8421},  
6107 ["pars1"] = 11005,  
6108 ["fflig"] = 64256,  
6109 ["filig"] = 64257,  
6110 ["fllig"] = 64258,  
6111 ["ffilig"] = 64259,  
6112 ["ffllig"] = 64260,  
6113 ["Ascr"] = 119964,  
6114 ["Cscr"] = 119966,  
6115 ["Dscr"] = 119967,  
6116 ["Gscr"] = 119970,  
6117 ["Jscr"] = 119973,  
6118 ["Kscr"] = 119974,  
6119 ["Nscr"] = 119977,  
6120 ["Oscr"] = 119978,  
6121 ["Pscr"] = 119979,  
6122 ["Qscr"] = 119980,  
6123 ["Sscr"] = 119982,  
6124 ["Tscr"] = 119983,  
6125 ["Uscr"] = 119984,  
6126 ["Vscr"] = 119985,  
6127 ["Wscr"] = 119986,  
6128 ["Xscr"] = 119987,  
6129 ["Yscr"] = 119988,  
6130 ["Zscr"] = 119989,  
6131 ["ascr"] = 119990,  
6132 ["bscr"] = 119991,  
6133 ["cscr"] = 119992,  
6134 ["dscr"] = 119993,  
6135 ["fscr"] = 119995,  
6136 ["hscr"] = 119997,  
6137 ["iscr"] = 119998,  
6138 ["jscr"] = 119999,  
6139 ["kscr"] = 120000,  
6140 ["lscr"] = 120001,  
6141 ["mscr"] = 120002,  
6142 ["nscr"] = 120003,  
6143 ["pscr"] = 120005,  
6144 ["qscr"] = 120006,  
6145 ["rscr"] = 120007,  
6146 ["sscr"] = 120008,  
6147 ["tscr"] = 120009,  
6148 ["uscr"] = 120010,  
6149 ["vscr"] = 120011,  
6150 ["wscr"] = 120012,  
6151 ["xscr"] = 120013,  
6152 ["yscr"] = 120014,

6153 ["zscr"] = 120015,  
6154 ["Afr"] = 120068,  
6155 ["Bfr"] = 120069,  
6156 ["Dfr"] = 120071,  
6157 ["Efr"] = 120072,  
6158 ["Ffr"] = 120073,  
6159 ["Gfr"] = 120074,  
6160 ["Jfr"] = 120077,  
6161 ["Kfr"] = 120078,  
6162 ["Lfr"] = 120079,  
6163 ["Mfr"] = 120080,  
6164 ["Nfr"] = 120081,  
6165 ["Ofr"] = 120082,  
6166 ["Pfr"] = 120083,  
6167 ["Qfr"] = 120084,  
6168 ["Sfr"] = 120086,  
6169 ["Tfr"] = 120087,  
6170 ["Ufr"] = 120088,  
6171 ["Vfr"] = 120089,  
6172 ["Wfr"] = 120090,  
6173 ["Xfr"] = 120091,  
6174 ["Yfr"] = 120092,  
6175 ["afr"] = 120094,  
6176 ["bfr"] = 120095,  
6177 ["cfr"] = 120096,  
6178 ["dfr"] = 120097,  
6179 ["efr"] = 120098,  
6180 ["ffr"] = 120099,  
6181 ["gfr"] = 120100,  
6182 ["hfr"] = 120101,  
6183 ["ifr"] = 120102,  
6184 ["jfr"] = 120103,  
6185 ["kfr"] = 120104,  
6186 ["lfr"] = 120105,  
6187 ["mfr"] = 120106,  
6188 ["nfr"] = 120107,  
6189 ["ofr"] = 120108,  
6190 ["pfr"] = 120109,  
6191 ["qfr"] = 120110,  
6192 ["rfr"] = 120111,  
6193 ["sfr"] = 120112,  
6194 ["tfr"] = 120113,  
6195 ["ufr"] = 120114,  
6196 ["vfr"] = 120115,  
6197 ["wfr"] = 120116,  
6198 ["xfr"] = 120117,  
6199 ["yfr"] = 120118,

```
6200 ["zfr"] = 120119,  
6201 ["Aopf"] = 120120,  
6202 ["Bopf"] = 120121,  
6203 ["Dopf"] = 120123,  
6204 ["Eopf"] = 120124,  
6205 ["Fopf"] = 120125,  
6206 ["Gopf"] = 120126,  
6207 ["Iopf"] = 120128,  
6208 ["Jopf"] = 120129,  
6209 ["Kopf"] = 120130,  
6210 ["Lopf"] = 120131,  
6211 ["Mopf"] = 120132,  
6212 ["Oopf"] = 120134,  
6213 ["Sopf"] = 120138,  
6214 ["Topf"] = 120139,  
6215 ["Uopf"] = 120140,  
6216 ["Vopf"] = 120141,  
6217 ["Wopf"] = 120142,  
6218 ["Xopf"] = 120143,  
6219 ["Yopf"] = 120144,  
6220 ["aopf"] = 120146,  
6221 ["bopf"] = 120147,  
6222 ["copf"] = 120148,  
6223 ["dopf"] = 120149,  
6224 ["eopf"] = 120150,  
6225 ["fopf"] = 120151,  
6226 ["gopf"] = 120152,  
6227 ["hopf"] = 120153,  
6228 ["iopf"] = 120154,  
6229 ["jopf"] = 120155,  
6230 ["kopf"] = 120156,  
6231 ["lopf"] = 120157,  
6232 ["mopf"] = 120158,  
6233 ["nopf"] = 120159,  
6234 ["oopf"] = 120160,  
6235 ["popf"] = 120161,  
6236 ["qopf"] = 120162,  
6237 ["ropf"] = 120163,  
6238 ["sopf"] = 120164,  
6239 ["topf"] = 120165,  
6240 ["uopf"] = 120166,  
6241 ["vopf"] = 120167,  
6242 ["wopf"] = 120168,  
6243 ["xopf"] = 120169,  
6244 ["yopf"] = 120170,  
6245 ["zopf"] = 120171,  
6246 }
```

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

```
6247 function entities.dec_entity(s)
6248   local n = tonumber(s)
6249   if n == nil then
6250     return "&#" .. s .. ";" -- fallback for unknown entities
6251   end
6252   return unicode.utf8.char(n)
6253 end
```

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

```
6254 function entities.hex_entity(s)
6255   local n = tonumber("0x"..s)
6256   if n == nil then
6257     return "&#x" .. s .. ";" -- fallback for unknown entities
6258   end
6259   return unicode.utf8.char(n)
6260 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.

```
6261 function entities.hex_entity_with_x_char(x, s)
6262   local n = tonumber("0x"..s)
6263   if n == nil then
6264     return "&#" .. x .. s .. ";" -- fallback for unknown entities
6265   end
6266   return unicode.utf8.char(n)
6267 end
```

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

```
6268 function entities.char_entity(s)
6269   local code_points = character_entities[s]
6270   if code_points == nil then
6271     return "&" .. s .. ";"
6272   end
6273   if type(code_points) ~= 'table' then
6274     code_points = {code_points}
6275   end
6276   local char_table = {}
6277   for _, code_point in ipairs(code_points) do
6278     table.insert(char_table, unicode.utf8.char(code_point))
6279   end
6280   return table.concat(char_table)
6281 end
```

### 3.1.3 Plain T<sub>E</sub>X Writer

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

```
6282 M.writer = {}
```

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

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

```
6283 function M.writer.new(options)
6284   local self = {}
```

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

```
6285   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.

```
6286   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.

```
6287   local slice_specifiers = {}
6288   for specifier in options.slice:gmatch("[^%s]+") do
6289     table.insert(slice_specifiers, specifier)
6290   end
6291
6292   if #slice_specifiers == 2 then
6293     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6294     local slice_begin_type = self.slice_begin:sub(1, 1)
6295     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
6296       self.slice_begin = "^" .. self.slice_begin
6297     end
6298     local slice_end_type = self.slice_end:sub(1, 1)
6299     if slice_end_type ~= "^" and slice_end_type ~= "$" then
```



```

6300     self.slice_end = "$" .. self.slice_end
6301   end
6302   elseif #slice_specifiers == 1 then
6303     self.slice_begin = "^" .. slice_specifiers[1]
6304     self.slice_end = "$" .. slice_specifiers[1]
6305   end
6306
6307   self.slice_begin_type = self.slice_begin:sub(1, 1)
6308   self.slice_begin_identifier = self.slice_begin:sub(2) or ""
6309   self.slice_end_type = self.slice_end:sub(1, 1)
6310   self.slice_end_identifier = self.slice_end:sub(2) or ""
6311
6312   if self.slice_begin == "^" and self.slice_end ~= "^" then
6313     self.is_writing = true
6314   else
6315     self.is_writing = false
6316   end

```

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

```
6317   self.space = " "
```

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

```
6318   self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
6319   function self.plain(s)
6320     return s
6321   end

```

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

```
6322   function self.paragraph(s)
6323     if not self.is_writing then return "" end
6324     return s
6325   end

```

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

```
6326   self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
6327   function self.interblocksep()
6328     if not self.is_writing then return "" end
6329     return self.interblocksep_text
6330   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.

```
6331   self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
```

```

6332 function self.paragraphsep()
6333     if not self.is_writing then return "" end
6334     return self.paragraphsep_text
6335 end

```

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

```

6336 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
6337 function self.undosep()
6338     if not self.is_writing then return "" end
6339     return self.undosep_text
6340 end

```

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

```

6341 self.soft_line_break = function()
6342     if self.flatten_inlines then return "\n" end
6343     return "\\markdownRendererSoftLineBreak\n{"
6344 end

```

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

```

6345 self.hard_line_break = function()
6346     if self.flatten_inlines then return "\n" end
6347     return "\\markdownRendererHardLineBreak\n{"
6348 end

```

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

```

6349 self.ellipsis = "\\markdownRendererEllipsis{"

```

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

```

6350 function self.thematic_break()
6351     if not self.is_writing then return "" end
6352     return "\\markdownRendererThematicBreak{"
6353 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.

```

6354 self.escaped_uri_chars = {
6355     [{""] = "\\markdownRendererLeftBrace{"},
6356     ["}"] = "\\markdownRendererRightBrace{"},
6357     [{"\\"}] = "\\markdownRendererBackslash{"},
6358     [{"r"}] = " ",
6359     [{"n"}] = " ",
6360 }
6361 self.escaped_minimal_strings = {
6362     [{"^"}] = "\\markdownRendererCircumflex",
6363     [{"^"}] = "\\markdownRendererCircumflex ",
6364     [{"☒"}] = "\\markdownRendererTickedBox{"},
6365     [{"◻"}] = "\\markdownRendererHalfTickedBox{"},

```

```

6366     ["□"] = "\\markdownRendererUntickedBox{}",
6367     [entities.hex_entity('FFFD')]
6368     = "\\markdownRendererReplacementCharacter{}",
6369   }

```

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

```

6370   self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6371   self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp

```

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

```

6372   self.escaped_chars = {
6373     [{""] = "\\markdownRendererLeftBrace{}",
6374     ["}"] = "\\markdownRendererRightBrace{}",
6375     ["%"] = "\\markdownRendererPercentSign{}",
6376     ["\\"] = "\\markdownRendererBackslash{}",
6377     ["#"] = "\\markdownRendererHash{}",
6378     ["$"] = "\\markdownRendererDollarSign{}",
6379     ["&"] = "\\markdownRendererAmpersand{}",
6380     ["_"] = "\\markdownRendererUnderscore{}",
6381     ["^"] = "\\markdownRendererCircumflex{}",
6382     ["~"] = "\\markdownRendererTilde{}",
6383     ["|"] = "\\markdownRendererPipe{}",
6384     [entities.hex_entity('0000')]
6385     = "\\markdownRendererReplacementCharacter{}",
6386   }

```

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

```

6387   local function create_escaper(char_escapes, string_escapes)
6388     local escape = util.escaper(char_escapes, string_escapes)
6389     return function(s)
6390       if self.flatten_inlines then return s end
6391       return escape(s)
6392     end
6393   end
6394   local escape_typographic_text = create_escaper(
6395     self.escaped_chars, self.escaped_strings)
6396   local escape_programmatic_text = create_escaper(
6397     self.escaped_uri_chars, self.escaped_minimal_strings)
6398   local escape_minimal = create_escaper(
6399     {}, 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.

```

6400 self.escape = escape_typographic_text
6401 self.math = escape_minimal
6402 if options.hybrid then
6403     self.identifier = escape_minimal
6404     self.string = escape_minimal
6405     self.uri = escape_minimal
6406     self.infostring = escape_minimal
6407 else
6408     self.identifier = escape_programmatic_text
6409     self.string = escape_typographic_text
6410     self.uri = escape_programmatic_text
6411     self.infostring = escape_programmatic_text
6412 end

```

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```

6413 function self.warning(t, m)
6414     return {"\markdownRendererWarning{" , self.escape(t), "}{" ,
6415           escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6416           escape_minimal(m or ""), "}"}
6417 end

```

Define `writer->error` as a function that will transform an input error text `t` with optional more error text `m` to the output format.

```

6418 function self.error(t, m)
6419     return {"\markdownRendererError{" , self.escape(t), "}{" ,
6420           escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6421           escape_minimal(m or ""), "}"}
6422 end

```

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

```

6423 function self.code(s, attributes)
6424     if self.flatten_inlines then return s end
6425     local buf = {}
6426     if attributes ~= nil then
6427         table.insert(buf,
6428             "\markdownRendererCodeSpanAttributeContextBegin\n")
6429         table.insert(buf, self.attributes(attributes))

```

```

6430     end
6431     table.insert(buf,
6432         {"\\markdownRendererCodeSpan{" , self.escape(s), "}")})
6433     if attributes ~= nil then
6434         table.insert(buf,
6435             "\\markdownRendererCodeSpanAttributeContextEnd{")
6436     end
6437     return buf
6438 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.

```

6439 function self.link(lab, src, tit, attributes)
6440     if self.flatten_inlines then return lab end
6441     local buf = {}
6442     if attributes ~= nil then
6443         table.insert(buf,
6444             "\\markdownRendererLinkAttributeContextBegin\n")
6445         table.insert(buf, self.attributes(attributes))
6446     end
6447     table.insert(buf, {"\\markdownRendererLink{" ,lab,"} ,
6448         {" ,self.escape(src),"} ,
6449         {" ,self.uri(src),"} ,
6450         {" ,self.string(tit or ""),"} })
6451     if attributes ~= nil then
6452         table.insert(buf,
6453             "\\markdownRendererLinkAttributeContextEnd{")
6454     end
6455     return buf
6456 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.

```

6457 function self.image(lab, src, tit, attributes)
6458     if self.flatten_inlines then return lab end
6459     local buf = {}
6460     if attributes ~= nil then
6461         table.insert(buf,
6462             "\\markdownRendererImageAttributeContextBegin\n")
6463         table.insert(buf, self.attributes(attributes))
6464     end
6465     table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ,
6466         {" ,self.string(src),"} ,
6467         {" ,self.uri(src),"} ,
6468         {" ,self.string(tit or ""),"} })

```

```

6469     if attributes ~= nil then
6470         table.insert(buf,
6471             "\\markdownRendererImageAttributeContextEnd{ }")
6472     end
6473     return buf
6474 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.

```

6475 function self.bulletlist(items,tight)
6476     if not self.is_writing then return "" end
6477     local buffer = {}
6478     for _,item in ipairs(items) do
6479         if item ~= "" then
6480             buffer[#buffer + 1] = self.bulletitem(item)
6481         end
6482     end
6483     local contents = util.intersperse(buffer,"\n")
6484     if tight and options.tightLists then
6485         return {"\\markdownRendererUlBeginTight\n",contents,
6486             "\n\\markdownRendererUlEndTight "}
6487     else
6488         return {"\\markdownRendererUlBegin\n",contents,
6489             "\n\\markdownRendererUlEnd "}
6490     end
6491 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.

```

6492 function self.bulletitem(s)
6493     return {"\\markdownRendererUlItem ",s,
6494         "\\markdownRendererUlItemEnd "}
6495 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.

```

6496 function self.orderedlist(items,tight,startnum)
6497     if not self.is_writing then return "" end
6498     local buffer = {}
6499     local num = startnum
6500     for _,item in ipairs(items) do
6501         if item ~= "" then
6502             buffer[#buffer + 1] = self.ordereditem(item,num)
6503         end
6504         if num ~= nil and item ~= "" then

```

```

6505         num = num + 1
6506     end
6507 end
6508 local contents = util.intersperse(buffer,"\n")
6509 if tight and options.tightLists then
6510     return {"\\markdownRenderer01BeginTight\n",contents,
6511           "\\n\\markdownRenderer01EndTight "}
6512 else
6513     return {"\\markdownRenderer01Begin\n",contents,
6514           "\\n\\markdownRenderer01End "}
6515 end
6516 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.

```

6517 function self.ordereditem(s,num)
6518     if num ~= nil then
6519         return {"\\markdownRenderer01ItemWithNumber{" ,num,"}",s,
6520               "\\markdownRenderer01ItemEnd "}
6521     else
6522         return {"\\markdownRenderer01Item ",s,
6523               "\\markdownRenderer01ItemEnd "}
6524     end
6525 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.

```

6526 function self.inline_html_comment(contents)
6527     if self.flatten_inlines then return contents end
6528     return {"\\markdownRendererInlineHtmlComment{" ,contents,""}
6529 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.

```

6530 function self.inline_html_tag(contents)
6531     if self.flatten_inlines then return contents end
6532     return {"\\markdownRendererInlineHtmlTag{" ,
6533           self.string(contents),""}
6534 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.

```

6535 function self.block_html_element(s)
6536     if not self.is_writing then return "" end

```

```

6537     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6538     return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
6539 end

```

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

```

6540 function self.emphasis(s)
6541     if self.flatten_inlines then return s end
6542     return {"\\markdownRendererEmphasis{" ,s,"}"}
6543 end

```

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

```

6544 function self.checkbox(f)
6545     if f == 1.0 then
6546         return "☒ "
6547     elseif f == 0.0 then
6548         return "☐ "
6549     else
6550         return "◻ "
6551     end
6552 end

```

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

```

6553 function self.strong(s)
6554     if self.flatten_inlines then return s end
6555     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
6556 end

```

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

```

6557 function self.blockquote(s)
6558     if not self.is_writing then return "" end
6559     return {"\\markdownRendererBlockQuoteBegin\n",s,
6560         "\\markdownRendererBlockQuoteEnd "}
6561 end

```

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

```

6562 function self.verbatim(s)
6563     if not self.is_writing then return "" end
6564     s = s:gsub("\n$", "")
6565     local name = util.cache_verbatim(options.cacheDir, s)
6566     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
6567 end

```

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



```

6568 function self.document(d)
6569     local buf = {"\\markdownRendererDocumentBegin\n"}
6570
6571     -- warn against the `hybrid` option
6572     if options.hybrid then
6573         local text = "The `hybrid` option has been soft-deprecated."
6574         local more = "Consider using one of the following better options "
6575             .. "for mixing TeX and markdown: `contentBlocks`, "
6576             .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6577             .. "`texMathSingleBackslash`, and "
6578             .. "`texMathDoubleBackslash`. "
6579             .. "For more information, see the user manual at "
6580             .. "<https://witiko.github.io/markdown/>."
6581         table.insert(buf, self.warning(text, more))
6582     end
6583
6584     -- insert the text of the document
6585     table.insert(buf, d)
6586
6587     -- pop all attributes
6588     table.insert(buf, self.pop_attributes())
6589
6590     table.insert(buf, "\\markdownRendererDocumentEnd")
6591
6592     return buf
6593 end

```

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

```

6594 local seen_identifiers = {}
6595 local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
6596 local function normalize_attributes(attributes, auto_identifiers)
6597     -- normalize attributes
6598     local normalized_attributes = {}
6599     local has_explicit_identifiers = false
6600     local key, value
6601     for _, attribute in ipairs(attributes or {}) do
6602         if attribute:sub(1, 1) == "#" then
6603             table.insert(normalized_attributes, attribute)
6604             has_explicit_identifiers = true
6605             seen_identifiers[attribute:sub(2)] = true
6606         elseif attribute:sub(1, 1) == "." then
6607             table.insert(normalized_attributes, attribute)
6608         else
6609             key, value = attribute:match(key_value_regex)
6610             if key:lower() == "id" then
6611                 table.insert(normalized_attributes, "#" .. value)

```

```

6612     elseif key:lower() == "class" then
6613         local classes = {}
6614         for class in value:gmatch("%S+") do
6615             table.insert(classes, class)
6616         end
6617         table.sort(classes)
6618         for _, class in ipairs(classes) do
6619             table.insert(normalized_attributes, "." .. class)
6620         end
6621     else
6622         table.insert(normalized_attributes, attribute)
6623     end
6624 end
6625 end
6626
6627 -- if no explicit identifiers exist, add auto identifiers
6628 if not has_explicit_identifiers and auto_identifiers ~= nil then
6629     local seen_auto_identifiers = {}
6630     for _, auto_identifier in ipairs(auto_identifiers) do
6631         if seen_auto_identifiers[auto_identifier] == nil then
6632             seen_auto_identifiers[auto_identifier] = true
6633             if seen_identifiers[auto_identifier] == nil then
6634                 seen_identifiers[auto_identifier] = true
6635                 table.insert(normalized_attributes,
6636                     "#" .. auto_identifier)
6637             else
6638                 local auto_identifier_number = 1
6639                 while true do
6640                     local numbered_auto_identifier = auto_identifier .. "-"
6641   .. auto_identifier_number
6642                     if seen_identifiers[numbered_auto_identifier] == nil then
6643                         seen_identifiers[numbered_auto_identifier] = true
6644                         table.insert(normalized_attributes,
6645                             "#" .. numbered_auto_identifier)
6646                     break
6647                 end
6648                 auto_identifier_number = auto_identifier_number + 1
6649             end
6650         end
6651     end
6652 end
6653 end
6654
6655 -- sort and deduplicate normalized attributes
6656 table.sort(normalized_attributes)
6657 local seen_normalized_attributes = {}
6658 local deduplicated_normalized_attributes = {}

```

```

6659     for _, attribute in ipairs(normalized_attributes) do
6660         if seen_normalized_attributes[attribute] == nil then
6661             seen_normalized_attributes[attribute] = true
6662             table.insert(deduplicated_normalized_attributes, attribute)
6663         end
6664     end
6665
6666     return deduplicated_normalized_attributes
6667 end
6668
6669 function self.attributes(attributes, should_normalize_attributes)
6670     local normalized_attributes
6671     if should_normalize_attributes == false then
6672         normalized_attributes = attributes
6673     else
6674         normalized_attributes = normalize_attributes(attributes)
6675     end
6676
6677     local buf = {}
6678     local key, value
6679     for _, attribute in ipairs(normalized_attributes) do
6680         if attribute:sub(1, 1) == "#" then
6681             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6682                 attribute:sub(2), "}")})
6683         elseif attribute:sub(1, 1) == "." then
6684             table.insert(buf, {"\\markdownRendererAttributeName{" ,
6685                 attribute:sub(2), "}")})
6686         else
6687             key, value = attribute:match(key_value_regex)
6688             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6689                 key, "{" , value, "}")})
6690         end
6691     end
6692
6693     return buf
6694 end

```

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

```

6695     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`.

```

6696     self.attribute_type_levels = {}
6697     setmetatable(self.attribute_type_levels,
6698         { __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`.

```

6699 local function apply_attributes()
6700   local buf = {}
6701   for i = 1, #self.active_attributes do
6702     local start_output = self.active_attributes[i][3]
6703     if start_output ~= nil then
6704       table.insert(buf, start_output)
6705     end
6706   end
6707   return buf
6708 end
6709
6710 local function tear_down_attributes()
6711   local buf = {}
6712   for i = #self.active_attributes, 1, -1 do
6713     local end_output = self.active_attributes[i][4]
6714     if end_output ~= nil then
6715       table.insert(buf, end_output)
6716     end
6717   end
6718   return buf
6719 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.

```

6720 function self.push_attributes(attribute_type, attributes,
6721                             start_output, end_output)
6722   local attribute_type_level
6723     = self.attribute_type_levels[attribute_type]
6724   self.attribute_type_levels[attribute_type]
6725     = attribute_type_level + 1
6726
6727   -- index attributes in a hash table for easy lookup
6728   attributes = attributes or {}
6729   for i = 1, #attributes do
6730     attributes[attributes[i]] = true
6731   end
6732
6733   local buf = {}
6734   -- handle slicing
6735   if attributes["#" .. self.slice_end_identifier] ~= nil and
6736     self.slice_end_type == "^" then

```

```

6737     if self.is_writing then
6738         table.insert(buf, self.undosep())
6739         table.insert(buf, tear_down_attributes())
6740     end
6741     self.is_writing = false
6742 end
6743 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6744     self.slice_begin_type == "^" then
6745     table.insert(buf, apply_attributes())
6746     self.is_writing = true
6747 end
6748 if self.is_writing and start_output ~= nil then
6749     table.insert(buf, start_output)
6750 end
6751 table.insert(self.active_attributes,
6752             {attribute_type, attributes,
6753              start_output, end_output})
6754 return buf
6755 end
6756

```

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`).

```

6757 function self.pop_attributes(attribute_type)
6758     local buf = {}
6759     -- pop attributes until we find attributes of correct type
6760     -- or until no attributes remain
6761     local current_attribute_type = false
6762     while current_attribute_type ~= attribute_type and
6763         #self.active_attributes > 0 do
6764         local attributes, _, end_output
6765         current_attribute_type, attributes, _, end_output = table.unpack(
6766             self.active_attributes[#self.active_attributes])
6767         local attribute_type_level
6768             = self.attribute_type_levels[current_attribute_type]
6769         self.attribute_type_levels[current_attribute_type]
6770             = attribute_type_level - 1
6771         if self.is_writing and end_output ~= nil then
6772             table.insert(buf, end_output)
6773         end
6774         table.remove(self.active_attributes, #self.active_attributes)
6775         -- handle slicing
6776         if attributes["#" .. self.slice_end_identifier] ~= nil

```

```

6777         and self.slice_end_type == "$" then
6778     if self.is_writing then
6779         table.insert(buf, self.undosep())
6780         table.insert(buf, tear_down_attributes())
6781     end
6782     self.is_writing = false
6783 end
6784 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6785     self.slice_begin_type == "$" then
6786     self.is_writing = true
6787     table.insert(buf, apply_attributes())
6788 end
6789 end
6790 return buf
6791 end

```

Create an auto identifier string by stripping and converting characters from string `s`.

```

6792 local function create_auto_identifier(s)
6793     local buffer = {}
6794     local prev_space = false
6795     local letter_found = false
6796     local normalized_s = s
6797     if not options.unicodeNormalization
6798         or options.unicodeNormalizationForm ~= "nfc" then
6799         normalized_s = uni_algos.normalize.NFC(normalized_s)
6800     end
6801
6802     for _, code in utf8.codes(normalized_s) do
6803         local char = utf8.char(code)
6804
6805         -- Remove everything up to the first letter.
6806         if not letter_found then
6807             local is_letter = unicode.utf8.match(char, "%a")
6808             if is_letter then
6809                 letter_found = true
6810             else
6811                 goto continue
6812             end
6813         end
6814
6815         -- Remove all non-alphanumeric characters, except underscores,
6816         -- hyphens, and periods.
6817         if not unicode.utf8.match(char, "[%w_-%.%s]") then
6818             goto continue
6819         end
6820
6821         -- Replace all spaces and newlines with hyphens.
6822         if unicode.utf8.match(char, "[%s\n]") then

```

```

6823     char = "-"
6824     if prev_space then
6825         goto continue
6826     else
6827         prev_space = true
6828     end
6829 else
6830     -- Convert all alphabetic characters to lowercase.
6831     char = unicode.utf8.lower(char)
6832     prev_space = false
6833 end
6834
6835 table.insert(buffer, char)
6836
6837 ::continue::
6838 end
6839
6840 if prev_space then
6841     table.remove(buffer)
6842 end
6843
6844 local identifier = #buffer == 0 and "section"
6845                  or table.concat(buffer, "")
6846 return identifier
6847 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

6848 local function create_gfm_auto_identifier(s)
6849     local buffer = {}
6850     local prev_space = false
6851     local letter_found = false
6852     local normalized_s = s
6853     if not options.unicodeNormalization
6854         or options.unicodeNormalizationForm ~= "nfc" then
6855         normalized_s = uni_algos.normalize.NFC(normalized_s)
6856     end
6857
6858     for _, code in utf8.codes(normalized_s) do
6859         local char = utf8.char(code)
6860
6861         -- Remove everything up to the first non-space.
6862         if not letter_found then
6863             local is_letter = unicode.utf8.match(char, "%S")
6864             if is_letter then
6865                 letter_found = true
6866             else

```

```

6867         goto continue
6868     end
6869 end
6870
6871 -- Remove all non-alphanumeric characters, except underscores
6872 -- and hyphens.
6873 if not unicode.utf8.match(char, "[%w_-%s]") then
6874     prev_space = false
6875     goto continue
6876 end
6877
6878 -- Replace all spaces and newlines with hyphens.
6879 if unicode.utf8.match(char, "[%s\n]") then
6880     char = "-"
6881     if prev_space then
6882         goto continue
6883     else
6884         prev_space = true
6885     end
6886 else
6887     -- Convert all alphabetic characters to lowercase.
6888     char = unicode.utf8.lower(char)
6889     prev_space = false
6890 end
6891
6892 table.insert(buffer, char)
6893
6894 ::continue::
6895 end
6896
6897 if prev_space then
6898     table.remove(buffer)
6899 end
6900
6901 local identifier = #buffer == 0 and "section"
6902                 or table.concat(buffer, "")
6903 return identifier
6904 end

```

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

```

6905 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6906 self.secend_text = "\n\\markdownRendererSectionEnd "
6907 function self.heading(s, level, attributes)
6908     local buf = {}
6909     local flat_text, inlines = table.unpack(s)
6910

```



```

6911 -- push empty attributes for implied sections
6912 while self.attribute_type_levels["heading"] < level - 1 do
6913     table.insert(buf,
6914                 self.push_attributes("heading",
6915                                     nil,
6916                                     self.secbegin_text,
6917                                     self.secend_text))
6918 end
6919
6920 -- pop attributes for sections that have ended
6921 while self.attribute_type_levels["heading"] >= level do
6922     table.insert(buf, self.pop_attributes("heading"))
6923 end
6924
6925 -- construct attributes for the new section
6926 local auto_identifiers = {}
6927 if self.options.autoIdentifiers then
6928     table.insert(auto_identifiers, create_auto_identifier(flat_text))
6929 end
6930 if self.options.gfmAutoIdentifiers then
6931     table.insert(auto_identifiers,
6932                 create_gfm_auto_identifier(flat_text))
6933 end
6934 local normalized_attributes = normalize_attributes(attributes,
6935  auto_identifiers)
6936
6937 -- push attributes for the new section
6938 local start_output = {}
6939 local end_output = {}
6940 table.insert(start_output, self.secbegin_text)
6941 table.insert(end_output, self.secend_text)
6942
6943 table.insert(buf, self.push_attributes("heading",
6944                                     normalized_attributes,
6945                                     start_output,
6946                                     end_output))
6947 assert(self.attribute_type_levels["heading"] == level)
6948
6949 -- render the heading and its attributes
6950 if self.is_writing and #normalized_attributes > 0 then
6951     table.insert(buf,
6952                 "\\markdownRendererHeaderAttributeContextBegin\n")
6953     table.insert(buf, self.attributes(normalized_attributes, false))
6954 end
6955
6956 local cmd
6957 level = level + options.shiftHeadings

```

```

6958     if level <= 1 then
6959         cmd = "\\markdownRendererHeadingOne"
6960     elseif level == 2 then
6961         cmd = "\\markdownRendererHeadingTwo"
6962     elseif level == 3 then
6963         cmd = "\\markdownRendererHeadingThree"
6964     elseif level == 4 then
6965         cmd = "\\markdownRendererHeadingFour"
6966     elseif level == 5 then
6967         cmd = "\\markdownRendererHeadingFive"
6968     elseif level >= 6 then
6969         cmd = "\\markdownRendererHeadingSix"
6970     else
6971         cmd = ""
6972     end
6973     if self.is_writing then
6974         table.insert(buf, {cmd, "{", inlines, "}"})
6975     end
6976
6977     if self.is_writing and #normalized_attributes > 0 then
6978         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6979     end
6980
6981     return buf
6982 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.

```

6983     function self.get_state()
6984         return {
6985             is_writing=self.is_writing,
6986             flatten_inlines=self.flatten_inlines,
6987             active_attributes={table.unpack(self.active_attributes)},
6988         }
6989     end

```

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

```

6990     function self.set_state(s)
6991         local previous_state = self.get_state()
6992         for key, value in pairs(s) do
6993             self[key] = value
6994         end
6995         return previous_state
6996     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`.

```
6997 function self.defer_call(f)
6998     local previous_state = self.get_state()
6999     return function(...)
7000         local state = self.set_state(previous_state)
7001         local return_value = f(...)
7002         self.set_state(state)
7003         return return_value
7004     end
7005 end
7006
7007 return self
7008 end
```

### 3.1.4 Parsers

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

```
7009 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
7010 parsers.percent = P("%")
7011 parsers.at = P("@")
7012 parsers.comma = P(",")
7013 parsers.asterisk = P("*")
7014 parsers.dash = P("-")
7015 parsers.plus = P("+")
7016 parsers.underscore = P("_")
7017 parsers.period = P(".")
7018 parsers.hash = P("#")
7019 parsers.dollar = P("$")
7020 parsers.ampersand = P("&")
7021 parsers.backtick = P("`")
7022 parsers.less = P("<")
7023 parsers.more = P(">")
7024 parsers.space = P(" ")
7025 parsers.squote = P("'")
7026 parsers.dquote = P('"')
7027 parsers.lparent = P("(")
7028 parsers.rparent = P(")")
7029 parsers.lbracket = P("[")
7030 parsers.rbracket = P("]")
7031 parsers.lbrace = P("{")
```

```

7032 parsers.rbrace           = P("}")
7033 parsers.circumflex       = P("^")
7034 parsers.slash            = P("/")
7035 parsers.equal            = P("=")
7036 parsers.colon            = P(":")
7037 parsers.semicolon        = P(";")
7038 parsers.exclamation      = P("!")
7039 parsers.pipe             = P("|")
7040 parsers.tilde            = P("~")
7041 parsers.backslash        = P("\\")
7042 parsers.tab              = P("\t")
7043 parsers.newline          = P("\n")
7044
7045 parsers.digit            = R("09")
7046 parsers.hexdigit         = R("09","af","AF")
7047 parsers.letter           = R("AZ","az")
7048 parsers.alphanumeric     = R("AZ","az","09")
7049 parsers.keyword          = parsers.letter
7050                          * (parsers.alphanumeric + parsers.dash)^0
7051
7052 parsers.doubleasterisks   = P("**")
7053 parsers.doubleunderscores = P("__")
7054 parsers.doubletildes     = P("~~")
7055 parsers.fourspace        = P("    ")
7056
7057 parsers.any              = P(1)
7058 parsers.succeed          = P(true)
7059 parsers.fail             = P(false)
7060
7061 parsers.internal_punctuation = S(";, .?")
7062 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
7063

```

### 3.1.5 Unicode punctuation

This section documents the Unicode punctuation<sup>33</sup> 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))

All code from this section will be executed during the compilation of the Markdown package and the standard output will be stored in a file named `markdown-unicode-data.lua` with the precompiled parser of Unicode punctuation.

<sup>33</sup>See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

7064 ;(function()
7065   local pathname = assert(kpse.find_file("UnicodeData.txt"),
7066     [[Could not locate file "UnicodeData.txt"]])
7067   local file = assert(io.open(pathname, "r"),
7068     [[Could not open file "UnicodeData.txt"]])

```

In order to minimize the size and speed of the parser, we will first construct a prefix tree of UTF-8 encodings for all codepoints of a given code length.

```

7069   local prefix_trees = {}
7070   for line in file:lines() do
7071     local codepoint, major_category = line:match("^(%x+);[~;]*;(%)a")
7072     if major_category == "P" or major_category == "S" then
7073       local code = unicode.utf8.char(tonumber(codepoint, 16))
7074       if prefix_trees[#code] == nil then
7075         prefix_trees[#code] = {}
7076       end
7077       local node = prefix_trees[#code]
7078       for i = 1, #code do
7079         local byte = code:sub(i, i)
7080         if i < #code then
7081           if node[byte] == nil then
7082             node[byte] = {}
7083           end
7084           node = node[byte]
7085         else
7086           table.insert(node, byte)
7087         end
7088       end
7089     end
7090   end
7091   assert(file:close())
7092

```

Next, we will construct a parser out of the prefix tree.

```

7093   local function depth_first_search(node, path, visit, leave)
7094     visit(node, path)
7095     for label, child in pairs(node) do
7096       if type(child) == "table" then
7097         depth_first_search(child, path .. label, visit, leave)
7098       else
7099         visit(child, path)
7100       end
7101     end
7102     leave(node, path)
7103   end
7104
7105   print("M.punctuation = {}")
7106   print("local S = lpeg.S")

```

```

7107 print("-- luacheck: push no max line length")
7108 for length, prefix_tree in pairs(prefix_trees) do
7109     local subparsers = {}
7110     depth_first_search(prefix_tree, "", function(node, path)
7111         if type(node) == "string" then
7112             local suffix
7113             if node == "]" then
7114                 suffix = "S('" .. node .. "'"")"
7115             else
7116                 suffix = "S([[ " .. node .. " ]])"
7117             end
7118             if subparsers[path] ~= nil then
7119                 subparsers[path] = subparsers[path] .. " + " .. suffix
7120             else
7121                 subparsers[path] = suffix
7122             end
7123         end
7124     end, function(_, path)
7125         if #path > 0 then
7126             local byte = path:sub(#path, #path)
7127             local parent_path = path:sub(1, #path-1)
7128             if subparsers[path] ~= nil then
7129                 local suffix
7130                 if byte == "]" then
7131                     suffix = "S('" .. byte .. "'"")"
7132                 else
7133                     suffix = "S([[ " .. byte .. " ]])"
7134                 end
7135                 suffix = suffix .. " * (" .. subparsers[path] .. ")"
7136                 if subparsers[parent_path] ~= nil then
7137                     subparsers[parent_path] = subparsers[parent_path]
7138                         .. " + " .. suffix
7139                 else
7140                     subparsers[parent_path] = suffix
7141                 end
7142             end
7143         else
7144             print("M.punctuation[" .. length .. "] = " .. subparsers[path])
7145         end
7146     end)
7147 end
7148 print("-- luacheck: pop")
7149 end)()
7150 print("return M")

```

Back in the Markdown package, we will load the precompiled parser of Unicode punctuation.

```

7151 local unicode_data = require("markdown-unicode-data")
7152 if metadata.version ~= unicode_data.metadata.version then
7153     util.warning(
7154         "markdown.lua " .. metadata.version .. " used with " ..
7155         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
7156     )
7157 end
7158 parsers.punctuation = unicode_data.punctuation
7159
7160 parsers.escapable           = parsers.ascii_punctuation
7161 parsers.anyescaped         = parsers.backslash / ""
7162                             * parsers.escapable
7163                             + parsers.any
7164
7165 parsers.spacechar          = S("\t ")
7166 parsers.spacing            = S(" \n\r\t")
7167 parsers.nonspacechar       = parsers.any - parsers.spacing
7168 parsers.optionalspace     = parsers.spacechar^0
7169
7170 parsers.normalchar        = parsers.any - (V("SpecialChar")
7171   + parsers.spacing)
7172 parsers.eof                = -parsers.any
7173 parsers.nonindentspace    = parsers.space^-3 * - parsers.spacechar
7174 parsers.indent             = parsers.space^-3 * parsers.tab
7175                             + parsers.fourspaces / ""
7176 parsers.linechar          = P(1 - parsers.newline)
7177
7178 parsers.blankline         = parsers.optionalspace
7179                             * parsers.newline / "\n"
7180 parsers.blanklines        = parsers.blankline^0
7181 parsers.skipblanklines    = ( parsers.optionalspace
7182                             * parsers.newline)^0
7183 parsers.indentedline      = parsers.indent / ""
7184                             * C( parsers.linechar^1
7185                                 * parsers.newline^-1)
7186 parsers.optionallyindentedline = parsers.indent^-1 / ""
7187                             * C( parsers.linechar^1
7188                                 * parsers.newline^-1)
7189 parsers.sp                 = parsers.spacing^0
7190 parsers.spnl              = parsers.optionalspace
7191                             * ( parsers.newline
7192                                 * parsers.optionalspace)^-1
7193 parsers.line               = parsers.linechar^0 * parsers.newline
7194 parsers.nonemptyline      = parsers.line - parsers.blankline

```

### 3.1.5.1 Parsers Used for Indentation

```
7195
7196 parsers.leader      = parsers.space^-3
7197
```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```
7198 local function has_trail(indent_table)
7199     return indent_table ~= nil and
7200         indent_table.trail ~= nil and
7201         next(indent_table.trail) ~= nil
7202 end
7203
```

Check if indent table `indent_table` has any indents.

```
7204 local function has_indents(indent_table)
7205     return indent_table ~= nil and
7206         indent_table.indents ~= nil and
7207         next(indent_table.indents) ~= nil
7208 end
7209
```

Add a trail `trail_info` to the indent table `indent_table`.

```
7210 local function add_trail(indent_table, trail_info)
7211     indent_table.trail = trail_info
7212     return indent_table
7213 end
7214
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
7215 local function remove_trail(indent_table)
7216     indent_table.trail = nil
7217     return indent_table
7218 end
7219
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
7220 local function update_indent_table(indent_table, new_indent, add)
7221     indent_table = remove_trail(indent_table)
7222
7223     if not has_indents(indent_table) then
7224         indent_table.indents = {}
7225     end
7226
7227
7228     if add then
7229         indent_table.indents[#indent_table.indents + 1] = new_indent
7230     else
7231         if indent_table.indents[#indent_table.indents].name
7232             == new_indent.name then
7233             indent_table.indents[#indent_table.indents] = nil
```



```

7234     end
7235 end
7236
7237 return indent_table
7238 end
7239

```

Remove an indent by its name `name`.

```

7240 local function remove_indent(name)
7241     local remove_indent_level =
7242         function(s, i, indent_table) -- luacheck: ignore s i
7243             indent_table = update_indent_table(indent_table, {name=name},
7244   false)
7245             return true, indent_table
7246         end
7247
7248     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
7249 end
7250

```

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`.

```

7251 local function process_starter_spacing(indent, spacing,
7252                                     minimum, left_strip_length)
7253     left_strip_length = left_strip_length or 0
7254
7255     local count = 0
7256     local tab_value = 4 - (indent) % 4
7257
7258     local code_started, minimum_found = false, false
7259     local code_start, minimum_remainder = "", ""
7260
7261     local left_total_stripped = 0
7262     local full_remainder = ""
7263
7264     if spacing ~= nil then
7265         for i = 1, #spacing do
7266             local character = spacing:sub(i, i)
7267
7268             if character == "\t" then
7269                 count = count + tab_value
7270                 tab_value = 4
7271             elseif character == " " then
7272                 count = count + 1

```

```

7273     tab_value = 4 - (1 - tab_value) % 4
7274 end
7275
7276 if (left_strip_length ~= 0) then
7277     local possible_to_strip = math.min(count, left_strip_length)
7278     count = count - possible_to_strip
7279     left_strip_length = left_strip_length - possible_to_strip
7280     left_total_stripped = left_total_stripped + possible_to_strip
7281 else
7282     full_remainder = full_remainder .. character
7283 end
7284
7285 if (minimum_found) then
7286     minimum_remainder = minimum_remainder .. character
7287 elseif (count >= minimum) then
7288     minimum_found = true
7289     minimum_remainder = minimum_remainder
7290         .. string.rep(" ", count - minimum)
7291 end
7292
7293 if (code_started) then
7294     code_start = code_start .. character
7295 elseif (count >= minimum + 4) then
7296     code_started = true
7297     code_start = code_start
7298         .. string.rep(" ", count - (minimum + 4))
7299 end
7300 end
7301 end
7302
7303 local remainder
7304 if (code_started) then
7305     remainder = code_start
7306 else
7307     remainder = string.rep(" ", count - minimum)
7308 end
7309
7310 local is_minimum = count >= minimum
7311 return {
7312     is_code = code_started,
7313     remainder = remainder,
7314     left_total_stripped = left_total_stripped,
7315     is_minimum = is_minimum,
7316     minimum_remainder = minimum_remainder,
7317     total_length = count,
7318     full_remainder = full_remainder
7319 }

```

```
7320 end
7321
```

Count the total width of all indents in the indent table `indent_table`.

```
7322 local function count_indent_tab_level(indent_table)
7323   local count = 0
7324   if not has_indents(indent_table) then
7325     return count
7326   end
7327
7328   for i=1, #indent_table.indents do
7329     count = count + indent_table.indents[i].length
7330   end
7331   return count
7332 end
7333
```

Count the total width of a delimiter `delimiter`.

```
7334 local function total_delimiter_length(delimiter)
7335   local count = 0
7336   if type(delimiter) == "string" then return #delimiter end
7337   for _, value in pairs(delimiter) do
7338     count = count + total_delimiter_length(value)
7339   end
7340   return count
7341 end
7342
```

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`.

```
7343 local function process_starter_indent(_, _, indent_table, starter,
7344                                       is_blank, indent_type, breakable)
7345   local last_trail = starter[1]
7346   local delimiter = starter[2]
7347   local raw_new_trail = starter[3]
7348
7349   if indent_type == "bq" and not breakable then
7350     indent_table.ignore_blockquote_blank = true
7351   end
7352
7353   if has_trail(indent_table) then
7354     local trail = indent_table.trail
7355     if trail.is_code then
7356       return false
7357     end
7358     last_trail = trail.remainder
7359   else
7360     local sp = process_starter_spacing(0, last_trail, 0, 0)

```

```

7361
7362     if sp.is_code then
7363         return false
7364     end
7365     last_trail = sp.remainder
7366 end
7367
7368 local preceding_indentation = count_indent_tab_level(indent_table) % 4
7369 local last_trail_length = #last_trail
7370 local delimiter_length = total_delimiter_length(delimiter)
7371
7372 local total_indent_level = preceding_indentation + last_trail_length
7373                        + delimiter_length
7374
7375 local sp = {}
7376 if not is_blank then
7377     sp = process_starter_spacing(total_indent_level, raw_new_trail,
7378                                0, 1)
7379 end
7380
7381 local del_trail_length = sp.left_total_stripped
7382 if is_blank then
7383     del_trail_length = 1
7384 elseif not sp.is_code then
7385     del_trail_length = del_trail_length + #sp.remainder
7386 end
7387
7388 local indent_length = last_trail_length + delimiter_length
7389                        + del_trail_length
7390 local new_indent_info = {name=indent_type, length=indent_length}
7391
7392 indent_table = update_indent_table(indent_table, new_indent_info,
7393                                    true)
7394 indent_table = add_trail(indent_table,
7395                          {is_code=sp.is_code,
7396                           remainder=sp.remainder,
7397                           total_length=sp.total_length,
7398                           full_remainder=sp.full_remainder})
7399
7400 return true, indent_table
7401 end
7402

```

Return the pattern corresponding with the indent name [name](#).

```

7403 local function decode_pattern(name)
7404     local delimiter = parsers.succeed
7405     if name == "bq" then
7406         delimiter = parsers.more

```

```

7407 end
7408
7409 return C(parsers.optionalspace) * C(delimiter)
7410       * C(parsers.optionalspace) * Cp()
7411 end
7412

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

7413 local function left_blank_starter(indent_table)
7414   local blank_starter_index
7415
7416   if not has_indents(indent_table) then
7417     return
7418   end
7419
7420   for i = #indent_table.indents,1,-1 do
7421     local value = indent_table.indents[i]
7422     if value.name == "li" then
7423       blank_starter_index = i
7424     else
7425       break
7426     end
7427   end
7428
7429   return blank_starter_index
7430 end
7431

```

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.

```

7432 local function traverse_indent(s, i, indent_table, is_optional,
7433                               is_blank, current_line_indents)
7434   local new_index = i
7435
7436   local preceding_indentation = 0
7437   local current_trail = {}
7438
7439   local blank_starter = left_blank_starter(indent_table)
7440
7441   if current_line_indents == nil then
7442     current_line_indents = {}
7443   end
7444

```

```

7445 for index = 1,#indent_table.indents do
7446     local value = indent_table.indents[index]
7447     local pattern = decode_pattern(value.name)
7448
7449     -- match decoded pattern
7450     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7451     if new_indent_info == nil then
7452         local blankline_end = lpeg.match(
7453             Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7454         if is_optional or not indent_table.ignore_blockquote_blank
7455             or not blankline_end then
7456             return is_optional, new_index, current_trail,
7457                 current_line_indents
7458         end
7459
7460         return traverse_indent(s, tonumber(blankline_end.pos),
7461             indent_table, is_optional, is_blank,
7462             current_line_indents)
7463     end
7464
7465     local raw_last_trail = new_indent_info[1]
7466     local delimiter = new_indent_info[2]
7467     local raw_new_trail = new_indent_info[3]
7468     local next_index = new_indent_info[4]
7469
7470     local space_only = delimiter == ""
7471
7472     -- check previous trail
7473     if not space_only and next(current_trail) == nil then
7474         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7475         current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7476             total_length=sp.total_length,
7477             full_remainder=sp.full_remainder}
7478     end
7479
7480     if next(current_trail) ~= nil then
7481         if not space_only and current_trail.is_code then
7482             return is_optional, new_index, current_trail,
7483                 current_line_indents
7484         end
7485         if current_trail.internal_remainder ~= nil then
7486             raw_last_trail = current_trail.internal_remainder
7487         end
7488     end
7489
7490     local raw_last_trail_length = 0
7491     local delimiter_length = 0

```

```

7492
7493     if not space_only then
7494         delimiter_length = #delimiter
7495         raw_last_trail_length = #raw_last_trail
7496     end
7497
7498     local total_indent_level = preceding_indentation
7499         + raw_last_trail_length + delimiter_length
7500
7501     local spacing_to_process
7502     local minimum = 0
7503     local left_strip_length = 0
7504
7505     if not space_only then
7506         spacing_to_process = raw_new_trail
7507         left_strip_length = 1
7508     else
7509         spacing_to_process = raw_last_trail
7510         minimum = value.length
7511     end
7512
7513     local sp = process_starter_spacing(total_indent_level,
7514         spacing_to_process, minimum,
7515         left_strip_length)
7516
7517     if space_only and not sp.is_minimum then
7518         return is_optional or (is_blank and blank_starter <= index),
7519             new_index, current_trail, current_line_indents
7520     end
7521
7522     local indent_length = raw_last_trail_length + delimiter_length
7523         + sp.left_total_stripped
7524
7525     -- update info for the next pattern
7526     if not space_only then
7527         preceding_indentation = preceding_indentation + indent_length
7528     else
7529         preceding_indentation = preceding_indentation + value.length
7530     end
7531
7532     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7533         internal_remainder=sp.minimum_remainder,
7534         total_length=sp.total_length,
7535         full_remainder=sp.full_remainder}
7536
7537     current_line_indents[#current_line_indents + 1] = new_indent_info
7538     new_index = next_index

```

```

7539 end
7540
7541 return true, new_index, current_trail, current_line_indents
7542 end
7543

```

Check if a code trail is expected.

```

7544 local function check_trail(expect_code, is_code)
7545   return (expect_code and is_code) or (not expect_code and not is_code)
7546 end
7547

```

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`.

```

7548 local check_trail_joined =
7549   function(s, i, indent_table, -- luacheck: ignore s i
7550           spacing, expect_code, omit_remainder)
7551     local is_code
7552     local remainder
7553
7554     if has_trail(indent_table) then
7555       local trail = indent_table.trail
7556       is_code = trail.is_code
7557       if is_code then
7558         remainder = trail.remainder
7559       else
7560         remainder = trail.full_remainder
7561       end
7562     else
7563       local sp = process_starter_spacing(0, spacing, 0, 0)
7564       is_code = sp.is_code
7565       if is_code then
7566         remainder = sp.remainder
7567       else
7568         remainder = sp.full_remainder
7569       end
7570     end
7571
7572     local result = check_trail(expect_code, is_code)
7573     if omit_remainder then
7574       return result
7575     end
7576     return result, remainder
7577   end
7578

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.



```

7579 local check_trail_length =
7580   function(s, i, indent_table, -- luacheck: ignore s i
7581           spacing, min, max)
7582     local trail
7583
7584     if has_trail(indent_table) then
7585       trail = indent_table.trail
7586     else
7587       trail = process_starter_spacing(0, spacing, 0, 0)
7588     end
7589
7590     local total_length = trail.total_length
7591     if total_length == nil then
7592       return false
7593     end
7594
7595     return min <= total_length and total_length <= max
7596   end
7597

```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```

7598 local function check_continuation_indentation(s, i, indent_table,
7599   is_optional, is_blank)
7600   if not has_indents(indent_table) then
7601     return true
7602   end
7603
7604   local passes, new_index, current_trail, current_line_indents =
7605     traverse_indent(s, i, indent_table, is_optional, is_blank)
7606
7607   if passes then
7608     indent_table.current_line_indents = current_line_indents
7609     indent_table = add_trail(indent_table, current_trail)
7610     return new_index, indent_table
7611   end
7612   return false
7613 end
7614

```

Get name of the last indent from the `indent_table`.

```

7615 local function get_last_indent_name(indent_table)
7616   if has_indents(indent_table) then
7617     return indent_table.indents[#indent_table.indents].name
7618   end
7619 end
7620

```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
7621 local function remove_remainder_if_blank(indent_table, remainder)
7622   if get_last_indent_name(indent_table) == "li" then
7623     return ""
7624   end
7625   return remainder
7626 end
7627
```

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.

```
7628 local check_trail_type =
7629   function(s, i, -- luacheck: ignore s i
7630     trail, spacing, trail_type)
7631     if trail == nil then
7632       trail = process_starter_spacing(0, spacing, 0, 0)
7633     end
7634
7635     if trail_type == "non-code" then
7636       return check_trail(false, trail.is_code)
7637     end
7638     if trail_type == "code" then
7639       return check_trail(true, trail.is_code)
7640     end
7641     if trail_type == "full-code" then
7642       if (trail.is_code) then
7643         return i, trail.remainder
7644       end
7645       return i, ""
7646     end
7647     if trail_type == "full-any" then
7648       return i, trail.internal_remainder
7649     end
7650   end
7651
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
7652 local trail_freezing =
7653   function(s, i, -- luacheck: ignore s i
7654     indent_table, is_freezing)
7655     if is_freezing then
7656       if indent_table.is_trail_frozen then
7657         indent_table.trail = indent_table.frozen_trail
7658       else
7659         indent_table.frozen_trail = indent_table.trail
7660         indent_table.is_trail_frozen = true
7661       end
7662     end
7663   end
```

```

7661     end
7662   else
7663     indent_table.frozen_trail = nil
7664     indent_table.is_trail_frozen = false
7665   end
7666   return true, indent_table
7667 end
7668

```

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`.

```

7669 local check_continuation_indentation_and_trail =
7670 function (s, i, indent_table, is_optional, is_blank, trail_type,
7671          reset_rem, omit_remainder)
7672   if not has_indents(indent_table) then
7673     local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
7674   * Cp(), s, i)
7675     local result, remainder = check_trail_type(s, i,
7676       indent_table.trail, spacing, trail_type)
7677     if remainder == nil then
7678       if result then
7679         return new_index
7680       end
7681       return false
7682     end
7683     if result then
7684       return new_index, remainder
7685     end
7686     return false
7687   end
7688
7689   local passes, new_index, current_trail = traverse_indent(s, i,
7690     indent_table, is_optional, is_blank)
7691
7692   if passes then
7693     local spacing
7694     if current_trail == nil then
7695       local newer_spacing, newer_index = lpeg.match(
7696         C(parsers.spacechar^0) * Cp(), s, i)
7697       current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7698       new_index = newer_index
7699       spacing = newer_spacing
7700     else
7701       spacing = current_trail.remainder
7702     end
7703     local result, remainder = check_trail_type(s, new_index,

```

```

7704     current_trail, spacing, trail_type)
7705     if remainder == nil or omit_remainder then
7706         if result then
7707             return new_index
7708         end
7709         return false
7710     end
7711
7712     if is_blank and reset_rem then
7713         remainder = remove_remainder_if_blank(indent_table, remainder)
7714     end
7715     if result then
7716         return new_index, remainder
7717     end
7718     return false
7719 end
7720 return false
7721 end
7722

```

The following patterns check whitespace indentation at the start of a block.

```

7723 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
7724     * Cc(false), check_trail_joined)
7725
7726 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7727     * C(parsers.spacechar^0) * Cc(false)
7728     * Cc(true), check_trail_joined)
7729
7730 parsers.check_code_trail = Cmt( Cb("indent_info")
7731     * C(parsers.spacechar^0)
7732     * Cc(true), check_trail_joined)
7733
7734 parsers.check_trail_length_range = function(min, max)
7735     return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7736         * Cc(max), check_trail_length)
7737 end
7738
7739 parsers.check_trail_length = function(n)
7740     return parsers.check_trail_length_range(n, n)
7741 end
7742

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7743 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
7744     * Cc(true), trail_freezing), "indent_info")
7745
7746 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),

```

```
7747         trail_freezing), "indent_info")
7748
```

The following patterns check indentation in continuation lines as defined by the container start.

```
7749 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7750                                     check_continuation_indentation)
7751
7752 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7753                                     check_continuation_indentation)
7754
7755 parsers.check_minimal_blank_indent
7756 = Cmt( Cb("indent_info") * Cc(false)
7757       * Cc(true)
7758       , check_continuation_indentation)
7759
```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```
7760
7761 parsers.check_minimal_indent_and_trail =
7762   Cmt( Cb("indent_info")
7763       * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7764       , check_continuation_indentation_and_trail)
7765
7766 parsers.check_minimal_indent_and_code_trail =
7767   Cmt( Cb("indent_info")
7768       * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7769       , check_continuation_indentation_and_trail)
7770
7771 parsers.check_minimal_blank_indent_and_full_code_trail =
7772   Cmt( Cb("indent_info")
7773       * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7774       , check_continuation_indentation_and_trail)
7775
7776 parsers.check_minimal_indent_and_any_trail =
7777   Cmt( Cb("indent_info")
7778       * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7779       , check_continuation_indentation_and_trail)
7780
7781 parsers.check_minimal_blank_indent_and_any_trail =
7782   Cmt( Cb("indent_info")
7783       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7784       , check_continuation_indentation_and_trail)
7785
7786 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7787   Cmt( Cb("indent_info")
7788       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)

```

```

7789     , check_continuation_indentation_and_trail)
7790
7791 parsers.check_optional_indent_and_any_trail =
7792   Cmt( Cb("indent_info")
7793     * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7794     , check_continuation_indentation_and_trail)
7795
7796 parsers.check_optional_blank_indent_and_any_trail =
7797   Cmt( Cb("indent_info")
7798     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7799     , check_continuation_indentation_and_trail)
7800

```

The following patterns specify behaviour around newlines.

```

7801
7802 parsers.spnlc_noexc = parsers.optionalspace
7803                   * ( parsers.newline
7804                   * parsers.check_minimal_indent_and_any_trail)^-1
7805
7806 parsers.spnlc = parsers.optionalspace
7807               * (V("EndlineNoSub"))^-1
7808
7809 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7810                 + parsers.spacechar^1
7811
7812 parsers.only_blank = parsers.spacechar^0
7813                   * (parsers.newline + parsers.eof)
7814

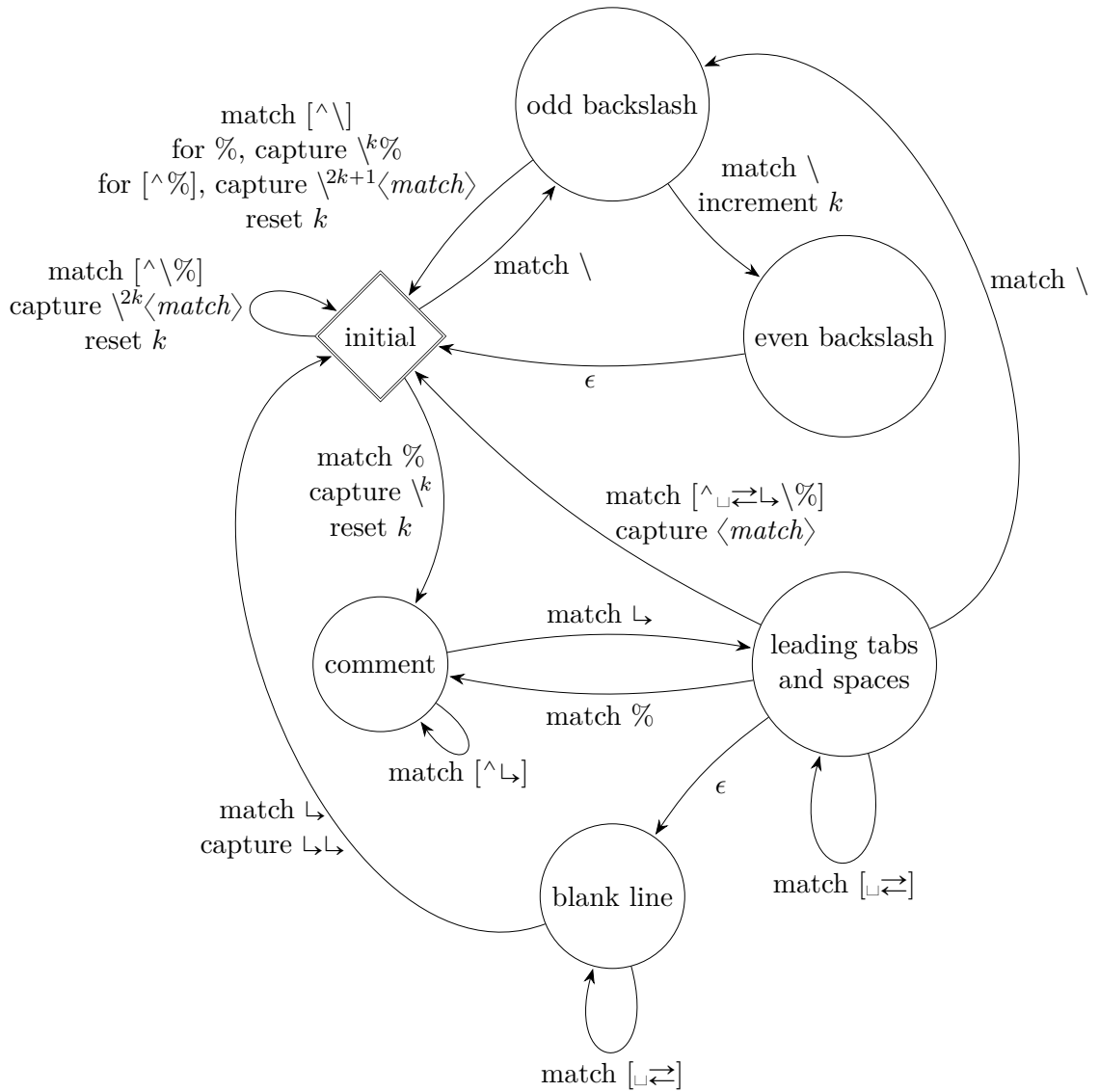
```

The `parsers.commented\_line^1` parser recognizes the regular language of  $\text{T}_{\text{E}}\text{X}$  comments, see an equivalent finite automaton in Figure 6.

```

7815 parsers.commented_line_letter = parsers.linechar
7816                               + parsers.newline
7817                               - parsers.backslash
7818                               - parsers.percent
7819 parsers.commented_line = Cg(Cc(""), "backslashes")
7820                       * ((#(parsers.commented_line_letter
7821                           - parsers.newline)
7822                          * Cb("backslashes")
7823                          * Cs(parsers.commented_line_letter
7824                              - parsers.newline)^1 -- initial
7825                          * Cg(Cc(""), "backslashes"))
7826                       + #( parsers.backslash
7827                           * (parsers.backslash + parsers.newline))
7828                       * Cg((parsers.backslash -- even backslash
7829                           * ( parsers.backslash
7830                             + #parsers.newline))^1, "backslashes")
7831                       + (parsers.backslash

```



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

```

7832 * (#parsers.percent
7833 * Cb("backslashes")
7834 / function(backslashes)
7835     return string.rep("\\", #backslashes / 2)
7836 end
7837 * C(parsers.percent)
7838 + #parsers.commented_line_letter
7839 * Cb("backslashes")
7840 * Cc("\\")
7841 * C(parsers.commented_line_letter))
7842 * Cg(Cc(""), "backslashes"))^0
7843 * (#parsers.percent
7844 * Cb("backslashes")
7845 / function(backslashes)
7846     return string.rep("\\", #backslashes / 2)
7847 end
7848 * ((parsers.percent -- comment
7849 * parsers.line
7850 * #parsers.blankline) -- blank line
7851 / "\n"
7852 + parsers.percent -- comment
7853 * parsers.line
7854 * parsers.optionalspace) -- leading spaces
7855 + #parsers.newline)
7856 * Cb("backslashes")
7857 * C(parsers.newline))
7858
7859 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7860     - parsers.blankline)^0
7861
7862 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7863 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7864 parsers.attribute_key = (parsers.attribute_key_char
7865     - parsers.dash - parsers.digit)
7866 * parsers.attribute_key_char^0
7867 parsers.attribute_value = ( (parsers.dquote / "")
7868     * (parsers.anyescaped - parsers.dquote)^0
7869     * (parsers.dquote / ""))
7870 + ( (parsers.squote / "")
7871     * (parsers.anyescaped - parsers.squote)^0
7872     * (parsers.squote / ""))
7873 + ( parsers.anyescaped
7874     - parsers.dquote
7875     - parsers.rbrace
7876     - parsers.space)^0
7877 parsers.attribute_identifier = parsers.attribute_key_char^1
7878 parsers.attribute_classname = parsers.letter

```



```

7879             * parsers.attribute_key_char^0
7880 parsers.attribute_raw = parsers.attribute_raw_char^1
7881
7882 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7883             + C( parsers.hash
7884                 * parsers.attribute_identifier)
7885             + C( parsers.period
7886                 * parsers.attribute_classname)
7887             + Cs( parsers.attribute_key
7888                 * parsers.optionalspace
7889                 * parsers.equal
7890                 * parsers.optionalspace
7891                 * parsers.attribute_value)
7892 parsers.attributes = parsers.lbrace
7893             * parsers.optionalspace
7894             * parsers.attribute
7895             * (parsers.spacechar^1
7896                 * parsers.attribute)^0
7897             * parsers.optionalspace
7898             * parsers.rbrace
7899
7900 parsers.raw_attribute = parsers.lbrace
7901             * parsers.optionalspace
7902             * parsers.equal
7903             * C(parsers.attribute_raw)
7904             * parsers.optionalspace
7905             * parsers.rbrace
7906
7907 -- block followed by 0 or more optionally
7908 -- indented blocks with first line indented.
7909 parsers.indented_blocks = function(bl)
7910   return Cs( bl
7911             * ( parsers.blankline^1
7912                 * parsers.indent
7913                 * -parsers.blankline
7914                 * bl)^0
7915             * (parsers.blankline^1 + parsers.eof) )
7916 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

7917 local function repeat_between(pattern, min, max)
7918   return -pattern^(max + 1) * pattern^min
7919 end
7920
7921 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7922             * C(repeat_between(parsers.hexdigit, 1, 6))

```

```

7923             * parsers.semicolon
7924 parsers.decentity = parsers.ampersand * parsers.hash
7925             * C(repeat_between(parsers.digit, 1, 7))
7926             * parsers.semicolon
7927 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7928             * parsers.semicolon
7929
7930 parsers.html_entities
7931   = parsers.hexentity / entities.hex_entity_with_x_char
7932   + parsers.decentity / entities.dec_entity
7933   + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

7934 parsers.bullet = function(bullet_char, interrupting)
7935   local allowed_end
7936   if interrupting then
7937     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7938   else
7939     allowed_end = C(parsers.spacechar^1)
7940                 + #(parsers.newline + parsers.eof)
7941   end
7942   return parsers.check_trail
7943           * Ct(C(bullet_char) * Cc(""))
7944           * allowed_end
7945 end
7946
7947 local function tickbox(interior)
7948   return parsers.optionalspace * parsers.lbracket
7949           * interior * parsers.rbracket * parsers.spacechar^1
7950 end
7951
7952 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7953 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7954 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7955

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```

7956 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7957
7958 local function captures_equal_length(_,i,a,b)
7959   return #a == #b and i
7960 end
7961
7962 parsers.closeticks = Cmt(C(parsers.backtick^1)
7963                          * Cb("ticks"), captures_equal_length)
7964

```

```

7965 parsers.intickschar = (parsers.any - S("\n\r`"))
7966                       + V("NoSoftLineBreakEndline")
7967                       + (parsers.backtick^1 - parsers.closeticks)
7968
7969 local function process_inticks(s)
7970   s = s:gsub("\n", " ")
7971   s = s:gsub("^ (.*) $", "%1")
7972   return s
7973 end
7974
7975 parsers.inticks = parsers.openticks
7976                 * C(parsers.space^0)
7977                 * parsers.closeticks
7978                 + parsers.openticks
7979                 * Cs(Cs(parsers.intickschar^0) / process_inticks)
7980                 * parsers.closeticks
7981

```

### 3.1.5.5 Parsers Used for HTML

```

7982 -- case-insensitive match (we assume s is lowercase)
7983 -- must be single byte encoding
7984 parsers.keyword_exact = function(s)
7985   local parser = P(0)
7986   for i=1,#s do
7987     local c = s:sub(i,i)
7988     local m = c .. upper(c)
7989     parser = parser * S(m)
7990   end
7991   return parser
7992 end
7993
7994 parsers.special_block_keyword =
7995   parsers.keyword_exact("pre") +
7996   parsers.keyword_exact("script") +
7997   parsers.keyword_exact("style") +
7998   parsers.keyword_exact("textarea")
7999
8000 parsers.block_keyword =
8001   parsers.keyword_exact("address") +
8002   parsers.keyword_exact("article") +
8003   parsers.keyword_exact("aside") +
8004   parsers.keyword_exact("base") +
8005   parsers.keyword_exact("basefont") +
8006   parsers.keyword_exact("blockquote") +
8007   parsers.keyword_exact("body") +
8008   parsers.keyword_exact("caption") +

```

```
8009     parsers.keyword_exact("center") +
8010     parsers.keyword_exact("col") +
8011     parsers.keyword_exact("colgroup") +
8012     parsers.keyword_exact("dd") +
8013     parsers.keyword_exact("details") +
8014     parsers.keyword_exact("dialog") +
8015     parsers.keyword_exact("dir") +
8016     parsers.keyword_exact("div") +
8017     parsers.keyword_exact("dl") +
8018     parsers.keyword_exact("dt") +
8019     parsers.keyword_exact("fieldset") +
8020     parsers.keyword_exact("figcaption") +
8021     parsers.keyword_exact("figure") +
8022     parsers.keyword_exact("footer") +
8023     parsers.keyword_exact("form") +
8024     parsers.keyword_exact("frame") +
8025     parsers.keyword_exact("frameset") +
8026     parsers.keyword_exact("h1") +
8027     parsers.keyword_exact("h2") +
8028     parsers.keyword_exact("h3") +
8029     parsers.keyword_exact("h4") +
8030     parsers.keyword_exact("h5") +
8031     parsers.keyword_exact("h6") +
8032     parsers.keyword_exact("head") +
8033     parsers.keyword_exact("header") +
8034     parsers.keyword_exact("hr") +
8035     parsers.keyword_exact("html") +
8036     parsers.keyword_exact("iframe") +
8037     parsers.keyword_exact("legend") +
8038     parsers.keyword_exact("li") +
8039     parsers.keyword_exact("link") +
8040     parsers.keyword_exact("main") +
8041     parsers.keyword_exact("menu") +
8042     parsers.keyword_exact("menuitem") +
8043     parsers.keyword_exact("nav") +
8044     parsers.keyword_exact("noframes") +
8045     parsers.keyword_exact("ol") +
8046     parsers.keyword_exact("optgroup") +
8047     parsers.keyword_exact("option") +
8048     parsers.keyword_exact("p") +
8049     parsers.keyword_exact("param") +
8050     parsers.keyword_exact("section") +
8051     parsers.keyword_exact("source") +
8052     parsers.keyword_exact("summary") +
8053     parsers.keyword_exact("table") +
8054     parsers.keyword_exact("tbody") +
8055     parsers.keyword_exact("td") +
```

```

8056     parsers.keyword_exact("tfoot") +
8057     parsers.keyword_exact("th") +
8058     parsers.keyword_exact("thead") +
8059     parsers.keyword_exact("title") +
8060     parsers.keyword_exact("tr") +
8061     parsers.keyword_exact("track") +
8062     parsers.keyword_exact("ul")
8063
8064 -- end conditions
8065 parsers.html_blankline_end_condition
8066 = parsers.linechar^0
8067 * ( parsers.newline
8068     * (parsers.check_minimal_blank_indent_and_any_trail
8069         * #parsers.blankline
8070         + parsers.check_minimal_indent_and_any_trail)
8071     * parsers.linechar^1)^0
8072 * (parsers.newline^-1 / "")
8073
8074 local function remove_trailing_blank_lines(s)
8075     return s:gsub("[\n\r]+%s*$", "")
8076 end
8077
8078 parsers.html_until_end = function(end_marker)
8079     return Cs(Cs((parsers.newline
8080         * (parsers.check_minimal_blank_indent_and_any_trail
8081             * #parsers.blankline
8082             + parsers.check_minimal_indent_and_any_trail)
8083         + parsers.linechar - end_marker)^0
8084         * parsers.linechar^0 * parsers.newline^-1)
8085         / remove_trailing_blank_lines)
8086 end
8087
8088 -- attributes
8089 parsers.html_attribute_spacing = parsers.optionalspace
8090                                 * V("NoSoftLineBreakEndline")
8091                                 * parsers.optionalspace
8092                                 + parsers.spacechar^1
8093
8094 parsers.html_attribute_name = ( parsers.letter
8095                                 + parsers.colon
8096                                 + parsers.underscore)
8097                                 * ( parsers.alphanumeric
8098                                 + parsers.colon
8099                                 + parsers.underscore
8100                                 + parsers.period
8101                                 + parsers.dash)^0
8102

```

```

8103 parsers.html_attribute_value = parsers.squote
8104     * (parsers.linechar - parsers.squote)^0
8105     * parsers.squote
8106     + parsers.dquote
8107     * (parsers.linechar - parsers.dquote)^0
8108     * parsers.dquote
8109     + ( parsers.any
8110         - parsers.spacechar
8111         - parsers.newline
8112         - parsers.dquote
8113         - parsers.squote
8114         - parsers.backtick
8115         - parsers.equal
8116         - parsers.less
8117         - parsers.more)^1
8118
8119 parsers.html_inline_attribute_value = parsers.squote
8120     * (V("NoSoftLineBreakEndline")
8121         + parsers.any
8122         - parsers.blankline^2
8123         - parsers.squote)^0
8124     * parsers.squote
8125     + parsers.dquote
8126     * (V("NoSoftLineBreakEndline")
8127         + parsers.any
8128         - parsers.blankline^2
8129         - parsers.dquote)^0
8130     * parsers.dquote
8131     + (parsers.any
8132         - parsers.spacechar
8133         - parsers.newline
8134         - parsers.dquote
8135         - parsers.squote
8136         - parsers.backtick
8137         - parsers.equal
8138         - parsers.less
8139         - parsers.more)^1
8140
8141 parsers.html_attribute_value_specification
8142     = parsers.optionalspace
8143     * parsers.equal
8144     * parsers.optionalspace
8145     * parsers.html_attribute_value
8146
8147 parsers.html_spnl = parsers.optionalspace
8148     * (V("NoSoftLineBreakEndline")
8149     * parsers.optionalspace)^-1

```

```

8150
8151 parsers.html_inline_attribute_value_specification
8152     = parsers.html_spn1
8153     * parsers.equal
8154     * parsers.html_spn1
8155     * parsers.html_inline_attribute_value
8156
8157 parsers.html_attribute
8158     = parsers.html_attribute_spacing
8159     * parsers.html_attribute_name
8160     * parsers.html_inline_attribute_value_specification^-1
8161
8162 parsers.html_non_newline_attribute
8163     = parsers.spacechar^1
8164     * parsers.html_attribute_name
8165     * parsers.html_attribute_value_specification^-1
8166
8167 parsers.nested_breaking_blank = parsers.newline
8168                               * parsers.check_minimal_blank_indent
8169                               * parsers.blankline
8170
8171 parsers.html_comment_start = P("<!--")
8172
8173 parsers.html_comment_end = P("-->")
8174
8175 parsers.html_comment
8176     = Cs( parsers.html_comment_start
8177           * parsers.html_until_end(parsers.html_comment_end))
8178
8179 parsers.html_inline_comment = (parsers.html_comment_start / "")
8180                               * -P(">") * -P("-->")
8181                               * Cs(( V("NoSoftLineBreakEndline")
8182                                     + parsers.any
8183                                     - parsers.nested_breaking_blank
8184                                     - parsers.html_comment_end)^0)
8185                               * (parsers.html_comment_end / "")
8186
8187 parsers.html_cdatasection_start = P("<![CDATA[")
8188
8189 parsers.html_cdatasection_end = P("]]>")
8190
8191 parsers.html_cdatasection
8192     = Cs( parsers.html_cdatasection_start
8193           * parsers.html_until_end(parsers.html_cdatasection_end))
8194
8195 parsers.html_inline_cdatasection
8196     = parsers.html_cdatasection_start

```

```

8197 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8198     - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
8199 * parsers.html_cdatasection_end
8200
8201 parsers.html_declaration_start = P("<!") * parsers.letter
8202
8203 parsers.html_declaration_end = P(">")
8204
8205 parsers.html_declaration
8206 = Cs( parsers.html_declaration_start
8207     * parsers.html_until_end(parsers.html_declaration_end))
8208
8209 parsers.html_inline_declaration
8210 = parsers.html_declaration_start
8211 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8212     - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
8213 * parsers.html_declaration_end
8214
8215 parsers.html_instruction_start = P("<?")
8216
8217 parsers.html_instruction_end = P("?>")
8218
8219 parsers.html_instruction
8220 = Cs( parsers.html_instruction_start
8221     * parsers.html_until_end(parsers.html_instruction_end))
8222
8223 parsers.html_inline_instruction = parsers.html_instruction_start
8224     * Cs( V("NoSoftLineBreakEndline")
8225         + parsers.any
8226         - parsers.nested_breaking_blank
8227         - parsers.html_instruction_end)^0
8228     * parsers.html_instruction_end
8229
8230 parsers.html_blankline = parsers.newline
8231     * parsers.optionalspace
8232     * parsers.newline
8233
8234 parsers.html_tag_start = parsers.less
8235
8236 parsers.html_tag_closing_start = parsers.less
8237     * parsers.slash
8238
8239 parsers.html_tag_end = parsers.html_spnl
8240     * parsers.more
8241
8242 parsers.html_empty_tag_end = parsers.html_spnl
8243     * parsers.slash

```



```

8244             * parsers.more
8245
8246 -- opening tags
8247 parsers.html_any_open_inline_tag = parsers.html_tag_start
8248             * parsers.keyword
8249             * parsers.html_attribute^0
8250             * parsers.html_tag_end
8251
8252 parsers.html_any_open_tag = parsers.html_tag_start
8253             * parsers.keyword
8254             * parsers.html_non_newline_attribute^0
8255             * parsers.html_tag_end
8256
8257 parsers.html_open_tag = parsers.html_tag_start
8258             * parsers.block_keyword
8259             * parsers.html_attribute^0
8260             * parsers.html_tag_end
8261
8262 parsers.html_open_special_tag = parsers.html_tag_start
8263             * parsers.special_block_keyword
8264             * parsers.html_attribute^0
8265             * parsers.html_tag_end
8266
8267 -- incomplete tags
8268 parsers.incomplete_tag_following = parsers.spacechar
8269             + parsers.more
8270             + parsers.slash * parsers.more
8271             + #(parsers.newline + parsers.eof)
8272
8273 parsers.incomplete_special_tag_following = parsers.spacechar
8274             + parsers.more
8275             + #( parsers.newline
8276                 + parsers.eof)
8277
8278 parsers.html_incomplete_open_tag = parsers.html_tag_start
8279             * parsers.block_keyword
8280             * parsers.incomplete_tag_following
8281
8282 parsers.html_incomplete_open_special_tag
8283 = parsers.html_tag_start
8284 * parsers.special_block_keyword
8285 * parsers.incomplete_special_tag_following
8286
8287 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
8288             * parsers.block_keyword
8289             * parsers.incomplete_tag_following
8290

```

```

8291 parsers.html_incomplete_close_special_tag
8292     = parsers.html_tag_closing_start
8293     * parsers.special_block_keyword
8294     * parsers.incomplete_tag_following
8295
8296 -- closing tags
8297 parsers.html_close_tag = parsers.html_tag_closing_start
8298                       * parsers.block_keyword
8299                       * parsers.html_tag_end
8300
8301 parsers.html_any_close_tag = parsers.html_tag_closing_start
8302                             * parsers.keyword
8303                             * parsers.html_tag_end
8304
8305 parsers.html_close_special_tag = parsers.html_tag_closing_start
8306                                 * parsers.special_block_keyword
8307                                 * parsers.html_tag_end
8308
8309 -- empty tags
8310 parsers.html_any_empty_inline_tag = parsers.html_tag_start
8311                                   * parsers.keyword
8312                                   * parsers.html_attribute^0
8313                                   * parsers.html_empty_tag_end
8314
8315 parsers.html_any_empty_tag = parsers.html_tag_start
8316                             * parsers.keyword
8317                             * parsers.html_non_newline_attribute^0
8318                             * parsers.optionalspace
8319                             * parsers.slash
8320                             * parsers.more
8321
8322 parsers.html_empty_tag = parsers.html_tag_start
8323                         * parsers.block_keyword
8324                         * parsers.html_attribute^0
8325                         * parsers.html_empty_tag_end
8326
8327 parsers.html_empty_special_tag = parsers.html_tag_start
8328                                 * parsers.special_block_keyword
8329                                 * parsers.html_attribute^0
8330                                 * parsers.html_empty_tag_end
8331
8332 parsers.html_incomplete_blocks
8333     = parsers.html_incomplete_open_tag
8334     + parsers.html_incomplete_open_special_tag
8335     + parsers.html_incomplete_close_tag
8336
8337 -- parse special html blocks

```

```

8338 parsers.html_blankline_ending_special_block_opening
8339 = ( parsers.html_close_special_tag
8340   + parsers.html_empty_special_tag)
8341 * #( parsers.optionalspace
8342   * (parsers.newline + parsers.eof))
8343
8344 parsers.html_blankline_ending_special_block
8345 = parsers.html_blankline_ending_special_block_opening
8346 * parsers.html_blankline_end_condition
8347
8348 parsers.html_special_block_opening
8349 = parsers.html_incomplete_open_special_tag
8350 - parsers.html_empty_special_tag
8351
8352 parsers.html_closing_special_block
8353 = parsers.html_special_block_opening
8354 * parsers.html_until_end(parsers.html_close_special_tag)
8355
8356 parsers.html_special_block
8357 = parsers.html_blankline_ending_special_block
8358 + parsers.html_closing_special_block
8359
8360 -- parse html blocks
8361 parsers.html_block_opening = parsers.html_incomplete_open_tag
8362   + parsers.html_incomplete_close_tag
8363
8364 parsers.html_block = parsers.html_block_opening
8365   * parsers.html_blankline_end_condition
8366
8367 -- parse any html blocks
8368 parsers.html_any_block_opening
8369 = ( parsers.html_any_open_tag
8370   + parsers.html_any_close_tag
8371   + parsers.html_any_empty_tag)
8372 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8373
8374 parsers.html_any_block = parsers.html_any_block_opening
8375   * parsers.html_blankline_end_condition
8376
8377 parsers.html_inline_comment_full = parsers.html_comment_start
8378   * -P(">") * -P("->")
8379   * Cs(( V("NoSoftLineBreakEndline")
8380     + parsers.any - P("--")
8381     - parsers.nested_breaking_blank
8382     - parsers.html_comment_end)^0)
8383   * parsers.html_comment_end
8384

```

```

8385 parsers.html_inline_tags = parsers.html_inline_comment_full
8386                             + parsers.html_any_empty_inline_tag
8387                             + parsers.html_inline_instruction
8388                             + parsers.html_inline_cdatasection
8389                             + parsers.html_inline_declaration
8390                             + parsers.html_any_open_inline_tag
8391                             + parsers.html_any_close_tag
8392

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

8393 parsers.urlchar = parsers.anyescaped
8394                 - parsers.newline
8395                 - parsers.more
8396
8397 parsers.auto_link_scheme_part = parsers.alphanumeric
8398                               + parsers.plus
8399                               + parsers.period
8400                               + parsers.dash
8401
8402 parsers.auto_link_scheme = parsers.letter
8403                           * parsers.auto_link_scheme_part
8404                           * parsers.auto_link_scheme_part^-30
8405
8406 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
8407                    * ( parsers.any - parsers.spacing
8408                      - parsers.less - parsers.more)^0
8409
8410 parsers.printable_characters = S(" !#$%&'*/+=?^_`{|}~-")
8411
8412 parsers.email_address_local_part_char = parsers.alphanumeric
8413   + parsers.printable_characters
8414
8415 parsers.email_address_local_part
8416   = parsers.email_address_local_part_char^1
8417
8418 parsers.email_address_dns_label = parsers.alphanumeric
8419                                  * ( parsers.alphanumeric
8420                                    + parsers.dash)^-62
8421                                  * B(parsers.alphanumeric)
8422
8423 parsers.email_address_domain = parsers.email_address_dns_label
8424                               * ( parsers.period
8425                                 * parsers.email_address_dns_label)^0
8426
8427 parsers.email_address = parsers.email_address_local_part
8428                       * parsers.at

```

```

8429             * parsers.email_address_domain
8430
8431 parsers.auto_link_url = parsers.less
8432             * C(parsers.absolute_uri)
8433             * parsers.more
8434
8435 parsers.auto_link_email = parsers.less
8436             * C(parsers.email_address)
8437             * parsers.more
8438
8439 parsers.auto_link_relative_reference = parsers.less
8440             * C(parsers.urlchar^1)
8441             * parsers.more
8442
8443 parsers.autolink = parsers.auto_link_url
8444             + parsers.auto_link_email
8445
8446 -- content in balanced brackets, parentheses, or quotes:
8447 parsers.bracketed = P{ parsers.lbracket
8448             * (( parsers.backslash / "\"" * parsers.rbracket
8449             + parsers.any - (parsers.lbracket
8450             + parsers.rbracket
8451             + parsers.blankline^2)
8452             ) + V(1))^0
8453             * parsers.rbracket }
8454
8455 parsers.inparens = P{ parsers.lparent
8456             * ((parsers.anyescaped - (parsers.lparent
8457             + parsers.rparent
8458             + parsers.blankline^2)
8459             ) + V(1))^0
8460             * parsers.rparent }
8461
8462 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8463             * ((parsers.anyescaped - (parsers.squote
8464             + parsers.blankline^2)
8465             ) + V(1))^0
8466             * parsers.squote }
8467
8468 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8469             * ((parsers.anyescaped - (parsers.dquote
8470             + parsers.blankline^2)
8471             ) + V(1))^0
8472             * parsers.dquote }
8473
8474 parsers.link_text = parsers.lbracket
8475             * Cs((parsers.alphanumeric^1

```

```

8476         + parsers.bracketed
8477         + parsers.inticks
8478         + parsers.autolink
8479         + V("InlineHtml")
8480         + ( parsers.backslash * parsers.backslash)
8481         + ( parsers.backslash
8482           * ( parsers.lbracket
8483             + parsers.rbracket)
8484           + V("NoSoftLineBreakSpace")
8485           + V("NoSoftLineBreakEndline")
8486           + (parsers.any
8487             - ( parsers.newline
8488               + parsers.lbracket
8489               + parsers.rbracket
8490               + parsers.blankline^2))))^0)
8491     * parsers.rbracket
8492
8493 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
8494     * #( ( parsers.any
8495         - parsers.rbracket)^-999
8496         * parsers.rbracket)
8497     * Cs((parsers.alphanumeric^1
8498         + parsers.inticks
8499         + parsers.autolink
8500         + V("InlineHtml")
8501         + ( parsers.backslash * parsers.backslash)
8502         + ( parsers.backslash
8503           * ( parsers.lbracket
8504             + parsers.rbracket)
8505           + V("NoSoftLineBreakSpace")
8506           + V("NoSoftLineBreakEndline")
8507           + (parsers.any
8508             - ( parsers.newline
8509               + parsers.lbracket
8510               + parsers.rbracket
8511               + parsers.blankline^2))))^1)
8512
8513 parsers.link_label = parsers.lbracket
8514     * parsers.link_label_body
8515     * parsers.rbracket
8516
8517 parsers.inparens_url = P{ parsers.lparent
8518     * ((parsers.anyescaped - (parsers.lparent
8519     + parsers.rparent
8520     + parsers.spacing)
8521     ) + V(1))^0
8522     * parsers.rparent }

```

```

8523
8524 -- url for markdown links, allowing nested brackets:
8525 parsers.url      = parsers.less * Cs((parsers.anyescaped
8526                  - parsers.newline
8527                  - parsers.less
8528                  - parsers.more)^0)
8529                  * parsers.more
8530
8531                  + -parsers.less
8532                  * Cs((parsers.inparens_url + (parsers.anyescaped
8533                  - parsers.spacing
8534                  - parsers.lparent
8535                  - parsers.rparent))^1)
8536 -- quoted text:
8537 parsers.title_s   = parsers.squote
8538                  * Cs((parsers.html_entities
8539                  + V("NoSoftLineBreakSpace")
8540                  + V("NoSoftLineBreakEndline")
8541                  + ( parsers.anyescaped
8542                  - parsers.newline
8543                  - parsers.squote
8544                  - parsers.blankline^2))^0)
8545                  * parsers.squote
8546
8547 parsers.title_d   = parsers.dquote
8548                  * Cs((parsers.html_entities
8549                  + V("NoSoftLineBreakSpace")
8550                  + V("NoSoftLineBreakEndline")
8551                  + ( parsers.anyescaped
8552                  - parsers.newline
8553                  - parsers.dquote
8554                  - parsers.blankline^2))^0)
8555                  * parsers.dquote
8556
8557 parsers.title_p   = parsers.lparent
8558                  * Cs((parsers.html_entities
8559                  + V("NoSoftLineBreakSpace")
8560                  + V("NoSoftLineBreakEndline")
8561                  + ( parsers.anyescaped
8562                  - parsers.newline
8563                  - parsers.lparent
8564                  - parsers.rparent
8565                  - parsers.blankline^2))^0)
8566                  * parsers.rparent
8567
8568 parsers.title     = parsers.title_d + parsers.title_s + parsers.title_p
8569

```

```

8570
8571 parsers.optionaltitle
8572   = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8573

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

8574 -- parse a reference definition: [foo]: /bar "title"
8575 parsers.define_reference_parser = (parsers.check_trail / "")
8576                                 * parsers.link_label * parsers.colon
8577                                 * parsers.spnlc * parsers.url
8578                                 * ( parsers.spnlc_sep * parsers.title
8579                                   * parsers.only_blank
8580                                   + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

8581 parsers.Inline           = V("Inline")
8582
8583 -- parse many p between starter and ender
8584 parsers.between = function(p, starter, ender)
8585   local ender2 = B(parsers.nonspacechar) * ender
8586   return ( starter
8587           * #parsers.nonspacechar
8588           * Ct(p * (p - ender2)^0)
8589           * ender2)
8590 end
8591

```

### 3.1.5.9 Block Elements

```

8592 parsers.lineof = function(c)
8593   return ( parsers.check_trail_no_rem
8594           * (P(c) * parsers.optionalspace)^3
8595           * (parsers.newline + parsers.eof))
8596 end
8597
8598 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8599                               + parsers.lineof(parsers.dash)
8600                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

8601 -- parse Atx heading start and return level
8602 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8603                       * -parsers.hash / length
8604
8605 -- parse setext header ending and return level
8606 parsers.heading_level

```



```

8607 = parsers.nonindentspace * parsers.equal^1
8608 * parsers.optionalspace * #parsers.newline * Cc(1)
8609 + parsers.nonindentspace * parsers.dash^1
8610 * parsers.optionalspace * #parsers.newline * Cc(2)
8611
8612 local function strip_atx_end(s)
8613   return s:gsub("%s+#+%s*\n$", "")
8614 end
8615
8616 parsers.atx_heading = parsers.check_trail_no_rem
8617                       * Cg(parsers.heading_start, "level")
8618                       * (C( parsers.optionalspace
8619                           * parsers.hash^0
8620                           * parsers.optionalspace
8621                           * parsers.newline)
8622                       + parsers.spacechar^1
8623                       * C(parsers.line))

```

### 3.1.6 Markdown Reader

This section documents the `reader` object, which implements the routines for parsing the markdown input. The object corresponds to the markdown reader object that was located in the `lunamark/reader/markdown.lua` file in the Lunamark Lua module.

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

The objects produced by the `reader.new` method expose instance methods and variables of their own. As a convention, I will refer to these *member*s as `reader->member`.

```

8624 M.reader = {}
8625 function M.reader.new(writer, options)
8626   local self = {}

```

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

```

8627   self.writer = writer
8628   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.

```

8629   self.parsers = {}
8630   (function(parsers)
8631     setmetatable(self.parsers, {
8632       __index = function (_, key)
8633         return parsers[key]

```

```

8634     end
8635   })
8636 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.

```

8637 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.

```

8638 function self.normalize_tag(tag)
8639   tag = util.ropetostring(tag)
8640   tag = tag:gsub("[\n\r\t]+", " ")
8641   tag = tag:gsub("^ ", ""):gsub(" $", "")
8642   tag = uni_algos.case.casefold(tag, true, false)
8643   return tag
8644 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.

```

8645 local function iterlines(s, f)
8646   local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8647   return util.ropetostring(rope)
8648 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.

```

8649 if options.preserveTabs then
8650   self.expandtabs = function(s) return s end
8651 else
8652   self.expandtabs = function(s)
8653     if s:find("\t") then
8654       return iterlines(s, util.expand_tabs_in_line)
8655     else
8656       return s
8657     end
8658   end
8659 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.

```
8660 self.parser_functions = {}
8661 self.create_parser = function(name, grammar, toplevel)
8662     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.

```
8663     if toplevel and options.stripIndent then
8664         local min_prefix_length, min_prefix = nil, ''
8665         str = iterlines(str, function(line)
8666             if lpeg.match(parsers.nonemptyline, line) == nil then
8667                 return line
8668             end
8669             line = util.expand_tabs_in_line(line)
8670             local prefix = lpeg.match(C(parsers.optionalspace), line)
8671             local prefix_length = #prefix
8672             local is_shorter = min_prefix_length == nil
8673             if not is_shorter then
8674                 is_shorter = prefix_length < min_prefix_length
8675             end
8676             if is_shorter then
8677                 min_prefix_length, min_prefix = prefix_length, prefix
8678             end
8679             return line
8680         end)
8681         str = str:gsub('^' .. min_prefix, '')
8682     end
```

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

```
8683     if toplevel and (options.texComments or options.hybrid) then
8684         str = lpeg.match(Ct(parsers.commented_line^1), str)
8685         str = util.rope_to_string(str)
8686     end
8687     local res = lpeg.match(grammar(), str)
8688     if res == nil then
8689         return writer.error(
8690             format("Parser `%s` failed to process the input text.", name),
8691             format("Here are the first 20 characters of the remaining "
8692                 .. "unprocessed text: `%s`.", str:sub(1,20))
8693         )
8694     else
8695         return res
```

```

8696     end
8697   end
8698 end
8699
8700 self.create_parser("parse_blocks",
8701                   function()
8702                     return parsers.blocks
8703                   end, true)
8704
8705 self.create_parser("parse_blocks_nested",
8706                   function()
8707                     return parsers.blocks_nested
8708                   end, false)
8709
8710 self.create_parser("parse_inlines",
8711                   function()
8712                     return parsers.inlines
8713                   end, false)
8714
8715 self.create_parser("parse_inlines_no_inline_note",
8716                   function()
8717                     return parsers.inlines_no_inline_note
8718                   end, false)
8719
8720 self.create_parser("parse_inlines_no_html",
8721                   function()
8722                     return parsers.inlines_no_html
8723                   end, false)
8724
8725 self.create_parser("parse_inlines_nbsp",
8726                   function()
8727                     return parsers.inlines_nbsp
8728                   end, false)
8729 self.create_parser("parse_inlines_no_link_or_emphasis",
8730                   function()
8731                     return parsers.inlines_no_link_or_emphasis
8732                   end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8733 parsers.minimally_indented_blankline
8734   = parsers.check_minimal_indent * (parsers.blankline / "")
8735
8736 parsers.minimally_indented_block
8737   = parsers.check_minimal_indent * V("Block")
8738

```

```

8739 parsers.minimally_indented_block_or_paragraph
8740     = parsers.check_minimal_indent * V("BlockOrParagraph")
8741
8742 parsers.minimally_indented_paragraph
8743     = parsers.check_minimal_indent * V("Paragraph")
8744
8745 parsers.minimally_indented_plain
8746     = parsers.check_minimal_indent * V("Plain")
8747
8748 parsers.minimally_indented_par_or_plain
8749     = parsers.minimally_indented_paragraph
8750     + parsers.minimally_indented_plain
8751
8752 parsers.minimally_indented_par_or_plain_no_blank
8753     = parsers.minimally_indented_par_or_plain
8754     - parsers.minimally_indented_blankline
8755
8756 parsers.minimally_indented_ref
8757     = parsers.check_minimal_indent * V("Reference")
8758
8759 parsers.minimally_indented_blank
8760     = parsers.check_minimal_indent * V("Blank")
8761
8762 parsers.conditionally_indented_blankline
8763     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8764
8765 parsers.minimally_indented_ref_or_block
8766     = parsers.minimally_indented_ref
8767     + parsers.minimally_indented_block
8768     - parsers.minimally_indented_blankline
8769
8770 parsers.minimally_indented_ref_or_block_or_par
8771     = parsers.minimally_indented_ref
8772     + parsers.minimally_indented_block_or_paragraph
8773     - parsers.minimally_indented_blankline
8774

```

The following pattern parses the properly indented content that follows the initial container start.

```

8775
8776 function parsers.separator_loop(separated_block, paragraph,
8777                                block_separator, paragraph_separator)
8778     return separated_block
8779         + block_separator
8780         * paragraph
8781         * separated_block
8782         + paragraph_separator

```

```

8783         * paragraph
8784     end
8785
8786     function parsers.create_loop_body_pair(separated_block, paragraph,
8787   block_separator,
8788   paragraph_separator)
8789         return {
8790             block = parsers.separator_loop(separated_block, paragraph,
8791   block_separator, block_separator),
8792             par = parsers.separator_loop(separated_block, paragraph,
8793   block_separator, paragraph_separator)
8794         }
8795     end
8796
8797     parsers.block_sep_group = function(blank)
8798         return blank^0 * parsers.eof
8799             + ( blank^2 / writer.paragraphsep
8800               + blank^0 / writer.interblocksep
8801             )
8802     end
8803
8804     parsers.par_sep_group = function(blank)
8805         return blank^0 * parsers.eof
8806             + blank^0 / writer.paragraphsep
8807     end
8808
8809     parsers.sep_group_no_output = function(blank)
8810         return blank^0 * parsers.eof
8811             + blank^0
8812     end
8813
8814     parsers.content_blank = parsers.minimally_indented_blankline
8815
8816     parsers.ref_or_block_separated
8817         = parsers.sep_group_no_output(parsers.content_blank)
8818         * ( parsers.minimally_indented_ref
8819           - parsers.content_blank)
8820         + parsers.block_sep_group(parsers.content_blank)
8821         * ( parsers.minimally_indented_block
8822           - parsers.content_blank)
8823
8824     parsers.loop_body_pair =
8825         parsers.create_loop_body_pair(
8826             parsers.ref_or_block_separated,
8827             parsers.minimally_indented_par_or_plain_no_blank,
8828             parsers.block_sep_group(parsers.content_blank),
8829             parsers.par_sep_group(parsers.content_blank))

```

```

8830
8831 parsers.content_loop = ( V("Block")
8832     * parsers.loop_body_pair.block^0
8833     + (V("Paragraph") + V("Plain"))
8834     * parsers.ref_or_block_separated
8835     * parsers.loop_body_pair.block^0
8836     + (V("Paragraph") + V("Plain"))
8837     * parsers.loop_body_pair.par^0
8838     * parsers.content_blank^0
8839
8840 parsers.indented_content = function()
8841     return Ct( (V("Reference") + (parsers.blankline / ""))
8842         * parsers.content_blank^0
8843         * parsers.check_minimal_indent
8844         * parsers.content_loop
8845         + (V("Reference") + (parsers.blankline / ""))
8846         * parsers.content_blank^0
8847         + parsers.content_loop)
8848 end
8849
8850 parsers.add_indent = function(pattern, name, breakable)
8851     return Cg(Cmt( Cb("indent_info")
8852         * Ct(pattern)
8853         * ( #parsers.linechar -- check if starter is blank
8854             * Cc(false) + Cc(true))
8855         * Cc(name)
8856         * Cc(breakable),
8857         process_starter_indent), "indent_info")
8858 end
8859

```

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

```

8860 if options.hashEnumerators then
8861     parsers.dig = parsers.digit + parsers.hash
8862 else
8863     parsers.dig = parsers.digit
8864 end
8865
8866 parsers.enumerator = function(delimiter_type, interrupting)
8867     local delimiter_range
8868     local allowed_end
8869     if interrupting then
8870         delimiter_range = P("1")
8871         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8872     else
8873         delimiter_range = parsers.dig * parsers.dig^-8

```

```

8874     allowed_end = C(parsers.spacechar^1)
8875                   + #(parsers.newline + parsers.eof)
8876   end
8877
8878   return parsers.check_trail
8879         * Ct(C(delimiter_range) * C(delimiter_type))
8880         * allowed_end
8881 end
8882
8883 parsers.starter = parsers.bullet(parsers.dash)
8884                 + parsers.bullet(parsers.asterisk)
8885                 + parsers.bullet(parsers.plus)
8886                 + parsers.enumerator(parsers.period)
8887                 + parsers.enumerator(parsers.rparent)
8888

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

8889 parsers.blockquote_start
8890   = parsers.check_trail
8891   * C(parsers.more)
8892   * C(parsers.spacechar^0)
8893
8894 parsers.blockquote_body
8895   = parsers.add_indent(parsers.blockquote_start, "bq", true)
8896   * parsers.indented_content()
8897   * remove_indent("bq")
8898
8899 if not options.breakableBlockquotes then
8900   parsers.blockquote_body
8901     = parsers.add_indent(parsers.blockquote_start, "bq", false)
8902     * parsers.indented_content()
8903     * remove_indent("bq")
8904 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.

```

8905 local function parse_content_part(content_part)
8906   local rope = util.rope_to_string(content_part)
8907   local parsed
8908     = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8909   parsed.indent_info = nil
8910   return parsed
8911 end
8912

```



Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8913 local collect_emphasis_content =
8914   function(t, opening_index, closing_index)
8915     local content = {}
8916
8917     local content_part = {}
8918     for i = opening_index, closing_index do
8919       local value = t[i]
8920
8921       if value.rendered ~= nil then
8922         content[#content + 1] = parse_content_part(content_part)
8923         content_part = {}
8924         content[#content + 1] = value.rendered
8925         value.rendered = nil
8926       else
8927         if value.type == "delimiter"
8928           and value.element == "emphasis" then
8929           if value.is_active then
8930             content_part[#content_part + 1]
8931               = string.rep(value.character, value.current_count)
8932           end
8933         else
8934           content_part[#content_part + 1] = value.content
8935         end
8936         value.content = ''
8937         value.is_active = false
8938       end
8939     end
8940
8941     if next(content_part) ~= nil then
8942       content[#content + 1] = parse_content_part(content_part)
8943     end
8944
8945     return content
8946   end
8947
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```
8948 local function fill_emph(t, opening_index, closing_index)
8949   local content
8950     = collect_emphasis_content(t, opening_index + 1,
8951                               closing_index - 1)
8952   t[opening_index + 1].is_active = true
8953   t[opening_index + 1].rendered = writer.emphasis(content)
8954 end
```

8955

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
8956 local function fill_strong(t, opening_index, closing_index)
8957     local content
8958         = collect_emphasis_content(t, opening_index + 1,
8959                                     closing_index - 1)
8960     t[opening_index + 1].is_active = true
8961     t[opening_index + 1].rendered = writer.strong(content)
8962 end
8963
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
8964 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8965     return ( opening_delimiter.is_closing
8966             or closing_delimiter.is_opening)
8967     and (( opening_delimiter.original_count
8968            + closing_delimiter.original_count) % 3 == 0)
8969     and ( opening_delimiter.original_count % 3 ~= 0
8970          or closing_delimiter.original_count % 3 ~= 0)
8971 end
8972
```

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`.

```
8973 local find_emphasis_opener = function(t, bottom_index, latest_index,
8974                                     character, closing_delimiter)
8975     for i = latest_index, bottom_index, -1 do
8976         local value = t[i]
8977         if value.is_active and
8978            value.is_opening and
8979            value.type == "delimiter" and
8980            value.element == "emphasis" and
8981            (value.character == character) and
8982            (value.current_count > 0) then
8983             if not breaks_three_rule(value, closing_delimiter) then
8984                 return i
8985             end
8986         end
8987     end
8988 end
8989
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

8990 local function process_emphasis(t, opening_index, closing_index)
8991     for i = opening_index, closing_index do
8992         local value = t[i]
8993         if value.type == "delimiter" and value.element == "emphasis" then
8994             local delimiter_length = string.len(value.content)
8995             value.character = string.sub(value.content, 1, 1)
8996             value.current_count = delimiter_length
8997             value.original_count = delimiter_length
8998         end
8999     end
9000
9001     local openers_bottom = {
9002         ['*'] = {
9003             [true] = {opening_index, opening_index, opening_index},
9004             [false] = {opening_index, opening_index, opening_index}
9005         },
9006         ['_'] = {
9007             [true] = {opening_index, opening_index, opening_index},
9008             [false] = {opening_index, opening_index, opening_index}
9009         }
9010     }
9011
9012     local current_position = opening_index
9013     local max_position = closing_index
9014
9015     while current_position <= max_position do
9016         local value = t[current_position]
9017
9018         if value.type ~= "delimiter" or
9019            value.element ~= "emphasis" or
9020            not value.is_active or
9021            not value.is_closing or
9022            (value.current_count <= 0) then
9023             current_position = current_position + 1
9024             goto continue
9025         end
9026
9027         local character = value.character
9028         local is_opening = value.is_opening
9029         local closing_length_modulo_three = value.original_count % 3
9030
9031         local current_openers_bottom
9032             = openers_bottom[character][is_opening]
9033             [closing_length_modulo_three + 1]
9034
9035         local opener_position
9036             = find_emphasis_opener(t, current_openers_bottom,

```

```

9037             current_position - 1, character, value)
9038
9039     if (opener_position == nil) then
9040         openers_bottom[character][is_opening]
9041             [closing_length_modulo_three + 1]
9042         = current_position
9043         current_position = current_position + 1
9044         goto continue
9045     end
9046
9047     local opening_delimiter = t[opener_position]
9048
9049     local current_opening_count = opening_delimiter.current_count
9050     local current_closing_count = t[current_position].current_count
9051
9052     if (current_opening_count >= 2)
9053         and (current_closing_count >= 2) then
9054         opening_delimiter.current_count = current_opening_count - 2
9055         t[current_position].current_count = current_closing_count - 2
9056         fill_strong(t, opener_position, current_position)
9057     else
9058         opening_delimiter.current_count = current_opening_count - 1
9059         t[current_position].current_count = current_closing_count - 1
9060         fill_emph(t, opener_position, current_position)
9061     end
9062
9063     ::continue::
9064 end
9065 end
9066
9067 local cont = lpeg.R("\128\191") -- continuation byte
9068

```

Match a UTF-8 character of byte length *n*.

```

9069 local function utf8_by_byte_count(n)
9070     if (n == 1) then
9071         return lpeg.R("\0\127")
9072     end
9073     if (n == 2) then
9074         return lpeg.R("\194\223") * cont
9075     end
9076     if (n == 3) then
9077         return lpeg.R("\224\239") * cont * cont
9078     end
9079     if (n == 4) then
9080         return lpeg.R("\240\244") * cont * cont * cont
9081     end
9082 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`.

```
9083 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
9084     local c
9085     local char_length
9086     for pos = start_pos, end_pos, 1 do
9087         if (start_pos < 0) then
9088             char_length = -pos
9089         else
9090             char_length = pos + 1
9091         end
9092
9093         if (chartype == "punctuation") then
9094             if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
9095                 return i
9096             end
9097         else
9098             c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
9099             if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
9100                 return i
9101             end
9102         end
9103     end
9104 end
9105
9106 local function check_preceding_unicode_punctuation(s, i)
9107     return check_unicode_type(s, i, -4, -1, "punctuation")
9108 end
9109
9110 local function check_preceding_unicode_whitespace(s, i)
9111     return check_unicode_type(s, i, -4, -1, "%s")
9112 end
9113
9114 local function check_following_unicode_punctuation(s, i)
9115     return check_unicode_type(s, i, 0, 3, "punctuation")
9116 end
9117
9118 local function check_following_unicode_whitespace(s, i)
9119     return check_unicode_type(s, i, 0, 3, "%s")
9120 end
9121
9122 parsers.unicode_preceding_punctuation
9123     = B(parsers.escapable)
9124     + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
9125
9126 parsers.unicode_preceding_whitespace
9127     = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
```

```

9128
9129 parsers.unicode_following_punctuation
9130     = #parsers.escapable
9131     + Cmt(parsers.succeed, check_following_unicode_punctuation)
9132
9133 parsers.unicode_following_whitespace
9134     = Cmt(parsers.succeed, check_following_unicode_whitespace)
9135
9136 parsers.delimiter_run = function(character)
9137     return (B(parsers.backslash * character) + -B(character))
9138             * character^1
9139             * -#character
9140 end
9141
9142 parsers.left_flanking_delimiter_run = function(character)
9143     return (B( parsers.any)
9144             * ( parsers.unicode_preceding_punctuation
9145                 + parsers.unicode_preceding_whitespace)
9146             + -B(parsers.any))
9147             * parsers.delimiter_run(character)
9148             * parsers.unicode_following_punctuation
9149             + parsers.delimiter_run(character)
9150             * -( parsers.unicode_following_punctuation
9151                 + parsers.unicode_following_whitespace
9152                 + parsers.eof)
9153 end
9154
9155 parsers.right_flanking_delimiter_run = function(character)
9156     return parsers.unicode_preceding_punctuation
9157             * parsers.delimiter_run(character)
9158             * ( parsers.unicode_following_punctuation
9159                 + parsers.unicode_following_whitespace
9160                 + parsers.eof)
9161             + (B(parsers.any)
9162                 * -( parsers.unicode_preceding_punctuation
9163                     + parsers.unicode_preceding_whitespace))
9164             * parsers.delimiter_run(character)
9165 end
9166
9167 if options.underscores then
9168     parsers.emph_start
9169         = parsers.left_flanking_delimiter_run(parsers.asterisk)
9170         + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
9171             + ( parsers.unicode_preceding_punctuation
9172                 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
9173         * parsers.left_flanking_delimiter_run(parsers.underscore)
9174

```

```

9175     parsers.emph_end
9176         = parsers.right_flanking_delimiter_run(parsers.asterisk)
9177         + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
9178           + #( parsers.left_flanking_delimiter_run(parsers.underscore)
9179             * parsers.unicode_following_punctuation))
9180         * parsers.right_flanking_delimiter_run(parsers.underscore)
9181     else
9182         parsers.emph_start
9183             = parsers.left_flanking_delimiter_run(parsers.asterisk)
9184
9185         parsers.emph_end
9186             = parsers.right_flanking_delimiter_run(parsers.asterisk)
9187     end
9188
9189     parsers.emph_capturing_open_and_close
9190         = #parsers.emph_start * #parsers.emph_end
9191         * Ct( Cg(Cc("delimiter"), "type")
9192           * Cg(Cc("emphasis"), "element")
9193           * Cg(C(parsers.emph_start), "content")
9194           * Cg(Cc(true), "is_opening")
9195           * Cg(Cc(true), "is_closing"))
9196
9197     parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
9198           * Cg(Cc("emphasis"), "element")
9199           * Cg(C(parsers.emph_start), "content")
9200           * Cg(Cc(true), "is_opening")
9201           * Cg(Cc(false), "is_closing"))
9202
9203     parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
9204           * Cg(Cc("emphasis"), "element")
9205           * Cg(C(parsers.emph_end), "content")
9206           * Cg(Cc(false), "is_opening")
9207           * Cg(Cc(true), "is_closing"))
9208
9209     parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
9210           + parsers.emph_capturing_open
9211           + parsers.emph_capturing_close
9212
9213     parsers.emph_open = parsers.emph_capturing_open_and_close
9214           + parsers.emph_capturing_open
9215
9216     parsers.emph_close = parsers.emph_capturing_open_and_close
9217           + parsers.emph_capturing_close
9218

```

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

```

9219  -- List of references defined in the document
9220  local references
9221
9222  -- List of note references defined in the document
9223  parsers.rawnotes = {}
9224

```

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.

```

9225  function self.register_link(_, tag, url, title,
9226                               attributes)
9227      local normalized_tag = self.normalize_tag(tag)
9228      if references[normalized_tag] == nil then
9229          references[normalized_tag] = {
9230              url = url,
9231              title = title,
9232              attributes = attributes
9233          }
9234      end
9235      return ""
9236  end
9237

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

9238  function self.lookup_reference(tag)
9239      return references[self.normalize_tag(tag)]
9240  end
9241

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

9242  function self.lookup_note_reference(tag)
9243      return parsers.rawnotes[self.normalize_tag(tag)]
9244  end
9245
9246  parsers.title_s_direct_ref = parsers.squote
9247                               * Cs((parsers.html_entities
9248                                   + ( parsers.anyescaped
9249                                       - parsers.squote
9250                                       - parsers.blankline^2))^0)
9251                               * parsers.squote
9252
9253  parsers.title_d_direct_ref = parsers.dquote
9254                               * Cs((parsers.html_entities
9255                                   + ( parsers.anyescaped
9256                                       - parsers.dquote
9257                                       - parsers.blankline^2))^0)

```



```

9258             * parsers.dquote
9259
9260 parsers.title_p_direct_ref = parsers.lparent
9261             * Cs((parsers.html_entities
9262                 + ( parsers.anyescaped
9263                   - parsers.lparent
9264                   - parsers.rparent
9265                   - parsers.blankline^2))^0)
9266             * parsers.rparent
9267
9268 parsers.title_direct_ref = parsers.title_s_direct_ref
9269             + parsers.title_d_direct_ref
9270             + parsers.title_p_direct_ref
9271
9272 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
9273             * Cg(parsers.url + Cc(""), "url")
9274             * parsers.spnl
9275             * Cg( parsers.title_direct_ref
9276                 + Cc(""), "title")
9277             * parsers.spnl * parsers.rparent
9278
9279 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
9280             * Cg(parsers.url + Cc(""), "url")
9281             * parsers.spnlc
9282             * Cg(parsers.title + Cc(""), "title")
9283             * parsers.spnlc * parsers.rparent
9284
9285 parsers.empty_link = parsers.lbracket
9286             * parsers.rbracket
9287
9288 parsers.inline_link = parsers.link_text
9289             * parsers.inline_direct_ref
9290
9291 parsers.full_link = parsers.link_text
9292             * parsers.link_label
9293
9294 parsers.shortcut_link = parsers.link_label
9295             * -(parsers.empty_link + parsers.link_label)
9296
9297 parsers.collapsed_link = parsers.link_label
9298             * parsers.empty_link
9299
9300 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
9301             * Cg(Cc("inline"), "link_type")
9302             + #(parsers.exclamation * parsers.full_link)
9303             * Cg(Cc("full"), "link_type")
9304             + #( parsers.exclamation

```

```

9305         * parsers.collapsed_link)
9306     * Cg(Cc("collapsed"), "link_type")
9307     + #(parsers.exclamation * parsers.shortcut_link)
9308     * Cg(Cc("shortcut"), "link_type")
9309     + #(parsers.exclamation * parsers.empty_link)
9310     * Cg(Cc("empty"), "link_type")
9311
9312     parsers.link_opening = #parsers.inline_link
9313     * Cg(Cc("inline"), "link_type")
9314     + #parsers.full_link
9315     * Cg(Cc("full"), "link_type")
9316     + #parsers.collapsed_link
9317     * Cg(Cc("collapsed"), "link_type")
9318     + #parsers.shortcut_link
9319     * Cg(Cc("shortcut"), "link_type")
9320     + #parsers.empty_link
9321     * Cg(Cc("empty_link"), "link_type")
9322     + #parsers.link_text
9323     * Cg(Cc("link_text"), "link_type")
9324
9325     parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9326     * Cg(Cc("note_inline"), "link_type")
9327
9328     parsers.raw_note_opening = #( parsers.lbracket
9329         * parsers.circumflex
9330         * parsers.link_label_body
9331         * parsers.rbracket)
9332     * Cg(Cc("raw_note"), "link_type")
9333
9334     local inline_note_element = Cg(Cc("note"), "element")
9335     * parsers.note_opening
9336     * Cg( parsers.circumflex
9337         * parsers.lbracket, "content")
9338
9339     local image_element = Cg(Cc("image"), "element")
9340     * parsers.image_opening
9341     * Cg( parsers.exclamation
9342         * parsers.lbracket, "content")
9343
9344     local note_element = Cg(Cc("note"), "element")
9345     * parsers.raw_note_opening
9346     * Cg( parsers.lbracket
9347         * parsers.circumflex, "content")
9348
9349     local link_element = Cg(Cc("link"), "element")
9350     * parsers.link_opening
9351     * Cg(parsers.lbracket, "content")

```

```

9352
9353 local opening_elements = parsers.fail
9354
9355 if options.inlineNotes then
9356   opening_elements = opening_elements + inline_note_element
9357 end
9358
9359 opening_elements = opening_elements + image_element
9360
9361 if options.notes then
9362   opening_elements = opening_elements + note_element
9363 end
9364
9365 opening_elements = opening_elements + link_element
9366
9367 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
9368   * Cg(Cc(true), "is_opening")
9369   * Cg(Cc(false), "is_closing")
9370   * opening_elements)
9371
9372 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
9373   * Cg(Cc("link"), "element")
9374   * Cg(Cc(false), "is_opening")
9375   * Cg(Cc(true), "is_closing")
9376   * ( Cg(Cc(true), "is_direct")
9377     * Cg( parsers.rbracket
9378       * #parsers.inline_direct_ref,
9379         "content")
9380     + Cg(Cc(false), "is_direct")
9381     * Cg(parsers.rbracket, "content")))
9382
9383 parsers.link_image_open_or_close = parsers.link_image_opening
9384   + parsers.link_image_closing
9385
9386 if options.html then
9387   parsers.link_emph_precedence = parsers.inticks
9388   + parsers.autolink
9389   + parsers.html_inline_tags
9390 else
9391   parsers.link_emph_precedence = parsers.inticks
9392   + parsers.autolink
9393 end
9394
9395 parsers.link_and_emph_endline = parsers.newline
9396   * ((parsers.check_minimal_indent
9397     * -V("EndlineExceptions")
9398     + parsers.check_optional_indent

```

```

9399             * -V("EndlineExceptions")
9400             * -V("ListStarter")) / "")
9401             * parsers.spacechar^0 / "\n"
9402
9403 parsers.link_and_emph_content
9404   = Ct( Cg(Cc("content"), "type")
9405         * Cg(Cs(( parsers.link_emph_precedence
9406                   + parsers.backslash * parsers.linechar
9407                   + parsers.link_and_emph_endline
9408                   + (parsers.linechar
9409                     - parsers.blankline^2
9410                     - parsers.link_image_open_or_close
9411                     - parsers.emph_open_or_close))^0), "content"))
9412
9413 parsers.link_and_emph_table
9414   = (parsers.link_image_opening + parsers.emph_open)
9415     * parsers.link_and_emph_content
9416     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9417         * parsers.link_and_emph_content)^1
9418

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9419 local function collect_link_content(t, opening_index, closing_index)
9420   local content = {}
9421   for i = opening_index, closing_index do
9422     content[#content + 1] = t[i].content
9423   end
9424   return util.ropetostring(content)
9425 end
9426

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

9427 local function find_link_opener(t, bottom_index, latest_index)
9428   for i = latest_index, bottom_index, -1 do
9429     local value = t[i]
9430     if value.type == "delimiter" and
9431        value.is_opening and
9432        ( value.element == "link"
9433          or value.element == "image"
9434          or value.element == "note")
9435        and not value.removed then
9436       if value.is_active then
9437         return i
9438       end
9439       value.removed = true
9440     end
9441   end
9442   return nil

```

```

9441     end
9442   end
9443 end
9444

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

9445   local function find_next_link_closing_index(t, latest_index)
9446     for i = latest_index, #t do
9447       local value = t[i]
9448       if value.is_closing and
9449         value.element == "link" and
9450         not value.removed then
9451         return i
9452       end
9453     end
9454   end
9455

```

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.

```

9456   local function disable_previous_link_openers(t, opening_index)
9457     if t[opening_index].element == "image" then
9458       return
9459     end
9460
9461     for i = opening_index, 1, -1 do
9462       local value = t[i]
9463       if value.is_active and
9464         value.type == "delimiter" and
9465         value.is_opening and
9466         value.element == "link" then
9467         value.is_active = false
9468       end
9469     end
9470   end
9471

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

9472   local function disable_range(t, opening_index, closing_index)
9473     for i = opening_index, closing_index do
9474       local value = t[i]
9475       if value.is_active then
9476         value.is_active = false
9477         if value.type == "delimiter" then
9478           value.removed = true
9479         end

```

```

9480     end
9481     end
9482 end
9483

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9484 local delete_parsed_content_in_range =
9485   function(t, opening_index, closing_index)
9486     for i = opening_index, closing_index do
9487       t[i].rendered = nil
9488     end
9489   end
9490

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9491 local function empty_content_in_range(t, opening_index, closing_index)
9492   for i = opening_index, closing_index do
9493     t[i].content = ''
9494   end
9495 end
9496

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

9497 local function join_attributes(reference_attributes, own_attributes)
9498   local merged_attributes = {}
9499   for _, attribute in ipairs(reference_attributes or {}) do
9500     table.insert(merged_attributes, attribute)
9501   end
9502   for _, attribute in ipairs(own_attributes or {}) do
9503     table.insert(merged_attributes, attribute)
9504   end
9505   if next(merged_attributes) == nil then
9506     merged_attributes = nil
9507   end
9508   return merged_attributes
9509 end
9510

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

9511 local render_link_or_image =
9512   function(t, opening_index, closing_index, content_end_index,
9513     reference)
9514     process_emphasis(t, opening_index, content_end_index)
9515     local mapped = collect_emphasis_content(t, opening_index + 1,

```

```

9516                                     content_end_index - 1)
9517
9518     local rendered = {}
9519     if (t[opening_index].element == "link") then
9520         rendered = writer.link(mapped, reference.url,
9521                               reference.title, reference.attributes)
9522     end
9523
9524     if (t[opening_index].element == "image") then
9525         rendered = writer.image(mapped, reference.url, reference.title,
9526                                reference.attributes)
9527     end
9528
9529     if (t[opening_index].element == "note") then
9530         if (t[opening_index].link_type == "note_inline") then
9531             rendered = writer.note(mapped)
9532         end
9533         if (t[opening_index].link_type == "raw_note") then
9534             rendered = writer.note(reference)
9535         end
9536     end
9537
9538     t[opening_index].rendered = rendered
9539     delete_parsed_content_in_range(t, opening_index + 1,
9540                                   closing_index)
9541     empty_content_in_range(t, opening_index, closing_index)
9542     disable_previous_link_openers(t, opening_index)
9543     disable_range(t, opening_index, closing_index)
9544 end
9545

```

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.

```

9546     local resolve_inline_following_content =
9547         function(t, closing_index, match_reference, match_link_attributes)
9548             local content = ""
9549             for i = closing_index + 1, #t do
9550                 content = content .. t[i].content
9551             end
9552
9553             local matching_content = parsers.succeed
9554
9555             if match_reference then
9556                 matching_content = matching_content
9557                     * parsers.inline_direct_ref_inside
9558             end

```

```

9559
9560     if match_link_attributes then
9561         matching_content = matching_content
9562             * Cg(Ct(parsers.attributes^-1), "attributes")
9563     end
9564
9565     local matched = lpeg.match(Ct( matching_content
9566                                     * Cg(Cp(), "end_position")), content)
9567
9568     local matched_count = matched.end_position - 1
9569     for i = closing_index + 1, #t do
9570         local value = t[i]
9571
9572         local chars_left = matched_count
9573         matched_count = matched_count - #value.content
9574
9575         if matched_count <= 0 then
9576             value.content = value.content:sub(chars_left + 1)
9577             break
9578         end
9579
9580         value.content = ''
9581         value.is_active = false
9582     end
9583
9584     local attributes = matched.attributes
9585     if attributes == nil or next(attributes) == nil then
9586         attributes = nil
9587     end
9588
9589     return {
9590         url = matched.url or "",
9591         title = matched.title or "",
9592         attributes = attributes
9593     }
9594 end
9595

```

Resolve an inline link `[a](b "c")` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Here, compared to other types of links, no reference definition is needed.

```

9596 local function resolve_inline_link(t, opening_index, closing_index)
9597     local inline_content
9598         = resolve_inline_following_content(t, closing_index, true,
9599   t.match_link_attributes)
9600     render_link_or_image(t, opening_index, closing_index,
9601                         closing_index, inline_content)

```



```
9602 end
9603
```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
9604 local function resolve_note_inline_link =
9605     function(t, opening_index, closing_index)
9606         local inline_content
9607             = resolve_inline_following_content(t, closing_index,
9608   false, false)
9609         render_link_or_image(t, opening_index, closing_index,
9610                             closing_index, inline_content)
9611     end
9612
```

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.

```
9613 local function resolve_shortcut_link(t, opening_index, closing_index)
9614     local content
9615         = collect_link_content(t, opening_index + 1, closing_index - 1)
9616     local r = self.lookup_reference(content)
9617
9618     if r then
9619         local inline_content
9620             = resolve_inline_following_content(t, closing_index, false,
9621   t.match_link_attributes)
9622         r.attributes
9623             = join_attributes(r.attributes, inline_content.attributes)
9624         render_link_or_image(t, opening_index, closing_index,
9625                             closing_index, r)
9626     end
9627 end
9628
```

Resolve a note `[^a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`. Continue if a tag `a` is not found in the rawnotes.

```
9629 local function resolve_raw_note_link(t, opening_index, closing_index)
9630     local content
9631         = collect_link_content(t, opening_index + 1, closing_index - 1)
9632     local r = self.lookup_note_reference(content)
9633
9634     if r then
9635         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9636         render_link_or_image(t, opening_index, closing_index,
9637                             closing_index, parsed_ref)
9638     end
9639 end
```

9640

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.

```
9641 local function resolve_full_link(t, opening_index, closing_index)
9642     local next_link_closing_index
9643         = find_next_link_closing_index(t, closing_index + 4)
9644     local next_link_content
9645         = collect_link_content(t, closing_index + 3,
9646                               next_link_closing_index - 1)
9647     local r = self.lookup_reference(next_link_content)
9648
9649     if r then
9650         local inline_content
9651             = resolve_inline_following_content(t, next_link_closing_index,
9652   false,
9653   t.match_link_attributes)
9654         r.attributes
9655             = join_attributes(r.attributes, inline_content.attributes)
9656         render_link_or_image(t, opening_index, next_link_closing_index,
9657                             closing_index, r)
9658     end
9659 end
9660
```

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.

```
9661 local function resolve_collapsed_link(t, opening_index, closing_index)
9662     local next_link_closing_index
9663         = find_next_link_closing_index(t, closing_index + 4)
9664     local content
9665         = collect_link_content(t, opening_index + 1, closing_index - 1)
9666     local r = self.lookup_reference(content)
9667
9668     if r then
9669         local inline_content
9670             = resolve_inline_following_content(t, closing_index, false,
9671   t.match_link_attributes)
9672         r.attributes
9673             = join_attributes(r.attributes, inline_content.attributes)
9674         render_link_or_image(t, opening_index, next_link_closing_index,
9675                             closing_index, r)
9676     end
9677 end
9678
```

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.

```

9679 local function process_links_and_emphasis(t)
9680   for _,value in ipairs(t) do
9681     value.is_active = true
9682   end
9683
9684   for i,value in ipairs(t) do
9685     if not value.is_closing
9686       or value.type ~= "delimiter"
9687       or not ( value.element == "link"
9688         or value.element == "image"
9689         or value.element == "note")
9690       or value.removed then
9691       goto continue
9692     end
9693
9694     local opener_position = find_link_opener(t, 1, i - 1)
9695     if (opener_position == nil) then
9696       goto continue
9697     end
9698
9699     local opening_delimiter = t[opener_position]
9700     opening_delimiter.removed = true
9701
9702     local link_type = opening_delimiter.link_type
9703
9704     if (link_type == "inline") then
9705       resolve_inline_link(t, opener_position, i)
9706     end
9707     if (link_type == "shortcut") then
9708       resolve_shortcut_link(t, opener_position, i)
9709     end
9710     if (link_type == "full") then
9711       resolve_full_link(t, opener_position, i)
9712     end
9713     if (link_type == "collapsed") then
9714       resolve_collapsed_link(t, opener_position, i)
9715     end
9716     if (link_type == "note_inline") then
9717       resolve_note_inline_link(t, opener_position, i)
9718     end
9719     if (link_type == "raw_note") then
9720       resolve_raw_note_link(t, opener_position, i)

```

```

9721     end
9722
9723     ::continue::
9724 end
9725
9726 t[#t].content = t[#t].content.gsub("%s*$", "")
9727
9728 process_emphasis(t, 1, #t)
9729 local final_result = collect_emphasis_content(t, 1, #t)
9730 return final_result
9731 end
9732
9733 function self.defer_link_and_emphasis_processing(delimiter_table)
9734     return writer.defer_call(function()
9735         return process_links_and_emphasis(delimiter_table)
9736     end)
9737 end
9738

```

### 3.1.6.8 Inline Elements (local)

```

9739 parsers.Str      = ( parsers.normalchar
9740                    * (parsers.normalchar + parsers.at)^0)
9741                  / writer.string
9742
9743 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
9744                  / writer.string
9745
9746 parsers.Ellipsis = P("...") / writer.ellipsis
9747
9748 parsers.Smart    = parsers.Ellipsis
9749
9750 parsers.Code     = parsers.inticks / writer.code
9751
9752 if options.blankBeforeBlockquote then
9753     parsers.bqstart = parsers.fail
9754 else
9755     parsers.bqstart = parsers.blockquote_start
9756 end
9757
9758 if options.blankBeforeHeading then
9759     parsers.headerstart = parsers.fail
9760 else
9761     parsers.headerstart = parsers.atx_heading
9762 end
9763
9764 if options.blankBeforeList then

```

```

9765     parsers.interrupting_bullets = parsers.fail
9766     parsers.interrupting_enumerators = parsers.fail
9767 else
9768     parsers.interrupting_bullets
9769     = parsers.bullet(parsers.dash, true)
9770     + parsers.bullet(parsers.asterisk, true)
9771     + parsers.bullet(parsers.plus, true)
9772
9773     parsers.interrupting_enumerators
9774     = parsers.enumerator(parsers.period, true)
9775     + parsers.enumerator(parsers.rparent, true)
9776 end
9777
9778 if options.html then
9779     parsers.html_interrupting
9780     = parsers.check_trail
9781     * ( parsers.html_incomplete_open_tag
9782       + parsers.html_incomplete_close_tag
9783       + parsers.html_incomplete_open_special_tag
9784       + parsers.html_comment_start
9785       + parsers.html_cdatasection_start
9786       + parsers.html_declaration_start
9787       + parsers.html_instruction_start
9788       - parsers.html_close_special_tag
9789       - parsers.html_empty_special_tag)
9790 else
9791     parsers.html_interrupting = parsers.fail
9792 end
9793
9794 parsers.ListStarter = parsers.starter
9795
9796 parsers.EndlineExceptions
9797     = parsers.blankline -- paragraph break
9798     + parsers.eof      -- end of document
9799     + parsers.bqstart
9800     + parsers.thematic_break_lines
9801     + parsers.interrupting_bullets
9802     + parsers.interrupting_enumerators
9803     + parsers.headerstart
9804     + parsers.html_interrupting
9805
9806 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9807
9808 parsers.endline = parsers.newline
9809     * (parsers.check_minimal_indent
9810       * -V("EndlineExceptions")
9811       + parsers.check_optional_indent

```

```

9812         * -V("EndlineExceptions")
9813         * -V("ListStarter")) / function(_) return end
9814     * parsers.spacechar^0
9815
9816     parsers.Endline = parsers.endline
9817         / writer.soft_line_break
9818
9819     parsers.EndlineNoSub = parsers.endline
9820
9821     parsers.NoSoftLineBreakEndline
9822         = parsers.newline
9823         * (parsers.check_minimal_indent
9824         * -V("NoSoftLineBreakEndlineExceptions")
9825         + parsers.check_optional_indent
9826         * -V("NoSoftLineBreakEndlineExceptions")
9827         * -V("ListStarter"))
9828         * parsers.spacechar^0
9829         / writer.space
9830
9831     parsers.EndlineBreak = parsers.backslash * parsers.endline
9832         / writer.hard_line_break
9833
9834     parsers.OptionalIndent
9835         = parsers.spacechar^1 / writer.space
9836
9837     parsers.Space = parsers.spacechar^2 * parsers.endline
9838         / writer.hard_line_break
9839         + parsers.spacechar^1
9840         * parsers.endline^-1
9841         * parsers.eof / self.expandtabs
9842         + parsers.spacechar^1 * parsers.endline
9843         / writer.soft_line_break
9844         + parsers.spacechar^1
9845         * -parsers.newline / self.expandtabs
9846         + parsers.spacechar^1
9847
9848     parsers.NoSoftLineBreakSpace
9849         = parsers.spacechar^2 * parsers.endline
9850         / writer.hard_line_break
9851         + parsers.spacechar^1
9852         * parsers.endline^-1
9853         * parsers.eof / self.expandtabs
9854         + parsers.spacechar^1 * parsers.endline
9855         / writer.soft_line_break
9856         + parsers.spacechar^1
9857         * -parsers.newline / self.expandtabs
9858         + parsers.spacechar^1

```

```

9859
9860 parsers.NonbreakingEndline
9861         = parsers.endline
9862         / writer.nbsp
9863
9864 parsers.NonbreakingSpace
9865         = parsers.spacechar^2 * parsers.endline
9866         / writer.nbsp
9867         + parsers.spacechar^1
9868         * parsers.endline^-1 * parsers.eof / ""
9869         + parsers.spacechar^1 * parsers.endline
9870         * parsers.optionalspace
9871         / writer.nbsp
9872         + parsers.spacechar^1 * parsers.optionalspace
9873         / writer.nbsp
9874

```

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.

```

9875 function self.auto_link_url(url, attributes)
9876   return writer.link(writer.escape(url),
9877                     url, nil, attributes)
9878 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.

```

9879 function self.auto_link_email(email, attributes)
9880   return writer.link(writer.escape(email),
9881                     "mailto:".email,
9882                     nil, attributes)
9883 end
9884
9885 parsers.AutoLinkUrl = parsers.auto_link_url
9886                     / self.auto_link_url
9887
9888 parsers.AutoLinkEmail
9889         = parsers.auto_link_email
9890         / self.auto_link_email
9891
9892 parsers.AutoLinkRelativeReference
9893         = parsers.auto_link_relative_reference
9894         / self.auto_link_url
9895
9896 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9897                     / self.defer_link_and_emphasis_processing

```

```

9898
9899 parsers.EscapedChar = parsers.backslash
9900     * C(parsers.escapable) / writer.string
9901
9902 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9903     / writer.inline_html_comment
9904     + Cs(parsers.html_any_empty_inline_tag
9905         + parsers.html_inline_instruction
9906         + parsers.html_inline_cdatasection
9907         + parsers.html_inline_declaration
9908         + parsers.html_any_open_inline_tag
9909         + parsers.html_any_close_tag)
9910     / writer.inline_html_tag
9911
9912 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

9913 parsers.DisplayHtml = Cs(parsers.check_trail
9914     * ( parsers.html_comment
9915         + parsers.html_special_block
9916         + parsers.html_block
9917         + parsers.html_any_block
9918         + parsers.html_instruction
9919         + parsers.html_cdatasection
9920         + parsers.html_declaration))
9921     / writer.block_html_element
9922
9923 parsers.indented_non_blank_line = parsers.indentedline
9924     - parsers.blankline
9925
9926 parsers.Verbatim
9927     = Cs( parsers.check_code_trail
9928         * (parsers.line - parsers.blankline)
9929         * (( parsers.check_minimal_blank_indent_and_full_code_trail
9930             * parsers.blankline)^0
9931           * ( (parsers.check_minimal_indent / "")
9932             * parsers.check_code_trail
9933             * (parsers.line - parsers.blankline))^1)^0)
9934     / self.expandtabs / writer.verbatim
9935
9936 parsers.Blockquote    = parsers.blockquote_body
9937     / writer.blockquote
9938
9939 parsers.ThematicBreak = parsers.thematic_break_lines
9940     / writer.thematic_break
9941

```



```

9942 parsers.Reference      = parsers.define_reference_parser
9943                          / self.register_link
9944
9945 parsers.Paragraph      = parsers.freeze_trail
9946                          * (Ct((parsers.Inline)^1)
9947                          * (parsers.newline + parsers.eof)
9948                          * parsers.unfreeze_trail
9949                          / writer.paragraph)
9950
9951 parsers.Plain          = parsers.nonindentspace * Ct(parsers.Inline^1)
9952                          / writer.plain

```

### 3.1.6.10 Lists (local)

```

9953
9954 if options.taskLists then
9955     parsers.tickbox = ( parsers.ticked_box
9956                       + parsers.halfticked_box
9957                       + parsers.unticked_box
9958                       ) / writer.tickbox
9959 else
9960     parsers.tickbox = parsers.fail
9961 end
9962
9963 parsers.list_blank = parsers.conditionally_indented_blankline
9964
9965 parsers.ref_or_block_list_separated
9966     = parsers.sep_group_no_output(parsers.list_blank)
9967     * parsers.minimally_indented_ref
9968     + parsers.block_sep_group(parsers.list_blank)
9969     * parsers.minimally_indented_block
9970
9971 parsers.ref_or_block_non_separated
9972     = parsers.minimally_indented_ref
9973     + (parsers.succeed / writer.interblocksep)
9974     * parsers.minimally_indented_block
9975     - parsers.minimally_indented_blankline
9976
9977 parsers.tight_list_loop_body_pair =
9978     parsers.create_loop_body_pair(
9979         parsers.ref_or_block_non_separated,
9980         parsers.minimally_indented_par_or_plain_no_blank,
9981         (parsers.succeed / writer.interblocksep),
9982         (parsers.succeed / writer.paragraphsep))
9983
9984 parsers.loose_list_loop_body_pair =
9985     parsers.create_loop_body_pair(

```

```

9986     parsers.ref_or_block_list_separated,
9987     parsers.minimally_indented_par_or_plain,
9988     parsers.block_sep_group(parsers.list_blank),
9989     parsers.par_sep_group(parsers.list_blank))
9990
9991 parsers.tight_list_content_loop
9992     = V("Block")
9993     * parsers.tight_list_loop_body_pair.block^0
9994     + (V("Paragraph") + V("Plain"))
9995     * parsers.ref_or_block_non_separated
9996     * parsers.tight_list_loop_body_pair.block^0
9997     + (V("Paragraph") + V("Plain"))
9998     * parsers.tight_list_loop_body_pair.par^0
9999
10000 parsers.loose_list_content_loop
10001     = V("Block")
10002     * parsers.loose_list_loop_body_pair.block^0
10003     + (V("Paragraph") + V("Plain"))
10004     * parsers.ref_or_block_list_separated
10005     * parsers.loose_list_loop_body_pair.block^0
10006     + (V("Paragraph") + V("Plain"))
10007     * parsers.loose_list_loop_body_pair.par^0
10008
10009 parsers.list_item_tightness_condition
10010     = -( parsers.list_blank^0
10011         * parsers.minimally_indented_ref_or_block_or_par)
10012     * remove_indent("li")
10013     + remove_indent("li")
10014     * parsers.fail
10015
10016 parsers.indented_content_tight
10017     = Ct( (parsers.blankline / "")
10018         * #parsers.list_blank
10019         * remove_indent("li")
10020         + ( (V("Reference") + (parsers.blankline / ""))
10021             * parsers.check_minimal_indent
10022             * parsers.tight_list_content_loop
10023             + (V("Reference") + (parsers.blankline / ""))
10024             + (parsers.tickbox^-1 / writer.escape)
10025             * parsers.tight_list_content_loop
10026             )
10027         * parsers.list_item_tightness_condition)
10028
10029 parsers.indented_content_loose
10030     = Ct( (parsers.blankline / "")
10031         * #parsers.list_blank
10032         + ( (V("Reference") + (parsers.blankline / ""))

```

```

10033     * parsers.check_minimal_indent
10034     * parsers.loose_list_content_loop
10035     + (V("Reference") + (parsers.blankline / ""))
10036     + (parsers.checkbox^-1 / writer.escape)
10037     * parsers.loose_list_content_loop))
10038
10039 parsers.TightListItem = function(starter)
10040     return -parsers.ThematicBreak
10041         * parsers.add_indent(starter, "li")
10042         * parsers.indented_content_tight
10043     end
10044
10045 parsers.LooseListItem = function(starter)
10046     return -parsers.ThematicBreak
10047         * parsers.add_indent(starter, "li")
10048         * parsers.indented_content_loose
10049         * remove_indent("li")
10050     end
10051
10052 parsers.BulletListOfType = function(bullet_type)
10053     local bullet = parsers.bullet(bullet_type)
10054     return ( Ct( parsers.TightListItem(bullet)
10055         * ( (parsers.check_minimal_indent / "")
10056         * parsers.TightListItem(bullet)
10057         )^0
10058     )
10059     * Cc(true)
10060     * -#( (parsers.list_blank^0 / "")
10061         * parsers.check_minimal_indent
10062         * (bullet - parsers.ThematicBreak)
10063     )
10064     + Ct( parsers.LooseListItem(bullet)
10065         * ( (parsers.list_blank^0 / "")
10066         * (parsers.check_minimal_indent / "")
10067         * parsers.LooseListItem(bullet)
10068         )^0
10069     )
10070     * Cc(false)
10071     ) / writer.bulletlist
10072     end
10073
10074 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10075     + parsers.BulletListOfType(parsers.asterisk)
10076     + parsers.BulletListOfType(parsers.plus)
10077
10078 local function ordered_list(items,tight,starter)
10079     local startnum = starter[2][1]

```

```

10080     if options.startNumber then
10081         startnum = tonumber(startnum) or 1 -- fallback for '#'
10082         if startnum ~= nil then
10083             startnum = math.floor(startnum)
10084         end
10085     else
10086         startnum = nil
10087     end
10088     return writer.orderedlist(items,tight,startnum)
10089 end
10090
10091 parsers.OrderedListOfType = function(delimiter_type)
10092     local enumerator = parsers.enumerator(delimiter_type)
10093     return Cg(enumerator, "listtype")
10094         * (Ct( parsers.TightListItem(Cb("listtype"))
10095             * ( (parsers.check_minimal_indent / "")
10096                 * parsers.TightListItem(enumerator))^0)
10097             * Cc(true)
10098             * -#((parsers.list_blank^0 / "")
10099                 * parsers.check_minimal_indent * enumerator)
10100         + Ct( parsers.LooseListItem(Cb("listtype"))
10101             * ((parsers.list_blank^0 / "")
10102                 * (parsers.check_minimal_indent / "")
10103                 * parsers.LooseListItem(enumerator))^0)
10104             * Cc(false)
10105         ) * Ct(Cb("listtype")) / ordered_list
10106 end
10107
10108 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10109     + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

10110 parsers.Blank           = parsers.blankline / ""
10111                         + V("Reference")

```

### 3.1.6.12 Headings (local)

```

10112 function parsers.parse_heading_text(s)
10113     local inlines = self.parser_functions.parse_inlines(s)
10114     local flatten_inlines = self.writer.flatten_inlines
10115     self.writer.flatten_inlines = true
10116     local flat_text = self.parser_functions.parse_inlines(s)
10117     flat_text = util.rope_to_string(flat_text)
10118     self.writer.flatten_inlines = flatten_inlines
10119     return {flat_text, inlines}
10120 end
10121

```

```

10122 -- parse atx header
10123 parsers.AtxHeading = parsers.check_trail_no_rem
10124                      * Cg(parsers.heading_start, "level")
10125                      * ((C( parsers.optionalspace
10126                          * parsers.hash^0
10127                          * parsers.optionalspace
10128                          * parsers.newline)
10129                          + parsers.spacechar^1
10130                          * C(parsers.line))
10131                      / strip_atx_end
10132                      / parsers.parse_heading_text)
10133                      * Cb("level")
10134                      / writer.heading
10135
10136 parsers.heading_line = parsers.linechar^1
10137                      - parsers.thematic_break_lines
10138
10139 parsers.heading_text = parsers.heading_line
10140                      * ( (V("Endline") / "\n")
10141                      * ( parsers.heading_line
10142                        - parsers.heading_level))^0
10143                      * parsers.newline^-1
10144
10145 parsers.SetextHeading = parsers.freeze_trail
10146                      * parsers.check_trail_no_rem
10147                      * #( parsers.heading_text
10148                          * parsers.check_minimal_indent
10149                          * parsers.check_trail
10150                          * parsers.heading_level)
10151                      * Cs(parsers.heading_text)
10152                      / parsers.parse_heading_text
10153                      * parsers.check_minimal_indent_and_trail
10154                      * parsers.heading_level
10155                      * parsers.newline
10156                      * parsers.unfreeze_trail
10157                      / writer.heading
10158
10159 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  $\text{T}_{\text{E}}\text{X}$  output.

```
10160 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.

```

10161     local walkable_syntax = (function(global_walkable_syntax)
10162         local local_walkable_syntax = {}
10163         for lhs, rule in pairs(global_walkable_syntax) do
10164             local_walkable_syntax[lhs] = util.table_copy(rule)
10165         end
10166         return local_walkable_syntax
10167     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.

```

10168     local current_extension_name = nil
10169     self.insert_pattern = function(selector, pattern, pattern_name)
10170         assert(pattern_name == nil or type(pattern_name) == "string")
10171         local _, _, lhs, pos, rhs
10172             = selector:find("^(%a+)%s+([%a%s]+%a)%s+(%a+)$")
10173         assert(lhs ~= nil,
10174             [[Expected selector in form ]]
10175             .. [[ "LHS (before|after|instead of) RHS", not "]]
10176             .. selector .. [[ "]])
10177         assert(walkable_syntax[lhs] ~= nil,
10178             [[Rule ]] .. lhs
10179             .. [[ -> ... does not exist in markdown grammar]])
10180         assert(pos == "before" or pos == "after" or pos == "instead of",
10181             [[Expected positional specifier "before", "after", ]]
10182             .. [[or "instead of", not "]]
10183             .. pos .. [[ "]])
10184         local rule = walkable_syntax[lhs]
10185         local index = nil
10186         for current_index, current_rhs in ipairs(rule) do
10187             if type(current_rhs) == "string" and current_rhs == rhs then
10188                 index = current_index
10189                 if pos == "after" then
10190                     index = index + 1
10191                 end
10192                 break
10193             end
10194         end
10195         assert(index ~= nil,
10196             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
10197             .. [[ does not exist in markdown grammar]])
10198         local accountable_pattern
10199         if current_extension_name then
10200             accountable_pattern

```

```

10201         = {pattern, current_extension_name, pattern_name}
10202     else
10203         assert(type(pattern) == "string",
10204             [[reader->insert_pattern() was called outside ]]
10205             .. [[an extension with ]]
10206             .. [[a PEG pattern instead of a rule name]])
10207         accountable_pattern = pattern
10208     end
10209     if pos == "instead of" then
10210         rule[index] = accountable_pattern
10211     else
10212         table.insert(rule, index, accountable_pattern)
10213     end
10214 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.

```

10215     local syntax =
10216         { "Blocks",
10217
10218           Blocks = V("InitializeState")
10219                 * V("ExpectedJekyllData")
10220                 * 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.

```

10221         * ( V("Block")
10222           * ( V("Blank")^0 * parsers.eof
10223             + ( V("Blank")^2 / writer.paragraphsep
10224               + V("Blank")^0 / writer.interblocksep
10225             )
10226           )
10227         + ( V("Paragraph") + V("Plain") )
10228         * ( V("Blank")^0 * parsers.eof
10229           + ( V("Blank")^2 / writer.paragraphsep
10230             + V("Blank")^0 / writer.interblocksep
10231           )
10232         )
10233         * V("Block")
10234         * ( V("Blank")^0 * parsers.eof
10235           + ( V("Blank")^2 / writer.paragraphsep
10236             + V("Blank")^0 / writer.interblocksep
10237           )
10238         )
10239         + ( V("Paragraph") + V("Plain") )
10240         * ( V("Blank")^0 * parsers.eof
10241           + V("Blank")^0 / writer.paragraphsep

```

```

10242         )
10243     )~0,
10244
10245     ExpectedJekyllData = parsers.succeed,
10246
10247     Blank                = parsers.Blank,
10248     Reference            = parsers.Reference,
10249
10250     Blockquote           = parsers.Blockquote,
10251     Verbatim             = parsers.Verbatim,
10252     ThematicBreak        = parsers.ThematicBreak,
10253     BulletList           = parsers.BulletList,
10254     OrderedList          = parsers.OrderedList,
10255     DisplayHtml          = parsers.DisplayHtml,
10256     Heading              = parsers.Heading,
10257     Paragraph            = parsers.Paragraph,
10258     Plain                = parsers.Plain,
10259
10260     ListStarter          = parsers.ListStarter,
10261     EndlineExceptions    = parsers.EndlineExceptions,
10262     NoSoftLineBreakEndlineExceptions
10263         = parsers.NoSoftLineBreakEndlineExceptions,
10264
10265     Str                  = parsers.Str,
10266     Space                = parsers.Space,
10267     NoSoftLineBreakSpace
10268         = parsers.NoSoftLineBreakSpace,
10269     OptionalIndent      = parsers.OptionalIndent,
10270     Endline              = parsers.Endline,
10271     EndlineNoSub         = parsers.EndlineNoSub,
10272     NoSoftLineBreakEndline
10273         = parsers.NoSoftLineBreakEndline,
10274     EndlineBreak        = parsers.EndlineBreak,
10275     LinkAndEmph         = parsers.LinkAndEmph,
10276     Code                 = parsers.Code,
10277     AutoLinkUrl          = parsers.AutoLinkUrl,
10278     AutoLinkEmail        = parsers.AutoLinkEmail,
10279     AutoLinkRelativeReference
10280         = parsers.AutoLinkRelativeReference,
10281     InlineHtml           = parsers.InlineHtml,
10282     HtmlEntity           = parsers.HtmlEntity,
10283     EscapedChar          = parsers.EscapedChar,
10284     Smart                = parsers.Smart,
10285     Symbol               = parsers.Symbol,
10286     SpecialChar          = parsers.fail,
10287     InitializeState      = parsers.succeed,
10288 }

```



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].

```

10289     self.update_rule = function(rule_name, get_pattern)
10290         assert(current_extension_name ~= nil)
10291         assert(syntax[rule_name] ~= nil,
10292             [[Rule ]] .. rule_name
10293             .. [[ -> ... does not exist in markdown grammar]])
10294         local previous_pattern
10295         local extension_name
10296         if walkable_syntax[rule_name] then
10297             local previous_accountable_pattern
10298             = walkable_syntax[rule_name][1]
10299             previous_pattern = previous_accountable_pattern[1]
10300             extension_name
10301             = previous_accountable_pattern[2]
10302             .. ", " .. current_extension_name
10303         else
10304             previous_pattern = nil
10305             extension_name = current_extension_name
10306         end
10307         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

```

```

10308     if type(get_pattern) == "function" then
10309         pattern = get_pattern(previous_pattern)
10310     else
10311         assert(previous_pattern == nil,
10312             [[Rule ]] .. rule_name ..
10313             [[ has already been updated by ]] .. extension_name)
10314         pattern = get_pattern
10315     end
10316     local accountable_pattern = { pattern, extension_name, rule_name }
10317     walkable_syntax[rule_name] = { accountable_pattern }
10318 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.

```
10319     local special_characters = {}
10320     self.add_special_character = function(c)
10321         table.insert(special_characters, c)
10322         syntax.SpecialChar = S(table.concat(special_characters, ""))
10323     end
10324
10325     self.add_special_character("*")
10326     self.add_special_character("[")
10327     self.add_special_character("]")
10328     self.add_special_character("<")
10329     self.add_special_character("!")
10330     self.add_special_character("\\")
```

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

```
10331     self.initialize_named_group = function(name, value)
10332         local pattern = Ct("")
10333         if value ~= nil then
10334             pattern = pattern / value
10335         end
10336         syntax.InitializeState = syntax.InitializeState
10337                                 * Cg(pattern, name)
10338     end
```

Add a named group for indentation.

```
10339     self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
10340     for _, extension in ipairs(extensions) do
10341         current_extension_name = extension.name
10342         extension.extend_writer(writer)
10343         extension.extend_reader(self)
10344     end
10345     current_extension_name = nil
```

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

```
10346     if options.debugExtensions then
10347         local sorted_lhs = {}
10348         for lhs, _ in pairs(walkable_syntax) do
10349             table.insert(sorted_lhs, lhs)
10350         end
10351         table.sort(sorted_lhs)
10352
10353         local output_lines = {"{"}
```

```

10354     for lhs_index, lhs in ipairs(sorted_lhs) do
10355         local encoded_lhs = util.encode_json_string(lhs)
10356         table.insert(output_lines, [[      ]] .. encoded_lhs .. [[: ]])
10357         local rule = walkable_syntax[lhs]
10358         for rhs_index, rhs in ipairs(rule) do
10359             local human_readable_rhs
10360             if type(rhs) == "string" then
10361                 human_readable_rhs = rhs
10362             else
10363                 local pattern_name
10364                 if rhs[3] then
10365                     pattern_name = rhs[3]
10366                 else
10367                     pattern_name = "Anonymous Pattern"
10368                 end
10369                 local extension_name = rhs[2]
10370                 human_readable_rhs = pattern_name .. [[ (]]
10371                     .. extension_name .. [[)]]
10372             end
10373             local encoded_rhs
10374                 = util.encode_json_string(human_readable_rhs)
10375             local output_line = [[      ]] .. encoded_rhs
10376             if rhs_index < #rule then
10377                 output_line = output_line .. ", "
10378             end
10379             table.insert(output_lines, output_line)
10380         end
10381         local output_line = "    ]"
10382         if lhs_index < #sorted_lhs then
10383             output_line = output_line .. ", "
10384         end
10385         table.insert(output_lines, output_line)
10386     end
10387     table.insert(output_lines, "}")
10388
10389     local output = table.concat(output_lines, "\n")
10390     local output_filename = options.debugExtensionsFileName
10391     local output_file = assert(io.open(output_filename, "w"),
10392         [[Could not open file ]] .. output_filename
10393         .. [[ for writing]])
10394     assert(output_file:write(output))
10395     assert(output_file:close())
10396 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`.

```

10397     for lhs, rule in pairs(walkable_syntax) do
10398         syntax[lhs] = parsers.fail
10399         for _, rhs in ipairs(rule) do
10400             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.

```

10401         if type(rhs) == "string" then
10402             pattern = V(rhs)
10403         else
10404             pattern = rhs[1]
10405             if type(pattern) == "string" then
10406                 pattern = V(pattern)
10407             end
10408         end
10409         syntax[lhs] = syntax[lhs] + pattern
10410     end
10411 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.

```

10412     if options.underscores then
10413         self.add_special_character("_")
10414     end
10415
10416     if not options.codeSpans then
10417         syntax.Code = parsers.fail
10418     else
10419         self.add_special_character("`")
10420     end
10421
10422     if not options.html then
10423         syntax.DisplayHtml = parsers.fail
10424         syntax.InlineHtml = parsers.fail
10425         syntax.HtmlEntity = parsers.fail
10426     else
10427         self.add_special_character("&")
10428     end
10429
10430     if options.preserveTabs then
10431         options.stripIndent = false
10432     end
10433
10434     if not options.smartEllipses then

```

```

10435     syntax.Smart = parsers.fail
10436 else
10437     self.add_special_character(".")
10438 end
10439
10440 if not options.relativeReferences then
10441     syntax.AutoLinkRelativeReference = parsers.fail
10442 end
10443
10444 if options.contentLevel == "inline" then
10445     syntax[1] = "Inlines"
10446     syntax.Inlines = V("InitializeState")
10447         * parsers.Inline^0
10448         * ( parsers.spacing^0
10449             * parsers.eof / "" )
10450     syntax.Space = parsers.Space + parsers.blankline / writer.space
10451 end
10452
10453 local blocks_nested_t = util.table_copy(syntax)
10454 blocks_nested_t.ExpectedJekyllData = parsers.succeed
10455 parsers.blocks_nested = Ct(blocks_nested_t)
10456
10457 parsers.blocks = Ct(syntax)
10458
10459 local inlines_t = util.table_copy(syntax)
10460 inlines_t[1] = "Inlines"
10461 inlines_t.Inlines = V("InitializeState")
10462     * parsers.Inline^0
10463     * ( parsers.spacing^0
10464         * parsers.eof / "" )
10465 parsers.inlines = Ct(inlines_t)
10466
10467 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10468 inlines_no_inline_note_t.InlineNote = parsers.fail
10469 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10470
10471 local inlines_no_html_t = util.table_copy(inlines_t)
10472 inlines_no_html_t.DisplayHtml = parsers.fail
10473 inlines_no_html_t.InlineHtml = parsers.fail
10474 inlines_no_html_t.HtmlEntity = parsers.fail
10475 parsers.inlines_no_html = Ct(inlines_no_html_t)
10476
10477 local inlines_nbsp_t = util.table_copy(inlines_t)
10478 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10479 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10480 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10481

```

```

10482     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10483     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10484     inlines_no_link_or_emphasis_t.EndlineExceptions
10485     = parsers.EndlineExceptions - parsers.eof
10486     parsers.inlines_no_link_or_emphasis
10487     = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain TeX output and returns it..

```

10488     return function(input)
Unicode-normalize the input.
10489         if options.unicodeNormalization then
10490             local form = options.unicodeNormalizationForm
10491             if form == "nfc" then
10492                 input = uni_algos.normalize.NFC(input)
10493             elseif form == "nfd" then
10494                 input = uni_algos.normalize.NFD(input)
10495             elseif form == "nfkc" then
10496                 input = uni_algos.normalize.NFKC(input)
10497             elseif form == "nfkd" then
10498                 input = uni_algos.normalize.NFKD(input)
10499             else
10500                 return writer.error(
10501                     format("Unknown normalization form %s.", form))
10502             end
10503         end

```

Since the Lua converter expects UNIX line endings, normalize the input. Also add a line ending at the end of the file in case the input file has none.

```

10504         input = input:gsub("\r\n?", "\n")
10505         if input:sub(-1) ~= "\n" then
10506             input = input .. "\n"
10507         end

```

Clear the table of references.

```

10508         references = {}
10509         local document = self.parser_functions.parse_blocks(input)
10510         local output = util.ropo_to_string(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`.

```

10511         local undosep_start, undosep_end
10512         local potential_secend_start, secend_start
10513         local potential_sep_start, sep_start
10514         while true do
10515             -- find a `writer->undosep`
10516             undosep_start, undosep_end

```

```

10517         = output:find(writer.undosep_text, 1, true)
10518     if undosep_start == nil then break end
10519     -- skip any preceding section ends
10520     secend_start = undosep_start
10521     while true do
10522         potential_secend_start = secend_start - #writer.secend_text
10523         if potential_secend_start < 1
10524             or output:sub(potential_secend_start,
10525                 secend_start - 1) ~= writer.secend_text
10526             then
10527                 break
10528             end
10529             secend_start = potential_secend_start
10530         end
10531         -- find an immediately preceding
10532         -- block element / paragraph separator
10533         sep_start = secend_start
10534         potential_sep_start = sep_start - #writer.interblocksep_text
10535         if potential_sep_start >= 1
10536             and output:sub(potential_sep_start,
10537                 sep_start - 1) == writer.interblocksep_text
10538             then
10539                 sep_start = potential_sep_start
10540             else
10541                 potential_sep_start = sep_start - #writer.paragraphsep_text
10542                 if potential_sep_start >= 1
10543                     and output:sub(potential_sep_start,
10544                         sep_start - 1) == writer.paragraphsep_text
10545                     then
10546                         sep_start = potential_sep_start
10547                     end
10548                 end
10549                 -- remove `writer->undosep` and immediately preceding
10550                 -- block element / paragraph separator
10551                 output = output:sub(1, sep_start - 1)
10552                 .. output:sub(secend_start, undosep_start - 1)
10553                 .. output:sub(undosep_end + 1)
10554             end
10555             return output
10556         end
10557     end
10558     return self
10559 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.

```
10560 M.extensions = {}
```

### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
10561 M.extensions.bracketed_spans = function()
10562   return {
10563     name = "built-in bracketed_spans syntax extension",
10564     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.

```
10565     function self.span(s, attr)
10566       if self.flatten_inlines then return s end
10567       return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10568             self.attributes(attr),
10569             s,
10570             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10571     end
10572   end, extend_reader = function(self)
10573     local parsers = self.parsers
10574     local writer = self.writer
10575
10576     local span_label = parsers.lbracket
10577                       * (Cs((parsers.alphanumeric^1
10578                             + parsers.inticks
10579                             + parsers.autolink
10580                             + V("InlineHtml")
10581                             + ( parsers.backslash * parsers.backslash)
10582                             + ( parsers.backslash
10583                               * (parsers.lbracket + parsers.rbracket)
10584                               + V("Space") + V("Endline")
10585                               + (parsers.any
10586                                 - ( parsers.newline
10587                                   + parsers.lbracket
10588                                   + parsers.rbracket
10589                                   + parsers.blankline^2))))^1)
10590                       / self.parser_functions.parse_inlines)
10591                       * parsers.rbracket
10592
10593     local Span = span_label
10594               * Ct(parsers.attributes)
```



```

10595             / writer.span
10596
10597         self.insert_pattern("Inline before LinkAndEmph",
10598                             Span, "Span")
10599     end
10600 }
10601 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.

```

10602 M.extensions.citations = function(citation_nbsps)
10603   return {
10604     name = "built-in citations syntax extension",
10605     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.

```

10606     function self.citations(text_cites, cites)
10607       local buffer = {}
10608       if self.flatten_inlines then
10609         for _,cite in ipairs(cites) do
10610           if cite.prenote then
10611             table.insert(buffer, {cite.prenote, " "})
10612           end
10613           table.insert(buffer, cite.name)
10614           if cite.postnote then
10615             table.insert(buffer, {" ", cite.postnote})
10616           end
10617         end
10618       else

```

```

10619         table.insert(buffer,
10620             {"\\markdownRenderer",
10621             text_cites and "TextCite" or "Cite",
10622             "{", #cites, "}"}))
10623     for _,cite in ipairs(cites) do
10624         table.insert(buffer,
10625             {cite.suppress_author and "-" or "+", "{",
10626             cite.prenote or "", "}{" ,
10627             cite.postnote or "", "}{" , cite.name, "}"}))
10628     end
10629     end
10630     return buffer
10631 end
10632 end, extend_reader = function(self)
10633     local parsers = self.parsers
10634     local writer = self.writer
10635
10636     local citation_chars
10637         = parsers.alphanumeric
10638         + S("#$%&-+<>~/_")
10639
10640     local citation_name
10641         = Cs(parsers.dash~-1) * parsers.at
10642         * Cs(citation_chars
10643             * ((( citation_chars
10644                 + parsers.internal_punctuation
10645                 - parsers.comma - parsers.semicolon)
10646             * -#(( parsers.internal_punctuation
10647                 - parsers.comma
10648                 - parsers.semicolon)^0
10649             * -( citation_chars
10650                 + parsers.internal_punctuation
10651                 - parsers.comma
10652                 - parsers.semicolon)))^0
10653             * citation_chars)^-1)
10654
10655     local citation_body_prenote
10656         = Cs((parsers.alphanumeric^1
10657             + parsers.bracketed
10658             + parsers.inticks
10659             + parsers.autolink
10660             + V("InlineHtml")
10661             + V("Space") + V("EndlineNoSub")
10662             + (parsers.anyescaped
10663             - ( parsers.newline
10664             + parsers.rbracket
10665             + parsers.blankline^2))

```

```

10666         - ( parsers.spnl
10667           * parsers.dash^-1
10668           * parsers.at))^1)
10669
10670     local citation_body_postnote
10671       = Cs((parsers.alphanumeric^1
10672         + parsers.bracketed
10673         + parsers.inticks
10674         + parsers.autolink
10675         + V("InlineHtml")
10676         + V("Space") + V("EndlineNoSub"))
10677         + (parsers.anyescaped
10678           - ( parsers.newline
10679             + parsers.rbracket
10680             + parsers.semicolon
10681             + parsers.blankline^2))
10682         - (parsers.spnl * parsers.rbracket))^1)
10683
10684     local citation_body_chunk
10685       = ( citation_body_prenote
10686         * parsers.spnlc_sep
10687         + Cc("")
10688         * parsers.spnlc
10689       )
10690       * citation_name
10691       * ( parsers.internal_punctuation
10692         - parsers.semicolon)^-1
10693       * ( parsers.spnlc / function(_) return end
10694         * citation_body_postnote
10695         + Cc("")
10696         * parsers.spnlc
10697       )
10698
10699     local citation_body
10700       = citation_body_chunk
10701       * ( parsers.semicolon
10702         * parsers.spnlc
10703         * citation_body_chunk
10704       )^0
10705
10706     local citation_headless_body_postnote
10707       = Cs((parsers.alphanumeric^1
10708         + parsers.bracketed
10709         + parsers.inticks
10710         + parsers.autolink
10711         + V("InlineHtml")
10712         + V("Space") + V("Endline"))

```

```

10713         + (parsers.anyescaped
10714         - ( parsers.newline
10715         + parsers.rbracket
10716         + parsers.at
10717         + parsers.semicolon + parsers.blankline^2))
10718         - (parsers.spnl * parsers.rbracket))^0)
10719
10720     local citation_headless_body
10721         = citation_headless_body_postnote
10722         * ( parsers.semicolon
10723         * parsers.spnlc
10724         * citation_body_chunk
10725         )^0
10726
10727     local citations
10728         = function(text_cites, raw_cites)
10729         local function normalize(str)
10730             if str == "" then
10731                 str = nil
10732             else
10733                 str = (citation_nbsps and
10734                 self.parser_functions.parse_inlines_nbsp or
10735                 self.parser_functions.parse_inlines)(str)
10736             end
10737             return str
10738         end
10739
10740         local cites = {}
10741         for i = 1,#raw_cites,4 do
10742             cites[#cites+1] = {
10743                 prenote = normalize(raw_cites[i]),
10744                 suppress_author = raw_cites[i+1] == "-",
10745                 name = writer.identifier(raw_cites[i+2]),
10746                 postnote = normalize(raw_cites[i+3]),
10747             }
10748         end
10749         return writer.citations(text_cites, cites)
10750     end
10751
10752     local TextCitations
10753         = Ct((parsers.spnlc
10754         * Cc("")
10755         * citation_name
10756         * ((parsers.spnlc
10757         * parsers.lbracket
10758         * citation_headless_body
10759         * parsers.rbracket) + Cc("")))^1)

```

```

10760         / function(raw_cites)
10761             return citations(true, raw_cites)
10762         end
10763
10764     local ParenthesizedCitations
10765         = Ct((parsers.spnlc
10766             * parsers.lbracket
10767             * citation_body
10768             * parsers.rbracket)^1)
10769     / function(raw_cites)
10770         return citations(false, raw_cites)
10771     end
10772
10773     local Citations = TextCitations + ParenthesizedCitations
10774
10775     self.insert_pattern("Inline before LinkAndEmph",
10776                       Citations, "Citations")
10777
10778     self.add_special_character("@")
10779     self.add_special_character("-")
10780 end
10781 }
10782 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.

```

10783 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.

```

10784 local languages_json = (function()
10785     local base, prev, curr
10786     for _, pathname in ipairs{kpse.lookup(language_map,
10787   {all=true})} do
10788         local file = io.open(pathname, "r")
10789         if not file then goto continue end
10790         local input = assert(file:read("*a"))
10791         assert(file:close())
10792         local json = input:gsub('("[^\\n]-"):','[%1]=')
10793         curr = load("_ENV = {}; return "..json")()
10794         if type(curr) == "table" then
10795             if base == nil then

```

```

10796         base = curr
10797     else
10798         setmetatable(prev, { __index = curr })
10799     end
10800     prev = curr
10801 end
10802 ::continue::
10803 end
10804 return base or {}
10805 end)()
10806
10807 return {
10808     name = "built-in content_blocks syntax extension",
10809     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.

```

10810     function self.contentblock(src,suf,type,tit)
10811         if not self.is_writing then return "" end
10812         src = src.." "..suf
10813         suf = suf:lower()
10814         if type == "onlineimage" then
10815             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10816                 "{" ,self.string(src),"} ",
10817                 "{" ,self.uri(src),"} ",
10818                 "{" ,self.string(tit or ""),"} "}
10819         elseif languages_json[suf] then
10820             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10821                 "{" ,self.string(languages_json[suf]),"} ",
10822                 "{" ,self.string(src),"} ",
10823                 "{" ,self.uri(src),"} ",
10824                 "{" ,self.string(tit or ""),"} "}
10825         else
10826             return {"\\markdownRendererContentBlock{" ,suf,"} ",
10827                 "{" ,self.string(src),"} ",
10828                 "{" ,self.uri(src),"} ",
10829                 "{" ,self.string(tit or ""),"} "}
10830         end
10831     end
10832 end, extend_reader = function(self)
10833     local parsers = self.parsers
10834     local writer = self.writer
10835
10836     local contentblock_tail
10837         = parsers.optionaltitle

```

```

10838             * (parsers.newline + parsers.eof)
10839
10840 -- case insensitive online image suffix:
10841 local onlineimagesuffix
10842     = (function(...)
10843         local parser = nil
10844         for _, suffix in ipairs({...}) do
10845             local pattern=nil
10846             for i=1,#suffix do
10847                 local char=suffix:sub(i,i)
10848                 char = S(char:lower()..char:upper())
10849                 if pattern == nil then
10850                     pattern = char
10851                 else
10852                     pattern = pattern * char
10853                 end
10854             end
10855             if parser == nil then
10856                 parser = pattern
10857             else
10858                 parser = parser + pattern
10859             end
10860         end
10861         return parser
10862     end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10863
10864 -- online image url for iA Writer content blocks with
10865 -- mandatory suffix, allowing nested brackets:
10866 local onlineimageurl
10867     = (parsers.less
10868         * Cs((parsers.anyescaped
10869             - parsers.more
10870             - parsers.spacing
10871             - #(parsers.period
10872                 * onlineimagesuffix
10873                 * parsers.more
10874                 * contentblock_tail))^0)
10875         * parsers.period
10876         * Cs(onlineimagesuffix)
10877         * parsers.more
10878         + (Cs((parsers.inparens
10879             + (parsers.anyescaped
10880                 - parsers.spacing
10881                 - parsers.rparent
10882                 - #(parsers.period
10883                     * onlineimagesuffix
10884                     * contentblock_tail))))^0)

```

```

10885         * parsers.period
10886         * Cs(onlineimagesuffix))
10887     ) * Cc("onlineimage")
10888
10889     -- filename for iA Writer content blocks with mandatory suffix:
10890     local localfilepath
10891         = parsers.slash
10892         * Cs((parsers.anyescaped
10893             - parsers.tab
10894             - parsers.newline
10895             - #(parsers.period
10896                 * parsers.alphanumeric^1
10897                 * contentblock_tail))^1)
10898         * parsers.period
10899         * Cs(parsers.alphanumeric^1)
10900         * Cc("localfile")
10901
10902     local ContentBlock
10903         = parsers.check_trail_no_rem
10904         * (localfilepath + onlineimageurl)
10905         * contentblock_tail
10906         / writer.contentblock
10907
10908     self.insert_pattern("Block before Blockquote",
10909                       ContentBlock, "ContentBlock")
10910 end
10911 }
10912 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.

```

10913 M.extensions.definition_lists = function(tight_lists)
10914   return {
10915     name = "built-in definition_lists syntax extension",
10916     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.

```

10917     local function dlitem(term, defs)
10918       local retVal = {"\\markdownRendererDlItem{",term,""}
10919       for _, def in ipairs(defs) do
10920         retVal[#retVal+1]

```



```

10921         = {"\\markdownRendererDlDefinitionBegin ",def,
10922             "\\markdownRendererDlDefinitionEnd "}
10923     end
10924     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10925     return retVal
10926 end
10927
10928 function self.definitionlist(items,tight)
10929     if not self.is_writing then return "" end
10930     local buffer = {}
10931     for _,item in ipairs(items) do
10932         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
10933     end
10934     if tight and tight_lists then
10935         return {"\\markdownRendererDlBeginTight\n", buffer,
10936             "\n\\markdownRendererDlEndTight"}
10937     else
10938         return {"\\markdownRendererDlBegin\n", buffer,
10939             "\n\\markdownRendererDlEnd"}
10940     end
10941 end
10942 end, extend_reader = function(self)
10943     local parsers = self.parsers
10944     local writer = self.writer
10945
10946     local defstartchar = S("~:")
10947
10948     local defstart
10949     = parsers.check_trail_length(0) * defstartchar
10950     * #parsers.spacing
10951     * (parsers.tab + parsers.space^-3)
10952     + parsers.check_trail_length(1)
10953     * defstartchar * #parsers.spacing
10954     * (parsers.tab + parsers.space^-2)
10955     + parsers.check_trail_length(2)
10956     * defstartchar * #parsers.spacing
10957     * (parsers.tab + parsers.space^-1)
10958     + parsers.check_trail_length(3)
10959     * defstartchar * #parsers.spacing
10960
10961     local indented_line
10962     = (parsers.check_minimal_indent / "")
10963     * parsers.check_code_trail * parsers.line
10964
10965     local blank
10966     = parsers.check_minimal_blank_indent_and_any_trail
10967     * parsers.optionalspace * parsers.newline

```

```

10968
10969     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10970
10971     local indented_blocks = function(bl)
10972         return Cs( bl
10973             * (blank^1 * (parsers.check_minimal_indent / ""))
10974             * parsers.check_code_trail * -parsers.blankline * bl)^0
10975             * (blank^1 + parsers.eof))
10976     end
10977
10978     local function definition_list_item(term, defs, _)
10979         return { term = self.parser_functions.parse_inlines(term),
10980             definitions = defs }
10981     end
10982
10983     local DefinitionListItemLoose
10984     = C(parsers.line) * blank^0
10985     * Ct((parsers.check_minimal_indent * (defstart
10986         * indented_blocks(dlchunk)
10987         / self.parser_functions.parse_blocks_nested))^1)
10988     * Cc(false) / definition_list_item
10989
10990     local DefinitionListItemTight
10991     = C(parsers.line)
10992     * Ct((parsers.check_minimal_indent * (defstart * dlchunk
10993         / self.parser_functions.parse_blocks_nested))^1)
10994     * Cc(true) / definition_list_item
10995
10996     local DefinitionList
10997     = ( Ct(DefinitionListItemLoose^1) * Cc(false)
10998     + Ct(DefinitionListItemTight^1)
10999     * (blank^0
11000     * -DefinitionListItemLoose * Cc(true))
11001     ) / writer.definitionlist
11002
11003     self.insert_pattern("Block after Heading",
11004         DefinitionList, "DefinitionList")
11005     end
11006 }
11007 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

11008 M.extensions.fancy_lists = function()
11009     return {

```

```

11010     name = "built-in fancy_lists syntax extension",
11011     extend_writer = function(self)
11012         local options = self.options
11013

```

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.

```

11014     function self.fancylist(items,tight,startnum,numstyle,numdelim)
11015         if not self.is_writing then return "" end
11016         local buffer = {}
11017         local num = startnum
11018         for _,item in ipairs(items) do
11019             if item ~= "" then
11020                 buffer[#buffer + 1] = self.fancyitem(item,num)
11021             end
11022             if num ~= nil and item ~= "" then
11023                 num = num + 1
11024             end
11025         end
11026         local contents = util.intersperse(buffer,"\n")
11027         if tight and options.tightLists then
11028             return {"\markdownRendererFancyOlBeginTight{",
11029                 numstyle,"}{",numdelim,"}",contents,
11030                 "\n\markdownRendererFancyOlEndTight "}
11031         else
11032             return {"\markdownRendererFancyOlBegin{",
11033                 numstyle,"}{",numdelim,"}",contents,
11034                 "\n\markdownRendererFancyOlEnd "}

```

```
11035         end
11036     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.

```
11037     function self.fancyitem(s,num)
11038         if num ~= nil then
11039             return {"\\markdownRendererFancyOListItemWithNumber{" ,num,"} ",s,
11040                 "\\markdownRendererFancyOListItemEnd "}
11041         else
11042             return {"\\markdownRendererFancyOListItem ",s,
11043                 "\\markdownRendererFancyOListItemEnd "}
11044         end
11045     end
11046 end, extend_reader = function(self)
11047     local parsers = self.parsers
11048     local options = self.options
11049     local writer = self.writer
11050
11051     local function combine_markers_and_delims(markers, delims)
11052         local markers_table = {}
11053         for _,marker in ipairs(markers) do
11054             local start_marker
11055             local continuation_marker
11056             if type(marker) == "table" then
11057                 start_marker = marker[1]
11058                 continuation_marker = marker[2]
11059             else
11060                 start_marker = marker
11061                 continuation_marker = marker
11062             end
11063             for _,delim in ipairs(delims) do
11064                 table.insert(markers_table,
11065                     {start_marker, continuation_marker, delim})
11066             end
11067         end
11068         return markers_table
11069     end
11070
11071     local function join_table_with_func(func, markers_table)
11072         local pattern = func(table.unpack(markers_table[1]))
11073         for i = 2, #markers_table do
11074             pattern = pattern + func(table.unpack(markers_table[i]))
11075         end
11076         return pattern
11077     end
```

```

11078
11079     local lowercase_letter_marker = R("az")
11080     local uppercase_letter_marker = R("AZ")
11081
11082     local roman_marker = function(chars)
11083         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11084         local l, x, v, i
11085             = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11086         return  m-3
11087             * (c*m + c*d + d-1 * c-3)
11088             * (x*c + x*l + l-1 * x-3)
11089             * (i*x + i*v + v-1 * i-3)
11090     end
11091
11092     local lowercase_roman_marker
11093         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11094     local uppercase_roman_marker
11095         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11096
11097     local lowercase_opening_roman_marker = P("i")
11098     local uppercase_opening_roman_marker = P("I")
11099
11100     local digit_marker = parsers.dig * parsers.dig-8
11101
11102     local markers = {
11103         {lowercase_opening_roman_marker, lowercase_roman_marker},
11104         {uppercase_opening_roman_marker, uppercase_roman_marker},
11105         lowercase_letter_marker,
11106         uppercase_letter_marker,
11107         lowercase_roman_marker,
11108         uppercase_roman_marker,
11109         digit_marker
11110     }
11111
11112     local delims = {
11113         parsers.period,
11114         parsers.rparent
11115     }
11116
11117     local markers_table = combine_markers_and_delims(markers, delims)
11118
11119     local function enumerator(start_marker, _,
11120                             delimiter_type, interrupting)
11121         local delimiter_range
11122         local allowed_end
11123         if interrupting then
11124             delimiter_range = P("1")

```

```

11125         allowed_end = C(parsers.spacechar~1) * #parsers.linechar
11126     else
11127         delimiter_range = start_marker
11128         allowed_end = C(parsers.spacechar~1)
11129             + #(parsers.newline + parsers.eof)
11130     end
11131
11132     return parsers.check_trail
11133         * Ct(C(delimiter_range) * C(delimiter_type))
11134         * allowed_end
11135 end
11136
11137 local starter = join_table_with_func(enumerator, markers_table)
11138
11139 local TightListItem = function(starter)
11140     return parsers.add_indent(starter, "li")
11141         * parsers.indented_content_tight
11142 end
11143
11144 local LooseListItem = function(starter)
11145     return parsers.add_indent(starter, "li")
11146         * parsers.indented_content_loose
11147         * remove_indent("li")
11148 end
11149
11150 local function roman2number(roman)
11151     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11152         ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11153     local numeral = 0
11154
11155     local i = 1
11156     local len = string.len(roman)
11157     while i < len do
11158         local z1, z2 = romans[ string.sub(roman, i, i) ],
11159             romans[ string.sub(roman, i+1, i+1) ]
11160         if z1 < z2 then
11161             numeral = numeral + (z2 - z1)
11162             i = i + 2
11163         else
11164             numeral = numeral + z1
11165             i = i + 1
11166         end
11167     end
11168     if i <= len then
11169         numeral = numeral + romans[ string.sub(roman,i,i) ]
11170     end
11171     return numeral

```

```

11172     end
11173
11174     local function sniffstyle(numstr, delimend)
11175         local numdelim
11176         if delimend == ")" then
11177             numdelim = "OneParen"
11178         elseif delimend == "." then
11179             numdelim = "Period"
11180         else
11181             numdelim = "Default"
11182         end
11183
11184         local num
11185         num = numstr:match("^([I])$")
11186         if num then
11187             return roman2number(num), "UpperRoman", numdelim
11188         end
11189         num = numstr:match("^([i])$")
11190         if num then
11191             return roman2number(string.upper(num)), "LowerRoman", numdelim
11192         end
11193         num = numstr:match("^([A-Z])$")
11194         if num then
11195             return string.byte(num) - string.byte("A") + 1,
11196                "UpperAlpha", numdelim
11197         end
11198         num = numstr:match("^([a-z])$")
11199         if num then
11200             return string.byte(num) - string.byte("a") + 1,
11201                "LowerAlpha", numdelim
11202         end
11203         num = numstr:match("^([IVXLCDM]+)")
11204         if num then
11205             return roman2number(num), "UpperRoman", numdelim
11206         end
11207         num = numstr:match("^([ivxlc dm]+)")
11208         if num then
11209             return roman2number(string.upper(num)), "LowerRoman", numdelim
11210         end
11211         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11212     end
11213
11214     local function fancylist(items,tight,start)
11215         local startnum, numstyle, numdelim
11216         = sniffstyle(start[2][1], start[2][2])
11217         return writer.fancylist(items,tight,
11218                                options.startNumber and startnum or 1,

```

```

11219             numstyle or "Decimal",
11220             numdelim or "Default")
11221     end
11222
11223     local FancyListOfType
11224     = function(start_marker, continuation_marker, delimiter_type)
11225         local enumerator_start
11226         = enumerator(start_marker, continuation_marker,
11227                     delimiter_type)
11228         local enumerator_cont
11229         = enumerator(continuation_marker, continuation_marker,
11230                     delimiter_type)
11231         return Cg(enumerator_start, "listtype")
11232             * (Ct( TightListItem(Cb("listtype"))
11233                 * ((parsers.check_minimal_indent / "")
11234                   * TightListItem(enumerator_cont))^0)
11235             * Cc(true)
11236             * -#((parsers.conditionally_indented_blankline^0 / "")
11237                 * parsers.check_minimal_indent * enumerator_cont)
11238             + Ct( LooseListItem(Cb("listtype"))
11239                 * ((parsers.conditionally_indented_blankline^0 / "")
11240                   * (parsers.check_minimal_indent / "")
11241                   * LooseListItem(enumerator_cont))^0)
11242             * Cc(false)
11243             ) * Ct(Cb("listtype")) / fancylist
11244     end
11245
11246     local FancyList
11247     = join_table_with_func(FancyListOfType, markers_table)
11248
11249     local ListStarter = starter
11250
11251     self.update_rule("OrderedList", FancyList)
11252     self.update_rule("ListStarter", ListStarter)
11253 end
11254 }
11255 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.

```
11256 M.extensions.fenced_code = function(blank_before_code_fence,  
11257                                     allow_attributes,  
11258                                     allow_raw_blocks)  
11259   return {  
11260     name = "built-in fenced_code syntax extension",  
11261     extend_writer = function(self)  
11262       local options = self.options  
11263
```

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.

```
11264     function self.fencedCode(s, i, attr)  
11265       if not self.is_writing then return "" end  
11266       s = s:gsub("\n$", "")  
11267       local buf = {}  
11268       if attr ~= nil then  
11269         table.insert(buf,  
11270           {"\\markdownRendererFencedCodeAttributeContextBegin",  
11271             self.attributes(attr)})  
11272       end  
11273       local name = util.cache_verbatim(options.cacheDir, s)  
11274       table.insert(buf,  
11275         {"\\markdownRendererInputFencedCode{",  
11276           name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}"}  
11277       if attr ~= nil then  
11278         table.insert(buf,  
11279           "\\markdownRendererFencedCodeAttributeContextEnd{")  
11280       end  
11281       return buf  
11282     end  
11283
```

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

```
11284     if allow_raw_blocks then  
11285       function self.rawBlock(s, attr)  
11286         if not self.is_writing then return "" end  
11287         s = s:gsub("\n$", "")  
11288         local name = util.cache_verbatim(options.cacheDir, s)  
11289         return {"\\markdownRendererInputRawBlock{",  
11290           name,"}{" , self.string(attr),"}"}  
11291       end  
11292     end  
11293   end, extend_reader = function(self)  
11294     local parsers = self.parsers
```

```

11295     local writer = self.writer
11296
11297     local function captures_geq_length(_,i,a,b)
11298         return #a >= #b and i
11299     end
11300
11301     local function strip_enclosing_whitespaces(str)
11302         return str:gsub("^%s*(.)%s*$", "%1")
11303     end
11304
11305     local tilde_infostring = Cs(Cs((V("HtmlEntity")
11306         + parsers.anyescaped
11307         - parsers.newline)^0)
11308         / strip_enclosing_whitespaces)
11309
11310     local backtick_infostring
11311     = Cs( Cs((V("HtmlEntity")
11312         + ( -#(parsers.backslash * parsers.backtick)
11313         * parsers.anyescaped)
11314         - parsers.newline
11315         - parsers.backtick)^0)
11316         / strip_enclosing_whitespaces)
11317
11318     local fenceindent
11319
11320     local function has_trail(indent_table)
11321         return indent_table ~= nil and
11322             indent_table.trail ~= nil and
11323             next(indent_table.trail) ~= nil
11324     end
11325
11326     local function has_indents(indent_table)
11327         return indent_table ~= nil and
11328             indent_table.indents ~= nil and
11329             next(indent_table.indents) ~= nil
11330     end
11331
11332     local function get_last_indent_name(indent_table)
11333         if has_indents(indent_table) then
11334             return indent_table.indents[#indent_table.indents].name
11335         end
11336     end
11337
11338     local count_fenced_start_indent =
11339     function(_, _, indent_table, trail)
11340         local last_indent_name = get_last_indent_name(indent_table)
11341         fenceindent = 0

```

```

11342         if last_indent_name ~= "li" then
11343             fenceindent = #trail
11344         end
11345         return true
11346     end
11347
11348     local fencehead = function(char, infostring)
11349         return Cmt( Cb("indent_info")
11350             * parsers.check_trail, count_fenced_start_indent)
11351             * Cg(char^3, "fencelength")
11352             * parsers.optionalspace
11353             * infostring
11354             * (parsers.newline + parsers.eof)
11355     end
11356
11357     local fencetail = function(char)
11358         return parsers.check_trail_no_rem
11359             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11360             * parsers.optionalspace * (parsers.newline + parsers.eof)
11361             + parsers.eof
11362     end
11363
11364     local process_fenced_line =
11365         function(s, i, -- luacheck: ignore s i
11366             indent_table, line_content, is_blank)
11367         local remainder = ""
11368         if has_trail(indent_table) then
11369             remainder = indent_table.trail.internal_remainder
11370         end
11371
11372         if is_blank
11373             and get_last_indent_name(indent_table) == "li" then
11374             remainder = ""
11375         end
11376
11377         local str = remainder .. line_content
11378         local index = 1
11379         local remaining = fenceindent
11380
11381         while true do
11382             local c = str:sub(index, index)
11383             if c == " " and remaining > 0 then
11384                 remaining = remaining - 1
11385                 index = index + 1
11386             elseif c == "\t" and remaining > 3 then
11387                 remaining = remaining - 4
11388                 index = index + 1

```

```

11389         else
11390             break
11391         end
11392     end
11393
11394     return true, str:sub(index)
11395 end
11396
11397 local fencedline = function(char)
11398     return Cmt( Cb("indent_info")
11399         * C(parsers.line - fencetail(char))
11400         * Cc(false), process_fenced_line)
11401 end
11402
11403 local blankfencedline
11404     = Cmt( Cb("indent_info")
11405         * C(parsers.blankline)
11406         * Cc(true), process_fenced_line)
11407
11408 local TildeFencedCode
11409     = fencehead(parsers.tilde, tilde_infostring)
11410     * Cs(( parsers.check_minimal_blank_indent / ""
11411         * blankfencedline
11412         + ( parsers.check_minimal_indent / ""
11413         * fencedline(parsers.tilde))^0)
11414     * ( (parsers.check_minimal_indent / ""
11415         * fencetail(parsers.tilde) + parsers.succeed)
11416
11417 local BacktickFencedCode
11418     = fencehead(parsers.backtick, backtick_infostring)
11419     * Cs(( (parsers.check_minimal_blank_indent / ""
11420         * blankfencedline
11421         + (parsers.check_minimal_indent / ""
11422         * fencedline(parsers.backtick))^0)
11423     * ( (parsers.check_minimal_indent / ""
11424         * fencetail(parsers.backtick) + parsers.succeed)
11425
11426 local infostring_with_attributes
11427     = Ct(C((parsers.linechar
11428         - ( parsers.optionalspace
11429         * parsers.attributes))^0)
11430     * parsers.optionalspace
11431     * Ct(parsers.attributes))
11432
11433 local FencedCode
11434     = ((TildeFencedCode + BacktickFencedCode)
11435     / function(infostring, code)

```

```

11436         local expanded_code = self.expandtabs(code)
11437
11438         if allow_raw_blocks then
11439             local raw_attr = lpeg.match(parsers.raw_attribute,
11440                                       infostring)
11441             if raw_attr then
11442                 return writer.rawBlock(expanded_code, raw_attr)
11443             end
11444         end
11445
11446         local attr = nil
11447         if allow_attributes then
11448             local match = lpeg.match(infostring_with_attributes,
11449                                     infostring)
11450             if match then
11451                 infostring, attr = table.unpack(match)
11452             end
11453         end
11454         return writer.fencedCode(expanded_code, infostring, attr)
11455     end)
11456
11457     self.insert_pattern("Block after Verbatim",
11458                       FencedCode, "FencedCode")
11459
11460     local fencestart
11461     if blank_before_code_fence then
11462         fencestart = parsers.fail
11463     else
11464         fencestart = fencehead(parsers.backtick, backtick_infostring)
11465                       + fencehead(parsers.tilde, tilde_infostring)
11466     end
11467
11468     self.update_rule("EndlineExceptions", function(previous_pattern)
11469         if previous_pattern == nil then
11470             previous_pattern = parsers.EndlineExceptions
11471         end
11472         return previous_pattern + fencestart
11473     end)
11474
11475     self.add_special_character("`")
11476     self.add_special_character("~")
11477 end
11478 }
11479 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.

```
11480 M.extensions.fenced_divs = function(blank_before_div_fence)
11481   return {
11482     name = "built-in fenced_divs syntax extension",
11483     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.

```
11484     function self.div_begin(attributes)
11485       local start_output
11486       = {"\\markdownRendererFencedDivAttributeContextBegin\n",
11487         self.attributes(attributes)}
11488       local end_output
11489       = {"\\markdownRendererFencedDivAttributeContextEnd{}}
11490       return self.push_attributes(
11491         "div", attributes, start_output, end_output)
11492     end
```

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

```
11493     function self.div_end()
11494       return self.pop_attributes("div")
11495     end
11496   end, extend_reader = function(self)
11497     local parsers = self.parsers
11498     local writer = self.writer
```

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

```
11499     local fenced_div_infostring
11500       = C((parsers.linechar
11501         - ( parsers.spacechar^1
11502           * parsers.colon^1))^1)
11503
11504     local fenced_div_begin = parsers.nonindentspace
11505       * parsers.colon^3
11506       * parsers.optionalspace
11507       * fenced_div_infostring
11508       * ( parsers.spacechar^1
11509         * parsers.colon^1)^0
11510       * parsers.optionalspace
11511       * (parsers.newline + parsers.eof)
11512
11513     local fenced_div_end = parsers.nonindentspace
11514       * parsers.colon^3
```

```

11515             * parsers.optionalspace
11516             * (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.

```

11517     self.initialize_named_group("fenced_div_level", "0")
11518     self.initialize_named_group("fenced_div_num_opening_indents")
11519
11520     local function increment_div_level()
11521         local push_indent_table =
11522             function(s, i, indent_table, -- luacheck: ignore s i
11523                 fenced_div_num_opening_indents, fenced_div_level)
11524                 fenced_div_level = tonumber(fenced_div_level) + 1
11525                 local num_opening_indents = 0
11526                 if indent_table.indents ~= nil then
11527                     num_opening_indents = #indent_table.indents
11528                 end
11529                 fenced_div_num_opening_indents[fenced_div_level]
11530                 = num_opening_indents
11531                 return true, fenced_div_num_opening_indents
11532             end
11533
11534     local increment_level =
11535         function(s, i, fenced_div_level) -- luacheck: ignore s i
11536             fenced_div_level = tonumber(fenced_div_level) + 1
11537             return true, tostring(fenced_div_level)
11538         end
11539
11540     return Cg( Cmt( Cb("indent_info")
11541         * Cb("fenced_div_num_opening_indents")
11542         * Cb("fenced_div_level"), push_indent_table)
11543         , "fenced_div_num_opening_indents")
11544         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11545         , "fenced_div_level")
11546     end
11547
11548     local function decrement_div_level()
11549         local pop_indent_table =
11550             function(s, i, -- luacheck: ignore s i
11551                 fenced_div_indent_table, fenced_div_level)
11552                 fenced_div_level = tonumber(fenced_div_level)
11553                 fenced_div_indent_table[fenced_div_level] = nil
11554                 return true, tostring(fenced_div_level - 1)

```

```

11555         end
11556
11557         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11558             * Cb("fenced_div_level"), pop_indent_table)
11559             , "fenced_div_level")
11560     end
11561
11562
11563     local non_fenced_div_block
11564         = parsers.check_minimal_indent * V("Block")
11565         - parsers.check_minimal_indent_and_trail * fenced_div_end
11566
11567     local non_fenced_div_paragraph
11568         = parsers.check_minimal_indent * V("Paragraph")
11569         - parsers.check_minimal_indent_and_trail * fenced_div_end
11570
11571     local blank = parsers.minimally_indented_blank
11572
11573     local block_separated = parsers.block_sep_group(blank)
11574         * non_fenced_div_block
11575
11576     local loop_body_pair
11577         = parsers.create_loop_body_pair(block_separated,
11578             non_fenced_div_paragraph,
11579             parsers.block_sep_group(blank),
11580             parsers.par_sep_group(blank))
11581
11582     local content_loop = ( non_fenced_div_block
11583         * loop_body_pair.block^0
11584         + non_fenced_div_paragraph
11585         * block_separated
11586         * loop_body_pair.block^0
11587         + non_fenced_div_paragraph
11588         * loop_body_pair.par^0)
11589         * blank^0
11590
11591     local FencedDiv = fenced_div_begin
11592         / function (infostring)
11593             local attr
11594                 = lpeg.match(Ct(parsers.attributes),
11595                     infostring)
11596             if attr == nil then
11597                 attr = {"." .. infostring}
11598             end
11599             return attr
11600         end
11601     / writer.div_begin

```



```

11602         * increment_div_level()
11603         * parsers.skipblanklines
11604         * Ct(content_loop)
11605         * parsers.minimally_indented_blank^0
11606         * parsers.check_minimal_indent_and_trail
11607         * fenced_div_end
11608         * decrement_div_level()
11609         * (Cc("") / writer.div_end)
11610
11611     self.insert_pattern("Block after Verbatim",
11612                       FencedDiv, "FencedDiv")
11613
11614     self.add_special_character(":")
11615

```

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.

```

11616     local function is_inside_div()
11617         local check_div_level =
11618             function(s, i, fenced_div_level) -- luacheck: ignore s i
11619                 fenced_div_level = tonumber(fenced_div_level)
11620                 return fenced_div_level > 0
11621             end
11622
11623     return Cmt(Cb("fenced_div_level"), check_div_level)
11624 end
11625
11626 local function check_indent()
11627     local compare_indent =
11628         function(s, i, indent_table, -- luacheck: ignore s i
11629                 fenced_div_num_opening_indents, fenced_div_level)
11630             fenced_div_level = tonumber(fenced_div_level)
11631             local num_current_indents
11632                 = ( indent_table.current_line_indents ~= nil and
11633                   #indent_table.current_line_indents) or 0
11634             local num_opening_indents
11635                 = fenced_div_num_opening_indents[fenced_div_level]
11636             return num_current_indents == num_opening_indents
11637         end
11638
11639     return Cmt( Cb("indent_info")
11640               * Cb("fenced_div_num_opening_indents")
11641               * Cb("fenced_div_level"), compare_indent)
11642 end
11643
11644 local fencestart = is_inside_div()

```

```

11645             * fenced_div_end
11646             * check_indent()
11647
11648     if not blank_before_div_fence then
11649         self.update_rule("EndlineExceptions", function(previous_pattern)
11650             if previous_pattern == nil then
11651                 previous_pattern = parsers.EndlineExceptions
11652             end
11653             return previous_pattern + fencestart
11654         end)
11655     end
11656 end
11657 }
11658 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11659 M.extensions.header_attributes = function()
11660     return {
11661         name = "built-in header_attributes syntax extension",
11662         extend_writer = function()
11663         end, extend_reader = function(self)
11664             local parsers = self.parsers
11665             local writer = self.writer
11666
11667             local function strip_atx_end(s)
11668                 return s:gsub("%s+##%s*$", "")
11669             end
11670
11671             local AtxHeading = Cg(parsers.heading_start, "level")
11672                 * parsers.optionalspace
11673                 * (C(((parsers.linechar
11674                     - (parsers.attributes
11675                         * parsers.optionalspace
11676                         * parsers.newline))
11677                     * (parsers.linechar
11678                         - parsers.lbrace)^0)^1)
11679                     / strip_atx_end
11680                     / parsers.parse_heading_text)
11681                 * Cg(Ct(parsers.newline
11682                     + (parsers.attributes
11683                         * parsers.optionalspace
11684                         * parsers.newline)), "attributes")
11685                 * Cb("level")
11686                 * Cb("attributes")

```

```

11687         / writer.heading
11688
11689     local function strip_trailing_spaces(s)
11690         return s:gsub("%s*$", "")
11691     end
11692
11693     local heading_line = (parsers.linechar
11694                         - (parsers.attributes
11695                           * parsers.optionalspace
11696                           * parsers.newline))^1
11697                         - parsers.thematic_break_lines
11698
11699     local heading_text
11700     = heading_line
11701     * ( (V("Endline") / "\n")
11702       * (heading_line - parsers.heading_level))^0
11703     * parsers.newline^-1
11704
11705     local SettextHeading
11706     = parsers.freeze_trail * parsers.check_trail_no_rem
11707     * #(heading_text
11708       * (parsers.attributes
11709         * parsers.optionalspace
11710         * parsers.newline)^-1
11711       * parsers.check_minimal_indent
11712       * parsers.check_trail
11713       * parsers.heading_level)
11714     * Cs(heading_text) / strip_trailing_spaces
11715     / parsers.parse_heading_text
11716     * Cg(Ct((parsers.attributes
11717           * parsers.optionalspace
11718           * parsers.newline)^-1), "attributes")
11719     * parsers.check_minimal_indent_and_trail * parsers.heading_level
11720     * Cb("attributes")
11721     * parsers.newline
11722     * parsers.unfreeze_trail
11723     / writer.heading
11724
11725     local Heading = AtxHeading + SettextHeading
11726     self.update_rule("Heading", Heading)
11727 end
11728 }
11729 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc in-line code attribute syntax extension.

```
11730 M.extensions.inline_code_attributes = function()
11731   return {
11732     name = "built-in inline_code_attributes syntax extension",
11733     extend_writer = function()
11734     end, extend_reader = function(self)
11735       local writer = self.writer
11736
11737       local CodeWithAttributes = parsers.inticks
11738         * Ct(parsers.attributes)
11739         / writer.code
11740
11741       self.insert_pattern("Inline before Code",
11742         CodeWithAttributes,
11743         "CodeWithAttributes")
11744     end
11745   }
11746 end
```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
11747 M.extensions.line_blocks = function()
11748   return {
11749     name = "built-in line_blocks syntax extension",
11750     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.

```
11751     function self.lineblock(lines)
11752       if not self.is_writing then return "" end
11753       local buffer = {}
11754       for i = 1, #lines - 1 do
11755         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11756       end
11757       buffer[#buffer + 1] = lines[#lines]
11758
11759       return {"\\markdownRendererLineBlockBegin\n"
11760         ,buffer,
11761         "\n\\markdownRendererLineBlockEnd "}
11762     end
11763   end, extend_reader = function(self)
11764     local parsers = self.parsers
11765     local writer = self.writer
11766
```

```

11767     local LineBlock
11768         = Ct((Cs(( (parsers.pipe * parsers.space) / ""
11769                 * ((parsers.space)/entities.char_entity("nbsp"))^0
11770                 * parsers.linechar^0 * (parsers.newline/""))
11771                 * (-parsers.pipe
11772                   * (parsers.space^1/" ")
11773                   * parsers.linechar^1
11774                   * (parsers.newline/"")
11775                   )^0
11776                 * (parsers.blankline/"")^0)
11777           / self.parser_functions.parse_inlines)^1)
11778     / writer.lineblock
11779
11780     self.insert_pattern("Block after Blockquote",
11781                       LineBlock, "LineBlock")
11782 end
11783 }
11784 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11785 M.extensions.mark = function()
11786   return {
11787     name = "built-in mark syntax extension",
11788     extend_writer = function(self)

```

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

```

11789     function self.mark(s)
11790       if self.flatten_inlines then return s end
11791       return {"\\markdownRendererMark{" , s, "}"}
11792     end
11793   end, extend_reader = function(self)
11794     local parsers = self.parsers
11795     local writer = self.writer
11796
11797     local doubleequals = P("==")
11798
11799     local Mark
11800       = parsers.between(V("Inline"), doubleequals, doubleequals)
11801       / function (inlines) return writer.mark(inlines) end
11802
11803     self.add_special_character("=")
11804     self.insert_pattern("Inline before LinkAndEmph",
11805                       Mark, "Mark")
11806   end
11807 }

```

11808 end

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11809 M.extensions.link_attributes = function()
11810   return {
11811     name = "built-in link_attributes syntax extension",
11812     extend_writer = function()
11813     end, extend_reader = function(self)
11814       local parsers = self.parsers
11815       local options = self.options
11816
```

The following patterns define link reference definitions with attributes.

```
11817     local define_reference_parser
11818       = (parsers.check_trail / "")
11819       * parsers.link_label
11820       * parsers.colon
11821       * parsers.spnlc * parsers.url
11822       * ( parsers.spnlc_sep * parsers.title
11823         * (parsers.spnlc * Ct(parsers.attributes))
11824         * parsers.only_blank
11825         + parsers.spnlc_sep * parsers.title * parsers.only_blank
11826         + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11827         * parsers.only_blank
11828         + Cc("") * parsers.only_blank)
11829
11830     local ReferenceWithAttributes = define_reference_parser
11831                                   / self.register_link
11832
11833     self.update_rule("Reference", ReferenceWithAttributes)
11834
```

The following patterns define direct and indirect links with attributes.

```
11835
11836     local LinkWithAttributesAndEmph
11837       = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11838         "match_link_attributes"))
11839       / self.defer_link_and_emphasis_processing
11840
11841     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11842
```

The following patterns define autolinks with attributes.

```
11843     local AutoLinkUrlWithAttributes
11844       = parsers.auto_link_url
```

```

11845             * Ct(parsers.attributes)
11846             / self.auto_link_url
11847
11848     self.insert_pattern("Inline before AutoLinkUrl",
11849                       AutoLinkUrlWithAttributes,
11850                       "AutoLinkUrlWithAttributes")
11851
11852     local AutoLinkEmailWithAttributes
11853           = parsers.auto_link_email
11854           * Ct(parsers.attributes)
11855           / self.auto_link_email
11856
11857     self.insert_pattern("Inline before AutoLinkEmail",
11858                       AutoLinkEmailWithAttributes,
11859                       "AutoLinkEmailWithAttributes")
11860
11861     if options.relativeReferences then
11862
11863         local AutoLinkRelativeReferenceWithAttributes
11864               = parsers.auto_link_relative_reference
11865               * Ct(parsers.attributes)
11866               / self.auto_link_url
11867
11868         self.insert_pattern(
11869           "Inline before AutoLinkRelativeReference",
11870           AutoLinkRelativeReferenceWithAttributes,
11871           "AutoLinkRelativeReferenceWithAttributes")
11872
11873     end
11874
11875 end
11876 }
11877 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.

```

11878 M.extensions.notes = function(notes, inline_notes)
11879   assert(notes or inline_notes)
11880   return {
11881     name = "built-in notes syntax extension",
11882     extend_writer = function(self)

```

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

```
11883     function self.note(s)
11884         if self.flatten_inlines then return "" end
11885         return {"\\markdownRendererNote{" ,s,"}"}
11886     end
11887 end, extend_reader = function(self)
11888     local parsers = self.parsers
11889     local writer = self.writer
11890
11891     local rawnotes = parsers.rawnotes
11892
11893     if inline_notes then
11894         local InlineNote
11895             = parsers.circumflex
11896             * ( parsers.link_label
11897               / self.parser_functions.parse_inlines_no_inline_note)
11898             / writer.note
11899
11900         self.insert_pattern("Inline after LinkAndEmph",
11901                             InlineNote, "InlineNote")
11902     end
11903     if notes then
11904         local function strip_first_char(s)
11905             return s:sub(2)
11906         end
11907
11908         local RawNoteRef
11909             = #(parsers.lbracket * parsers.circumflex)
11910             * parsers.link_label / strip_first_char
11911
11912         -- like indirect_link
11913         local function lookup_note(ref)
11914             return writer.defer_call(function()
11915                 local found = rawnotes[self.normalize_tag(ref)]
11916                 if found then
11917                     return writer.note(
11918                         self.parser_functions.parse_blocks_nested(found))
11919                 else
11920                     return {"[",
11921                             self.parser_functions.parse_inlines("^" .. ref), "]" }
11922                 end
11923             end)
11924         end
11925
11926         local function register_note(ref,rawnote)
11927             local normalized_tag = self.normalize_tag(ref)
```



```

11928         if rawnotes[normalized_tag] == nil then
11929             rawnotes[normalized_tag] = rawnote
11930         end
11931         return ""
11932     end
11933
11934     local NoteRef = RawNoteRef / lookup_note
11935
11936     local optionally_indented_line
11937         = parsers.check_optional_indent_and_any_trail * parsers.line
11938
11939     local blank
11940         = parsers.check_optional_blank_indent_and_any_trail
11941         * parsers.optionalspace * parsers.newline
11942
11943     local chunk
11944         = Cs(parsers.line
11945             * (optionally_indented_line - blank)^0)
11946
11947     local indented_blocks = function(bl)
11948         return Cs( bl
11949             * ( blank^1 * (parsers.check_optional_indent / "")
11950             * parsers.check_code_trail
11951             * -parsers.blankline * bl)^0)
11952     end
11953
11954     local NoteBlock
11955         = parsers.check_trail_no_rem
11956         * RawNoteRef * parsers.colon
11957         * parsers.spnlc * indented_blocks(chunk)
11958         / register_note
11959
11960     local Reference = NoteBlock + parsers.Reference
11961
11962     self.update_rule("Reference", Reference)
11963     self.insert_pattern("Inline before LinkAndEmph",
11964         NoteRef, "NoteRef")
11965     end
11966
11967     self.add_special_character("^")
11968 end
11969 }
11970 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syn-

tax 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.

```

11971 M.extensions.pipe_tables = function(table_captions, table_attributes)
11972
11973   local function make_pipe_table_rectangular(rows)
11974     local num_columns = #rows[2]
11975     local rectangular_rows = {}
11976     for i = 1, #rows do
11977       local row = rows[i]
11978       local rectangular_row = {}
11979       for j = 1, num_columns do
11980         rectangular_row[j] = row[j] or ""
11981       end
11982       table.insert(rectangular_rows, rectangular_row)
11983     end
11984     return rectangular_rows
11985   end
11986
11987   local function pipe_table_row(allow_empty_first_column
11988                                 , nonempty_column
11989                                 , column_separator
11990                                 , column)
11991     local row_beginning
11992     if allow_empty_first_column then
11993       row_beginning = -- empty first column
11994                       #(parsers.spacechar^4
11995                         * column_separator)
11996                       * parsers.optionalspace
11997                       * column
11998                       * parsers.optionalspace
11999                       -- non-empty first column
12000                       + parsers.nonindentSPACE
12001                       * nonempty_column^-1
12002                       * parsers.optionalspace
12003     else
12004       row_beginning = parsers.nonindentSPACE
12005                       * nonempty_column^-1
12006                       * parsers.optionalspace
12007     end
12008
12009     return Ct(row_beginning
12010              * (-- single column with no leading pipes
12011                #(column_separator)

```

```

12012         * parsers.optionalspace
12013         * parsers.newline)
12014     * column_separator
12015     * parsers.optionalspace
12016     -- single column with leading pipes or
12017     -- more than a single column
12018     + (column_separator
12019       * parsers.optionalspace
12020       * column
12021       * parsers.optionalspace)^1
12022     * (column_separator
12023       * parsers.optionalspace)^-1))
12024 end
12025
12026 return {
12027     name = "built-in pipe_tables syntax extension",
12028     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.

```

12029     function self.table(rows, caption, attributes)
12030         if not self.is_writing then return "" end
12031         local buffer = {}
12032         if attributes ~= nil then
12033             table.insert(buffer,
12034                 "\\markdownRendererTableAttributeContextBegin\n")
12035             table.insert(buffer, self.attributes(attributes))
12036         end
12037         table.insert(buffer,
12038             {"\\markdownRendererTable{",
12039             caption or "", "}{" , #rows - 1, "}{" ,
12040             #rows[1], "}")
12041         local temp = rows[2] -- put alignments on the first row
12042         rows[2] = rows[1]
12043         rows[1] = temp
12044         for i, row in ipairs(rows) do
12045             table.insert(buffer, "{")
12046             for _, column in ipairs(row) do
12047                 if i > 1 then -- do not use braces for alignments
12048                     table.insert(buffer, "{")
12049                 end
12050                 table.insert(buffer, column)
12051                 if i > 1 then
12052                     table.insert(buffer, "}")
12053                 end
12054             end
12055             table.insert(buffer, "}")

```

```

12056         end
12057         if attributes ~= nil then
12058             table.insert(buffer,
12059                 "\\markdownRendererTableAttributeContextEnd{)")
12060         end
12061         return buffer
12062     end
12063 end, extend_reader = function(self)
12064     local parsers = self.parsers
12065     local writer = self.writer
12066
12067     local table_hline_separator = parsers.pipe + parsers.plus
12068
12069     local table_hline_column = (parsers.dash
12070         - #(parsers.dash
12071             * (parsers.spacechar
12072                 + table_hline_separator
12073                 + parsers.newline)))^1
12074     * (parsers.colon * Cc("r")
12075         + parsers.dash * Cc("d"))
12076     + parsers.colon
12077     * (parsers.dash
12078         - #(parsers.dash
12079             * (parsers.spacechar
12080                 + table_hline_separator
12081                 + parsers.newline)))^1
12082     * (parsers.colon * Cc("c")
12083         + parsers.dash * Cc("l"))
12084
12085     local table_hline = pipe_table_row(false
12086         , table_hline_column
12087         , table_hline_separator
12088         , table_hline_column)
12089
12090     local table_caption_beginning
12091     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
12092         * parsers.optionalspace * parsers.newline)^0
12093     * parsers.check_minimal_indent_and_trail
12094     * (P("Table")^-1 * parsers.colon)
12095     * parsers.optionalspace
12096
12097     local function strip_trailing_spaces(s)
12098         return s:gsub("%s*$", "")
12099     end
12100
12101     local table_row
12102     = pipe_table_row(true

```

```

12103         , (C((parsers.linechar - parsers.pipe)^1)
12104           / strip_trailing_spaces
12105           / self.parser_functions.parse_inlines)
12106         , parsers.pipe
12107         , (C((parsers.linechar - parsers.pipe)^0)
12108           / strip_trailing_spaces
12109           / self.parser_functions.parse_inlines))
12110
12111     local table_caption
12112     if table_captions then
12113         table_caption = #table_caption_beginning
12114             * table_caption_beginning
12115         if table_attributes then
12116             table_caption = table_caption
12117                 * (C(((( parsers.linechar
12118                   - (parsers.attributes
12119                     * parsers.optionalspace
12120                     * parsers.newline
12121                     * -( parsers.optionalspace
12122                       * parsers.linechar))))
12123                   + ( parsers.newline
12124                     * #( parsers.optionalspace
12125                       * parsers.linechar)
12126                     * C(parsers.optionalspace)
12127                       / writer.space))
12128                   * (parsers.linechar
12129                     - parsers.lbrace)^0)^1)
12130                   / self.parser_functions.parse_inlines)
12131                 * (parsers.newline
12132                   + ( Ct(parsers.attributes)
12133                     * parsers.optionalspace
12134                     * parsers.newline))
12135         else
12136             table_caption = table_caption
12137                 * C(( parsers.linechar
12138                   + ( parsers.newline
12139                     * #( parsers.optionalspace
12140                       * parsers.linechar)
12141                     * C(parsers.optionalspace)
12142                       / writer.space))^1)
12143                   / self.parser_functions.parse_inlines
12144                 * parsers.newline
12145         end
12146     else
12147         table_caption = parsers.fail
12148     end
12149

```

```

12150     local PipeTable
12151         = Ct( table_row * parsers.newline
12152             * (parsers.check_minimal_indent_and_trail / {})
12153             * table_hline * parsers.newline
12154             * ( (parsers.check_minimal_indent / {})
12155                 * table_row * parsers.newline)^0)
12156         / make_pipe_table_rectangular
12157         * table_caption^-1
12158         / writer.table
12159
12160     self.insert_pattern("Block after Blockquote",
12161                       PipeTable, "PipeTable")
12162 end
12163 }
12164 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

12165 M.extensions.raw_inline = function()
12166     return {
12167         name = "built-in raw_inline syntax extension",
12168         extend_writer = function(self)
12169             local options = self.options
12170

```

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

```

12171     function self.rawInline(s, attr)
12172         if not self.is_writing then return "" end
12173         if self.flatten_inlines then return s end
12174         local name = util.cache_verbatim(options.cacheDir, s)
12175         return {"\\markdownRendererInputRawInline{" ,
12176               name,"}{" , self.string(attr),"}" }
12177     end
12178 end, extend_reader = function(self)
12179     local writer = self.writer
12180
12181     local RawInline = parsers.inticks
12182                     * parsers.raw_attribute
12183                     / writer.rawInline
12184
12185     self.insert_pattern("Inline before Code",
12186                       RawInline, "RawInline")
12187 end
12188 }
12189 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
12190 M.extensions.strike_through = function()
12191   return {
12192     name = "built-in strike_through syntax extension",
12193     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.

```
12194       function self.strike_through(s)
12195         if self.flatten_inlines then return s end
12196         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
12197       end
12198   end, extend_reader = function(self)
12199     local parsers = self.parsers
12200     local writer = self.writer
12201
12202     local StrikeThrough = (
12203       parsers.between(parsers.Inline, parsers.doubletildes,
12204         parsers.doubletildes)
12205     ) / writer.strike_through
12206
12207     self.insert_pattern("Inline after LinkAndEmph",
12208       StrikeThrough, "StrikeThrough")
12209
12210     self.add_special_character("~")
12211   end
12212 }
12213 end
```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
12214 M.extensions.subscripts = function()
12215   return {
12216     name = "built-in subscripts syntax extension",
12217     extend_writer = function(self)
```

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

```
12218       function self.subscript(s)
12219         if self.flatten_inlines then return s end
12220         return {"\\markdownRendererSubscript{" ,s,"}"}
12221       end
12222   end, extend_reader = function(self)
```

```

12223     local parsers = self.parsers
12224     local writer = self.writer
12225
12226     local Subscript = (
12227         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12228     ) / writer.subscript
12229
12230     self.insert_pattern("Inline after LinkAndEmph",
12231                         Subscript, "Subscript")
12232
12233     self.add_special_character("~")
12234 end
12235 }
12236 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

12237 M.extensions.superscripts = function()
12238     return {
12239         name = "built-in superscripts syntax extension",
12240         extend_writer = function(self)

```

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

```

12241             function self.superscript(s)
12242                 if self.flatten_inlines then return s end
12243                 return {"\\markdownRendererSuperscript{" ,s, "}"}
12244             end
12245         end, extend_reader = function(self)
12246             local parsers = self.parsers
12247             local writer = self.writer
12248
12249             local Superscript = (
12250                 parsers.between(parsers.Str, parsers.circumflex,
12251                                 parsers.circumflex)
12252             ) / writer.superscript
12253
12254             self.insert_pattern("Inline after LinkAndEmph",
12255                                 Superscript, "Superscript")
12256
12257             self.add_special_character("^")
12258         end
12259     }
12260 end

```



### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12261 M.extensions.tex_math = function(tex_math_dollars,  
12262                                     tex_math_single_backslash,  
12263                                     tex_math_double_backslash)  
12264   return {  
12265     name = "built-in tex_math syntax extension",  
12266     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.

```
12267     function self.display_math(s)  
12268       if self.flatten_inlines then return s end  
12269       return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}  
12270     end
```

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

```
12271     function self.inline_math(s)  
12272       if self.flatten_inlines then return s end  
12273       return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}  
12274     end  
12275   end, extend_reader = function(self)  
12276     local parsers = self.parsers  
12277     local writer = self.writer  
12278  
12279     local function between(p, starter, ender)  
12280       return (starter * Cs(p * (p - ender)^0) * ender)  
12281     end  
12282  
12283     local function strip_preceding_whitespaces(str)  
12284       return str:gsub("^%s*(.-)$", "%1")  
12285     end  
12286  
12287     local allowed_before_closing  
12288       = B( parsers.backslash * parsers.any  
12289           + parsers.any * (parsers.any - parsers.backslash))  
12290  
12291     local allowed_before_closing_no_space  
12292       = B( parsers.backslash * parsers.any  
12293           + parsers.any * (parsers.nonspacechar - parsers.backslash))  
12294
```

The following patterns implement the Pandoc dollar math syntax extension.

```
12295     local dollar_math_content  
12296       = (parsers.newline * (parsers.check_optional_indent / ""))  
12297       + parsers.backslash^-1
```

```

12298     * parsers.linechar)
12299     - parsers.blankline^2
12300     - parsers.dollar
12301
12302     local inline_math_opening_dollars = parsers.dollar
12303         * #(parsers.nonspacechar)
12304
12305     local inline_math_closing_dollars
12306     = allowed_before_closing_no_space
12307     * parsers.dollar
12308     * -#(parsers.digit)
12309
12310     local inline_math_dollars = between(Cs( dollar_math_content),
12311         inline_math_opening_dollars,
12312         inline_math_closing_dollars)
12313
12314     local display_math_opening_dollars = parsers.dollar
12315         * parsers.dollar
12316
12317     local display_math_closing_dollars = parsers.dollar
12318         * parsers.dollar
12319
12320     local display_math_dollars = between(Cs( dollar_math_content),
12321         display_math_opening_dollars,
12322         display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12323     local backslash_math_content
12324     = (parsers.newline * (parsers.check_optional_indent / ""))
12325     + parsers.linechar)
12326     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12327     local inline_math_opening_double = parsers.backslash
12328         * parsers.backslash
12329         * parsers.lparent
12330
12331     local inline_math_closing_double = allowed_before_closing
12332         * parsers.spacechar^0
12333         * parsers.backslash
12334         * parsers.backslash
12335         * parsers.rparent
12336
12337     local inline_math_double = between(Cs( backslash_math_content),
12338         inline_math_opening_double,
12339         inline_math_closing_double)

```

```

12340             / strip_preceding_whitespaces
12341
12342     local display_math_opening_double = parsers.backslash
12343             * parsers.backslash
12344             * parsers.lbracket
12345
12346     local display_math_closing_double = allowed_before_closing
12347             * parsers.spacechar^0
12348             * parsers.backslash
12349             * parsers.backslash
12350             * parsers.rbracket
12351
12352     local display_math_double = between(Cs( backslash_math_content),
12353             display_math_opening_double,
12354             display_math_closing_double)
12355             / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

12356     local inline_math_opening_single = parsers.backslash
12357             * parsers.lparent
12358
12359     local inline_math_closing_single = allowed_before_closing
12360             * parsers.spacechar^0
12361             * parsers.backslash
12362             * parsers.rparent
12363
12364     local inline_math_single = between(Cs( backslash_math_content),
12365             inline_math_opening_single,
12366             inline_math_closing_single)
12367             / strip_preceding_whitespaces
12368
12369     local display_math_opening_single = parsers.backslash
12370             * parsers.lbracket
12371
12372     local display_math_closing_single = allowed_before_closing
12373             * parsers.spacechar^0
12374             * parsers.backslash
12375             * parsers.rbracket
12376
12377     local display_math_single = between(Cs( backslash_math_content),
12378             display_math_opening_single,
12379             display_math_closing_single)
12380             / strip_preceding_whitespaces
12381
12382     local display_math = parsers.fail
12383
12384     local inline_math = parsers.fail
12385

```

```

12386     if tex_math_dollars then
12387         display_math = display_math + display_math_dollars
12388         inline_math = inline_math + inline_math_dollars
12389     end
12390
12391     if tex_math_double_backslash then
12392         display_math = display_math + display_math_double
12393         inline_math = inline_math + inline_math_double
12394     end
12395
12396     if tex_math_single_backslash then
12397         display_math = display_math + display_math_single
12398         inline_math = inline_math + inline_math_single
12399     end
12400
12401     local TexMath = display_math / writer.display_math
12402                   + inline_math / writer.inline_math
12403
12404     self.insert_pattern("Inline after LinkAndEmph",
12405                       TexMath, "TexMath")
12406
12407     if tex_math_dollars then
12408         self.add_special_character("$")
12409     end
12410
12411     if tex_math_single_backslash or tex_math_double_backslash then
12412         self.add_special_character("\\")
12413         self.add_special_character("[")
12414         self.add_special_character("]")
12415         self.add_special_character("(")
12416         self.add_special_character("(")
12417     end
12418 end
12419 }
12420 end

```

### 3.1.7.20 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc YAML metadata block syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata. When both `expect_jekyll_data` and `ensure_jekyll_data` parameters are `true`, then a a markdown document must begin directly with YAML metadata and must contain nothing but YAML metadata.

```

12421 M.extensions.jekyll_data = function(expect_jekyll_data,
12422                                     ensure_jekyll_data)
12423     return {

```

```

12424     name = "built-in jekyll_data syntax extension",
12425     extend_writer = function(self)

```

Define `writer->jekyllData` as a function that will transform an input YAML table `d` to the output format. The table is the value for the key `p` in the parent table; if `p` is nil, then the table has no parent. All scalar keys and values encountered in the table will be cast to a string following YAML serialization rules. String values will also be transformed using the function `t` for the typographic output format used by the `\markdownRendererJekyllDataTypographicString` macro.

```

12426     function self.jekyllData(d, t, p)
12427         if not self.is_writing then return "" end
12428
12429         local buf = {}
12430
12431         local keys = {}
12432         for k, _ in pairs(d) do
12433             table.insert(keys, k)
12434         end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

12435         table.sort(keys, function(first, second)
12436             if type(first) ~= type(second) then
12437                 return type(first) < type(second)
12438             else
12439                 return first < second
12440             end
12441         end)
12442
12443         if not p then
12444             table.insert(buf, "\\markdownRendererJekyllDataBegin")
12445         end
12446
12447         local is_sequence = false
12448         if #d > 0 and #d == #keys then
12449             for i=1, #d do
12450                 if d[i] == nil then
12451                     goto not_a_sequence
12452                 end
12453             end
12454             is_sequence = true
12455         end
12456         ::not_a_sequence::
12457
12458         if is_sequence then
12459             table.insert(buf,
12460                 "\\markdownRendererJekyllDataSequenceBegin{")

```

```

12461         table.insert(buf, self.identifier(p or "null"))
12462         table.insert(buf, "}{"")
12463         table.insert(buf, #keys)
12464         table.insert(buf, "}")
12465     else
12466         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12467         table.insert(buf, self.identifier(p or "null"))
12468         table.insert(buf, "}{"")
12469         table.insert(buf, #keys)
12470         table.insert(buf, "}")
12471     end
12472
12473     for _, k in ipairs(keys) do
12474         local v = d[k]
12475         local typ = type(v)
12476         k = tostring(k or "null")
12477         if typ == "table" and next(v) ~= nil then
12478             table.insert(
12479                 buf,
12480                 self.jekyllData(v, t, k)
12481             )
12482         else
12483             k = self.identifier(k)
12484             v = tostring(v)
12485             if typ == "boolean" then
12486                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12487                 table.insert(buf, k)
12488                 table.insert(buf, "}{"")
12489                 table.insert(buf, v)
12490                 table.insert(buf, "}")
12491             elseif typ == "number" then
12492                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12493                 table.insert(buf, k)
12494                 table.insert(buf, "}{"")
12495                 table.insert(buf, v)
12496                 table.insert(buf, "}")
12497             elseif typ == "string" then
12498                 table.insert(buf,
12499                     "\\markdownRendererJekyllDataProgrammaticString{")
12500                 table.insert(buf, k)
12501                 table.insert(buf, "}{"")
12502                 table.insert(buf, self.identifier(v))
12503                 table.insert(buf, "}")
12504                 table.insert(buf,
12505                     "\\markdownRendererJekyllDataTypographicString{")
12506                 table.insert(buf, k)
12507                 table.insert(buf, "}{"")

```

```

12508         table.insert(buf, t(v))
12509         table.insert(buf, "}")
12510     elseif typ == "table" then
12511         table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12512         table.insert(buf, k)
12513         table.insert(buf, "}")
12514     else
12515         local error = self.error(format(
12516             "Unexpected type %s for value of "
12517             .. "YAML key %s.", typ, k))
12518         table.insert(buf, error)
12519     end
12520 end
12521 end
12522
12523 if is_sequence then
12524     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12525 else
12526     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12527 end
12528
12529 if not p then
12530     table.insert(buf, "\\markdownRendererJekyllDataEnd")
12531 end
12532
12533 return buf
12534 end
12535 end, extend_reader = function(self)
12536     local parsers = self.parsers
12537     local writer = self.writer
12538
12539     local JekyllData
12540     = Cmt( C((parsers.line - P("---") - P("..."))^0)
12541         , function(s, i, text) -- luacheck: ignore s i
12542             local data
12543             local ran_ok, _ = pcall(function()
12544                 local tinyyaml = require("tinyyaml")
12545                 data = tinyyaml.parse(text, {timestamps=false})
12546             end)
12547             if ran_ok and data ~= nil then
12548                 return true, writer.jekyllData(data, function(s)
12549                     return self.parser_functions.parse_blocks_nested(s)
12550                 end, nil)
12551             else
12552                 return false
12553             end
12554         end
12555     end

```

```

12555         )
12556
12557     local UnexpectedJekyllData
12558     = P("----")
12559     * parsers.blankline / 0
12560     -- if followed by blank, it's thematic break
12561     * #(-parsers.blankline)
12562     * JekyllData
12563     * (P("----") + P("..."))
12564
12565     local ExpectedJekyllData
12566     = ( P("----")
12567     * parsers.blankline / 0
12568     -- if followed by blank, it's thematic break
12569     * #(-parsers.blankline)
12570     )^-1
12571     * JekyllData
12572     * (P("----") + P("..."))^-1
12573
12574     if ensure_jekyll_data then
12575         ExpectedJekyllData = ExpectedJekyllData
12576         * parsers.eof
12577     else
12578         ExpectedJekyllData = ( ExpectedJekyllData
12579         * (V("Blank")^0 / writer.interblocksep)
12580         )^-1
12581     end
12582
12583     self.insert_pattern("Block before Blockquote",
12584         UnexpectedJekyllData, "UnexpectedJekyllData")
12585     if expect_jekyll_data then
12586         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12587     end
12588 end
12589 }
12590 end

```

### 3.1.8 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its own function `new` unless option `eagerCache` or `finalizeCache` has been enabled and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```
12591 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```
12592     options = options or {}
```



```

12593 setmetatable(options, { __index = function (_, key)
12594     return defaultOptions[key] end })

```

Return a conversion function that tries to produce a cached conversion output exists. If no cached conversion output exists, we load the file `markdown-parser.lua` and use it to convert the input.

```

12595 local parser_convert = nil
12596 return function(input)
12597     local function convert(input)
12598         if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

12599         local parser = require("markdown-parser")
12600         if metadata.version ~= parser.metadata.version then
12601             warn("markdown.lua " .. metadata.version .. " used with " ..
12602                 "markdown-parser.lua " .. parser.metadata.version .. ".")
12603         end
12604         parser_convert = parser.new(options)
12605     end
12606     return parser_convert(input)
12607 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain  $\TeX$  output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```

12608 local output
12609 if options.eagerCache or options.finalizeCache then
12610     local salt = util.salt(options)
12611     local name = util.cache(options.cacheDir, input, salt, convert,
12612                             ".md.tex")
12613     output = [[\input{}}] .. name .. [{}\relax]]

```

Otherwise, return the result of the conversion directly.

```

12614 else
12615     output = convert(input)
12616 end

```

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

```

12617 if options.finalizeCache then
12618     local file, mode
12619     if options.frozenCacheCounter > 0 then
12620         mode = "a"
12621     else

```

```

12622     mode = "w"
12623     end
12624     file = assert(io.open(options.frozenCacheFileName, mode),
12625     [[Could not open file ]] .. options.frozenCacheFileName
12626     .. [[ for writing]])
12627     assert(file:write(
12628     [[\expandafter\global\expandafter\def\csname ]]
12629     .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12630     .. [[\endcsname{}}] .. output .. [{}]] .. "\n"))
12631     assert(file:close())
12632     end
12633     return output
12634     end
12635 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain TeX output. See Section 2.1.1.

```
12636 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```

12637     options = options or {}
12638     setmetatable(options, { __index = function (_, key)
12639     return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12640     if options.singletonCache and singletonCache.convert then
12641     for k, v in pairs(defaultOptions) do
12642     if type(v) == "table" then
12643     for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12644     if singletonCache.options[k][i] ~= options[k][i] then
12645     goto miss
12646     end
12647     end

```

The `cacheDir` option is disregarded.

```

12648     elseif k ~= "cacheDir"
12649     and singletonCache.options[k] ~= options[k] then
12650     goto miss
12651     end
12652     end
12653     return singletonCache.convert
12654     end
12655     ::miss::

```

Apply built-in syntax extensions based on `options`.

```

12656     local extensions = {}
12657
12658     if options.bracketedSpans then
12659     local bracketed_spans_extension = M.extensions.bracketed_spans()

```

```

12660     table.insert(extensions, bracketed_spans_extension)
12661 end
12662
12663 if options.contentBlocks then
12664     local content_blocks_extension = M.extensions.content_blocks(
12665         options.contentBlocksLanguageMap)
12666     table.insert(extensions, content_blocks_extension)
12667 end
12668
12669 if options.definitionLists then
12670     local definition_lists_extension = M.extensions.definition_lists(
12671         options.tightLists)
12672     table.insert(extensions, definition_lists_extension)
12673 end
12674
12675 if options.fencedCode then
12676     local fenced_code_extension = M.extensions.fenced_code(
12677         options.blankBeforeCodeFence,
12678         options.fencedCodeAttributes,
12679         options.rawAttribute)
12680     table.insert(extensions, fenced_code_extension)
12681 end
12682
12683 if options.fencedDivs then
12684     local fenced_div_extension = M.extensions.fenced_divs(
12685         options.blankBeforeDivFence)
12686     table.insert(extensions, fenced_div_extension)
12687 end
12688
12689 if options.headerAttributes then
12690     local header_attributes_extension = M.extensions.header_attributes()
12691     table.insert(extensions, header_attributes_extension)
12692 end
12693
12694 if options.inlineCodeAttributes then
12695     local inline_code_attributes_extension =
12696         M.extensions.inline_code_attributes()
12697     table.insert(extensions, inline_code_attributes_extension)
12698 end
12699
12700 if options.jekyllData then
12701     local jekyll_data_extension = M.extensions.jekyll_data(
12702         options.expectJekyllData, options.ensureJekyllData)
12703     table.insert(extensions, jekyll_data_extension)
12704 end
12705
12706 if options.linkAttributes then

```

```

12707     local link_attributes_extension =
12708         M.extensions.link_attributes()
12709     table.insert(extensions, link_attributes_extension)
12710 end
12711
12712 if options.lineBlocks then
12713     local line_block_extension = M.extensions.line_blocks()
12714     table.insert(extensions, line_block_extension)
12715 end
12716
12717 if options.mark then
12718     local mark_extension = M.extensions.mark()
12719     table.insert(extensions, mark_extension)
12720 end
12721
12722 if options.pipeTables then
12723     local pipe_tables_extension = M.extensions.pipe_tables(
12724         options.tableCaptions, options.tableAttributes)
12725     table.insert(extensions, pipe_tables_extension)
12726 end
12727
12728 if options.rawAttribute then
12729     local raw_inline_extension = M.extensions.raw_inline()
12730     table.insert(extensions, raw_inline_extension)
12731 end
12732
12733 if options.strikeThrough then
12734     local strike_through_extension = M.extensions.strike_through()
12735     table.insert(extensions, strike_through_extension)
12736 end
12737
12738 if options.subscripts then
12739     local subscript_extension = M.extensions.subscripts()
12740     table.insert(extensions, subscript_extension)
12741 end
12742
12743 if options.superscripts then
12744     local superscript_extension = M.extensions.superscripts()
12745     table.insert(extensions, superscript_extension)
12746 end
12747
12748 if options.texMathDollars or
12749     options.texMathSingleBackslash or
12750     options.texMathDoubleBackslash then
12751     local tex_math_extension = M.extensions.tex_math(
12752         options.texMathDollars,
12753         options.texMathSingleBackslash,

```

```

12754     options.texMathDoubleBackslash)
12755     table.insert(extensions, tex_math_extension)
12756 end
12757
12758 if options.notes or options.inlineNotes then
12759     local notes_extension = M.extensions.notes(
12760         options.notes, options.inlineNotes)
12761     table.insert(extensions, notes_extension)
12762 end
12763
12764 if options.citations then
12765     local citations_extension
12766         = M.extensions.citations(options.citationNbsps)
12767     table.insert(extensions, citations_extension)
12768 end
12769
12770 if options.fancyLists then
12771     local fancy_lists_extension = M.extensions.fancy_lists()
12772     table.insert(extensions, fancy_lists_extension)
12773 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12774 for _, user_extension_filename in ipairs(options.extensions) do
12775     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

12776     local pathname = assert(kpse.find_file(filename),
12777         [[Could not locate user-defined syntax extension "]]
12778         .. filename)
12779     local input_file = assert(io.open(pathname, "r"),
12780         [[Could not open user-defined syntax extension "]]
12781         .. pathname .. [{" for reading}]]))
12782     local input = assert(input_file:read("*a"))
12783     assert(input_file:close())
12784     local user_extension, err = load([[
12785         local sandbox = {}
12786         setmetatable(sandbox, {__index = _G})
12787         _ENV = sandbox
12788     ]] .. input)()
12789     assert(user_extension,
12790         [[Failed to compile user-defined syntax extension "]]
12791         .. pathname .. [{": }]] .. (err or [{"}]))

```

Then, validate the user-defined syntax extension.

```

12792     assert(user_extension.api_version ~= nil,
12793         [[User-defined syntax extension "]] .. pathname
12794         .. [{" does not specify mandatory field "api_version"}]])
12795     assert(type(user_extension.api_version) == "number",

```

```

12796     [[User-defined syntax extension "]] .. pathname
12797     .. [[[" specifies field "api_version" of type "]]
12798     .. type(user_extension.api_version)
12799     .. [[[" but "number" was expected]])
12800     assert(user_extension.api_version > 0
12801            and user_extension.api_version
12802            <= metadata.user_extension_api_version,
12803            [[User-defined syntax extension "]] .. pathname
12804     .. [[[" uses syntax extension API version "]]
12805     .. user_extension.api_version .. [[ but markdown.lua ]]
12806     .. metadata.version .. [[ uses API version ]]
12807     .. metadata.user_extension_api_version
12808     .. [[[, which is incompatible]])
12809
12810     assert(user_extension.grammar_version ~= nil,
12811            [[User-defined syntax extension "]] .. pathname
12812     .. [[[" does not specify mandatory field "grammar_version"]])
12813     assert(type(user_extension.grammar_version) == "number",
12814            [[User-defined syntax extension "]] .. pathname
12815     .. [[[" specifies field "grammar_version" of type "]]
12816     .. type(user_extension.grammar_version)
12817     .. [[[" but "number" was expected]])
12818     assert(user_extension.grammar_version == metadata.grammar_version,
12819            [[User-defined syntax extension "]] .. pathname
12820     .. [[[" uses grammar version "]]
12821     .. user_extension.grammar_version
12822     .. [[ but markdown.lua ]] .. metadata.version
12823     .. [[ uses grammar version ]] .. metadata.grammar_version
12824     .. [[[, which is incompatible]])
12825
12826     assert(user_extension.finalize_grammar ~= nil,
12827            [[User-defined syntax extension "]] .. pathname
12828     .. [[[" does not specify mandatory "finalize_grammar" field]])
12829     assert(type(user_extension.finalize_grammar) == "function",
12830            [[User-defined syntax extension "]] .. pathname
12831     .. [[[" specifies field "finalize_grammar" of type "]]
12832     .. type(user_extension.finalize_grammar)
12833     .. [[[" 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.)

```

12834     local extension = {
12835         name = [[user-defined "]] .. pathname .. [[[" syntax extension]],
12836         extend_reader = user_extension.finalize_grammar,
12837         extend_writer = function() end,
12838     }
12839     return extension

```

```

12840     end)(user_extension_filename)
12841     table.insert(extensions, user_extension)
12842 end

```

Produce a conversion function from markdown to plain TeX.

```

12843 local writer = M.writer.new(options)
12844 local reader = M.reader.new(writer, options)
12845 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`.

```

12846 collectgarbage("collect")

```

Update the singleton cache.

```

12847 if options.singletonCache then
12848     local singletonCacheOptions = {}
12849     for k, v in pairs(options) do
12850         singletonCacheOptions[k] = v
12851     end
12852     setmetatable(singletonCacheOptions,
12853         { __index = function (_, key)
12854             return defaultOptions[key] end })
12855     singletonCache.options = singletonCacheOptions
12856     singletonCache.convert = convert
12857 end

```

Return the conversion function from markdown to plain TeX.

```

12858 return convert
12859 end
12860 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.

```

12861
12862 local input
12863 if input_filename then
12864     local input_file = assert(io.open(input_filename, "r"),
12865         [[Could not open file "]] .. input_filename .. [[ for reading]])
12866     input = assert(input_file:read("*a"))
12867     assert(input_file:close())
12868 else
12869     input = assert(io.read("*a"))
12870 end
12871

```

First, ensure that the `options.cacheDir` directory exists.

```
12872 local lfs = require("lfs")
12873 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12874     assert(lfs.mkdir(options["cacheDir"]))
12875 end
```

If Kpathsea has not been loaded before or if Lua<sub>TEX</sub> has not yet been initialized, configure Kpathsea on top of loading it.

```
12876 local kpse
12877 (function()
12878     local should_initialize = package.loaded.kpse == nil
12879                             or tex.initialize ~= nil
12880     kpse = require("kpse")
12881     if should_initialize then
12882         kpse.set_program_name("luatex")
12883     end
12884 end)()
12885 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.

```
12886 if metadata.version ~= md.metadata.version then
12887     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12888         "markdown.lua " .. md.metadata.version .. ".")
12889 end
12890 local convert = md.new(options)
12891 local output = convert(input)
12892
12893 if output_filename then
12894     local output_file = assert(io.open(output_filename, "w"),
12895         [[Could not open file ]] .. output_filename .. [[ for writing]])
12896     assert(output_file:write(output))
12897     assert(output_file:close())
12898 else
12899     assert(io.write(output))
12900 end
```

Remove the `options.cacheDir` directory if it is empty.

```
12901 if options.cacheDir then
12902     lfs.rmdir(options.cacheDir)
12903 end
```

### 3.2 Plain <sub>TEX</sub> Implementation

The plain <sub>TEX</sub> implementation provides macros for the interfacing between <sub>TEX</sub> 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 <sub>TEX</sub> exposed by the plain <sub>TEX</sub> interface (see Section 2.2).



### 3.2.1 Logging Facilities

```
12904 \ExplSyntaxOn
12905 \cs_if_free:NT
12906   \markdownInfo
12907   {
12908     \cs_new:Npn
12909       \markdownInfo #1
12910       {
12911         \msg_info:nne
12912           { markdown }
12913           { generic-message }
12914           { #1 }
12915       }
12916   }
12917 \cs_if_free:NT
12918   \markdownWarning
12919   {
12920     \cs_new:Npn
12921       \markdownWarning #1
12922       {
12923         \msg_warning:nne
12924           { markdown }
12925           { generic-message }
12926           { #1 }
12927       }
12928   }
12929 \cs_if_free:NT
12930   \markdownError
12931   {
12932     \cs_new:Npn
12933       \markdownError #1 #2
12934       {
12935         \msg_error:nnee
12936           { markdown }
12937           { generic-message-with-help-text }
12938           { #1 }
12939           { #2 }
12940       }
12941   }
12942 \msg_new:nnn
12943   { markdown }
12944   { generic-message }
12945   { #1 }
12946 \msg_new:nnnn
12947   { markdown }
12948   { generic-message-with-help-text }
```

```

12949 { #1 }
12950 { #2 }
12951 \cs_generate_variant:Nn
12952 \msg_info:nnn
12953 { nne }
12954 \cs_generate_variant:Nn
12955 \msg_warning:nnn
12956 { nne }
12957 \cs_generate_variant:Nn
12958 \msg_error:nnnn
12959 { nnee }
12960 \ExplSyntaxOff

```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain T<sub>E</sub>X themes provided with the Markdown package.

```

12961 \ExplSyntaxOn
12962 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
12963 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
12964 \cs_new:Nn
12965 \@@_plain_tex_load_theme:nnn
12966 {
12967   \prop_get:NnNTF
12968     \g_@@_plain_tex_loaded_themes_linenos_prop
12969     { #1 }
12970     \l_tmpa_tl
12971     {
12972       \prop_get:NnN
12973         \g_@@_plain_tex_loaded_themes_versions_prop
12974         { #1 }
12975         \l_tmpb_tl
12976       \str_if_eq:nVTF
12977         { #2 }
12978         \l_tmpb_tl
12979         {
12980           \msg_warning:nnnVn
12981             { markdown }
12982             { repeatedly-loaded-plain-tex-theme }
12983             { #1 }
12984           \l_tmpa_tl
12985           { #2 }
12986         }
12987       {
12988         \msg_error:nnnnVV
12989         { markdown }

```

```

12990         { different-versions-of-plain-tex-theme }
12991         { #1 }
12992         { #2 }
12993         \l_tmpb_tl
12994         \l_tmpa_tl
12995     }
12996 }
12997 {
12998     \prop_gput:Nnx
12999     \g_@@_plain_tex_loaded_themes_linenos_prop
13000     { #1 }
13001     { \tex_the:D \tex_inputlineno:D }
13002     \prop_gput:Nnn
13003     \g_@@_plain_tex_loaded_themes_versions_prop
13004     { #1 }
13005     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

13006     \prop_if_in:NnTF
13007     \g_@@_plain_tex_built_in_themes_prop
13008     { #1 }
13009     {
13010         \msg_info:nnnn
13011         { markdown }
13012         { loading-built-in-plain-tex-theme }
13013         { #1 }
13014         { #2 }
13015     }
13016     \prop_item:Nn
13017     \g_@@_plain_tex_built_in_themes_prop
13018     { #1 }
13019 }
13020     \msg_info:nnnn
13021     { markdown }
13022     { loading-plain-tex-theme }
13023     { #1 }
13024     { #2 }
13025     \file_input:n
13026     { markdown theme #3 }
13027 }
13028 }
13029 }
13030 \msg_new:nnn
13031 { markdown }
13032 { loading-plain-tex-theme }
13033 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }

```

```

13034 \msg_new:nnn
13035   { markdown }
13036   { loading-built-in-plain-tex-theme }
13037   { Loading~version~#2~of~built-in-plain-TeX-Markdown~theme~#1 }
13038 \msg_new:nnn
13039   { markdown }
13040   { repeatedly-loaded-plain-tex-theme }
13041   {
13042     Version~#3~of~plain-TeX-Markdown~theme~#1~was~previously~
13043     loaded~on~line~#2,~not~loading~it~again
13044   }
13045 \msg_new:nnn
13046   { markdown }
13047   { different-versions-of-plain-tex-theme }
13048   {
13049     Tried~to~load~version~#2~of~plain-TeX-Markdown~theme~#1~
13050     but~version~#3~has~already~been~loaded~on~line~#4
13051   }
13052 \cs_generate_variant:Nn
13053   \prop_gput:Nnn
13054   { Nnx }
13055 \cs_gset_eq:NN
13056   \@@_load_theme:nnn
13057   \@@_plain_tex_load_theme:nnn
13058 \cs_generate_variant:Nn
13059   \@@_load_theme:nnn
13060   { VeV }
13061 \cs_generate_variant:Nn
13062   \msg_error:nnnnn
13063   { nnnnVV }
13064 \cs_generate_variant:Nn
13065   \msg_warning:nnnnn
13066   { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain T<sub>E</sub>X theme from within themes for higher-level T<sub>E</sub>X formats such as L<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

13067 \cs_new:Npn
13068   \markdownLoadPlainTeXTheme
13069   {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

13070   \tl_set:NV
13071     \l_tmpa_tl
13072     \g_@@_current_theme_tl
13073   \tl_reverse:N
13074     \l_tmpa_tl

```

```

13075 \tl_set:Ne
13076   \l_tmpb_tl
13077   {
13078     \tl_tail:V
13079     \l_tmpa_tl
13080   }
13081 \tl_reverse:N
13082   \l_tmpb_tl

```

Next, we munge the theme name.

```

13083 \str_set:NV
13084   \l_tmpa_str
13085   \l_tmpb_tl
13086 \str_replace_all:Nnn
13087   \l_tmpa_str
13088   { / }
13089   { _ }

```

Finally, we load the plain  $\TeX$  theme.

```

13090 \@@_plain_tex_load_theme:VeV
13091   \l_tmpb_tl
13092   { \markdownThemeVersion }
13093   \l_tmpa_str
13094 }
13095 \cs_generate_variant:Nn
13096   \tl_set:Nn
13097   { Ne }
13098 \cs_generate_variant:Nn
13099   \@@_plain_tex_load_theme:nnn
13100   { VeV }

```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```

13101 \prop_gput:Nnn
13102   \g_@@_plain_tex_built_in_themes_prop
13103   { witiko / dot }
13104   {
13105     \str_if_eq:enF
13106       { \markdownThemeVersion }
13107       { silent }
13108       {
13109         \markdownWarning
13110         {
13111           The~theme~name~"witiko/dot"~has~been~soft-deprecated.
13112           \iow_newline:
13113           Consider~changing~the~name~to~"witiko/diagrams@v1".
13114         }
13115       }

```

We enable the `fencedCode` Lua option.

```
13116 \markdownSetup{fencedCode}
```

We store the previous definition of the fenced code token renderer prototype:

```
13117 \cs_set_eq:NN
13118 \c_@@_dot_previous_definition:nmn
13119 \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T<sub>E</sub>X option is disabled and the code block has not been previously typeset:

```
13120 \regex_const:Nn
13121 \c_@@_dot_infostring_regex
13122 { ^dot(\s+(.+))? }
13123 \seq_new:N
13124 \l_@@_dot_matches_seq
13125 \markdownSetup {
13126   rendererPrototypes = {
13127     inputFencedCode = {
13128       \regex_extract_once:NnNTF
13129       \c_@@_dot_infostring_regex
13130       { #2 }
13131       \l_@@_dot_matches_seq
13132       {
13133         \@@_if_option:nF
13134         { frozenCache }
13135         {
13136           \sys_shell_now:n
13137           {
13138             if~!~test~-e~#1.pdf.source~
13139             ||~!~diff~#1~#1.pdf.source;
13140             then~
13141               dot~-Tpdf~-o~#1.pdf~#1;
13142               cp~#1~#1.pdf.source;
13143             fi
13144           }
13145         }
13146       }
13147     }
13148   }
13149 }
```

We include the typeset image using the image token renderer:

```
13146 \exp_args:NNne
13147 \exp_last_unbraced:No
13148 \markdownRendererImage
13149 {
13150   { Graphviz~image }
13151   { #1.pdf }
13152   { #1.pdf }
13153 }
```

```

13154         {
13155             \seq_item:Nn
13156             \l_@@_dot_matches_seq
13157             { 3 }
13158         }
13159     }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

13160     {
13161         \@@_dot_previous_definition:nnn
13162         { #1 }
13163         { #2 }
13164         { #3 }
13165     }
13166 },
13167 },
13168 }
13169 }

```

The `witiko/diagrams` loads the theme `witiko/dot`.

```

13170 \prop_gput:Nnn
13171 \g_@@_plain_tex_built_in_themes_prop
13172 { witiko / diagrams }
13173 {
13174     \str_case:enF
13175     { \markdownThemeVersion }
13176     {
13177         { latest }
13178         {
13179             \markdownWarning
13180             {
13181                 Write~"witiko/diagrams@v1"~to~pin~version~"v1"~of~the~
13182                 theme~"witiko/diagrams".~This~will~keep~your~documents~
13183                 from~suddenly~breaking~when~we~have~released~future~
13184                 versions~of~the~theme~with~backwards~incompatible~
13185                 syntax~and~behavior.
13186             }
13187             \markdownSetup
13188             {
13189                 import = witiko/dot@silent,
13190             }
13191         }
13192         { v1 }
13193         {
13194             \markdownSetup
13195             {
13196                 import = witiko/dot@silent,

```

```

13197         }
13198     }
13199 }
13200 {
13201     \msg_error:nnen
13202     { markdown }
13203     { unknown-theme-version }
13204     { witiko/diagrams }
13205     { \markdownThemeVersion }
13206     { v1 }
13207 }
13208 }
13209 \cs_generate_variant:Nn
13210 \msg_error:nnnnn
13211 { nnnen }
13212 \msg_new:nnnn
13213 { markdown }
13214 { unknown-theme-version }
13215 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
13216 { Known~versions~are:~#3 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

13217 \group_begin:
13218 \char_set_catcode_other:N \%

```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```

13219 \prop_gput:Nnn
13220 \g_@@_plain_tex_built_in_themes_prop
13221 { witiko / graphicx / http }
13222 {
13223     \cs_set_eq:NN
13224     \@@_graphicx_http_previous_definition:nnnn
13225     \markdownRendererImagePrototype

```

We define variables and functions to enumerate the images for caching and to store the pathname of the file containing the pathname of the downloaded image file.

```

13226     \int_new:N
13227     \g_@@_graphicx_http_image_number_int
13228     \int_gset:Nn
13229     \g_@@_graphicx_http_image_number_int
13230     { 0 }
13231     \cs_new:Nn
13232     \@@_graphicx_http_filename:
13233     {
13234         \markdownOptionCacheDir
13235         / witiko_graphicx_http .

```



```

13236     \int_use:N
13237     \g_@@_graphicx_http_image_number_int
13238 }

```

We define a function that will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The function produces a shell command that tries to download the online image to the pathname.

```

13239     \cs_new:Nn
13240     \@@_graphicx_http_download:nn
13241     {
13242     wget~-0~#2~#1~
13243     ||~curl~---location~-o~#2~#1~
13244     ||~rm~-f~#2
13245     }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

13246     \str_new:N
13247     \l_@@_graphicx_http_filename_str
13248     \ior_new:N
13249     \g_@@_graphicx_http_filename_ior
13250     \markdownSetup {
13251     rendererPrototypes = {
13252     image = {
13253     \@@_if_option:nF
13254     { frozenCache }
13255     {

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```

13256     \sys_shell_now:e
13257     {
13258     mkdir~-p~" \markdownOptionCacheDir ";
13259     if~printf~'%s'~"#3"~|~grep~-q~-E~'^https?:';
13260     then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

13261     OUTPUT_PREFIX=" \markdownOptionCacheDir ";
13262     OUTPUT_BODY="$(printf~'%s'~'#3'
13263     |~md5sum~|~cut~-d~'|'~-f1)";
13264     OUTPUT_SUFFIX="$(printf~'%s'~'#3'
13265     |~sed~'s/.*[.]//)";
13266     OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

13267     if~!~[~-e~"$OUTPUT"~];

```

```

13268         then~
13269             \@@_graphicx_http_download:nn
13270             { '#3' }
13271             { "$OUTPUT" } ;
13272             printf~'%s'~"$OUTPUT"~
13273             >~" \@@_graphicx_http_filename: ";
13274         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:

```

13275         else~
13276             printf~'%s'~'#3'~
13277             >~" \@@_graphicx_http_filename: ";
13278         fi
13279     }
13280 }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

13281     \ior_open:Ne
13282     \g_@@_graphicx_http_filename_ior
13283     { \@@_graphicx_http_filename: }
13284     \ior_str_get:NN
13285     \g_@@_graphicx_http_filename_ior
13286     \l_@@_graphicx_http_filename_str
13287     \ior_close:N
13288     \g_@@_graphicx_http_filename_ior
13289     \@@_graphicx_http_previous_definition:nnVn
13290     { #1 }
13291     { #2 }
13292     \l_@@_graphicx_http_filename_str
13293     { #4 }
13294     \int_gincr:N
13295     \g_@@_graphicx_http_image_number_int
13296     }
13297 }
13298 }
13299 \cs_generate_variant:Nn
13300 \ior_open:Nn
13301 { Ne }
13302 \cs_generate_variant:Nn
13303 \@@_graphicx_http_previous_definition:nnnn
13304 { nnVn }
13305 }
13306 \group_end:

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

13307 \prop_gput:Nnn
13308 \g_@@_plain_tex_built_in_themes_prop
13309 { witiko / tilde }
13310 {
13311   \markdownSetup {
13312     rendererPrototypes = {
13313       tilde = {-},
13314     },
13315   }
13316 }

```

The themes `witiko/example/foo` and `witiko/example/bar` are supposed to be used in code examples. They don't do anything.

```

13317 \clist_map_inline:nn
13318 { foo, bar }
13319 {
13320   \prop_gput:Nnn
13321   \g_@@_plain_tex_built_in_themes_prop
13322   { witiko / example / #1 }
13323   {
13324     \markdownWarning
13325     {
13326       The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
13327       examples.~Using~it~in~actual~code~has~no~effect,~except~
13328       this~warning~message,~and~is~usually~a~mistake.
13329     }
13330   }
13331 }
13332 \ExplSyntaxOff

```

The `witiko/markdown/defaults` plain T<sub>E</sub>X theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

13333 \def\markdownRendererInterblockSeparatorPrototype{\par}%
13334 \def\markdownRendererParagraphSeparatorPrototype{%
13335   \markdownRendererInterblockSeparator}%
13336 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
13337 \def\markdownRendererSoftLineBreakPrototype{ }%
13338 \let\markdownRendererEllipsisPrototype\dots
13339 \def\markdownRendererNbspPrototype{~}%
13340 \def\markdownRendererLeftBracePrototype{\char`}%
13341 \def\markdownRendererRightBracePrototype{\char`}%
13342 \def\markdownRendererDollarSignPrototype{\char`}%
13343 \def\markdownRendererPercentSignPrototype{\char`}%
13344 \def\markdownRendererAmpersandPrototype{\&}%

```



```

13392 \def\markdownRendererEmphasisPrototype#1{\it#1}%
13393 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13394 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
13395 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
13396 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=Opt}%
13397 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
13398 \def\markdownRendererInputVerbatimPrototype#1{%
13399   \par{\tt\input#1\relax{}}\par}%
13400 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
13401   \markdownRendererInputVerbatim{#1}}%
13402 \def\markdownRendererHeadingOnePrototype#1{#1}%
13403 \def\markdownRendererHeadingTwoPrototype#1{#1}%
13404 \def\markdownRendererHeadingThreePrototype#1{#1}%
13405 \def\markdownRendererHeadingFourPrototype#1{#1}%
13406 \def\markdownRendererHeadingFivePrototype#1{#1}%
13407 \def\markdownRendererHeadingSixPrototype#1{#1}%
13408 \def\markdownRendererThematicBreakPrototype{}%
13409 \def\markdownRendererNotePrototype#1{#1}%
13410 \def\markdownRendererCitePrototype#1{}%
13411 \def\markdownRendererTextCitePrototype#1{}%
13412 \def\markdownRendererTickedBoxPrototype{[X]}%
13413 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
13414 \def\markdownRendererUntickedBoxPrototype{[ ]}%
13415 \def\markdownRendererStrikeThroughPrototype#1{#1}%
13416 \def\markdownRendererSuperscriptPrototype#1{#1}%
13417 \def\markdownRendererSubscriptPrototype#1{#1}%
13418 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
13419 \def\markdownRendererInlineMathPrototype#1{ $#1$}%
13420 \ExplSyntaxOn
13421 \cs_gset:Npn
13422   \markdownRendererHeaderAttributeContextBeginPrototype
13423   {
13424     \group_begin:
13425     \color_group_begin:
13426   }
13427 \cs_gset:Npn
13428   \markdownRendererHeaderAttributeContextEndPrototype
13429   {
13430     \color_group_end:
13431     \group_end:
13432   }
13433 \cs_gset_eq:NN
13434   \markdownRendererBracketedSpanAttributeContextBeginPrototype
13435   \markdownRendererHeaderAttributeContextBeginPrototype
13436 \cs_gset_eq:NN
13437   \markdownRendererBracketedSpanAttributeContextEndPrototype
13438   \markdownRendererHeaderAttributeContextEndPrototype

```

```

13439 \cs_gset_eq:NN
13440   \markdownRendererFencedDivAttributeContextBeginPrototype
13441   \markdownRendererHeaderAttributeContextBeginPrototype
13442 \cs_gset_eq:NN
13443   \markdownRendererFencedDivAttributeContextEndPrototype
13444   \markdownRendererHeaderAttributeContextEndPrototype
13445 \cs_gset_eq:NN
13446   \markdownRendererFencedCodeAttributeContextBeginPrototype
13447   \markdownRendererHeaderAttributeContextBeginPrototype
13448 \cs_gset_eq:NN
13449   \markdownRendererFencedCodeAttributeContextEndPrototype
13450   \markdownRendererHeaderAttributeContextEndPrototype
13451 \cs_gset:Npn
13452   \markdownRendererReplacementCharacterPrototype
13453   { \codepoint_str_generate:n { fffd } }
13454 \ExplSyntaxOff
13455 \def\markdownRendererSectionBeginPrototype{}%
13456 \def\markdownRendererSectionEndPrototype{}%
13457 \ExplSyntaxOn
13458 \cs_gset:Npn
13459   \markdownRendererWarningPrototype
13460   #1#2#3#4
13461   {
13462     \tl_set:Nn
13463       \l_tmpa_tl
13464       { #2 }
13465     \tl_if_empty:nF
13466       { #4 }
13467     {
13468       \tl_put_right:Nn
13469         \l_tmpa_tl
13470         { \iow_newline: #4 }
13471     }
13472     \exp_args:NV
13473       \markdownWarning
13474       \l_tmpa_tl
13475   }
13476 \ExplSyntaxOff
13477 \def\markdownRendererErrorPrototype#1#2#3#4{%
13478   \markdownError{#2}{#4}}%

```

### 3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

13479 \ExplSyntaxOn

```

```

13480 \cs_new:Nn
13481   \@@_plain_tex_default_input_raw_inline:nn
13482   {
13483     \str_case:nn
13484       { #2 }
13485       {
13486         { md } { \markdownInput{#1} }
13487         { tex } { \markdownEscape{#1} \unskip }
13488       }
13489   }
13490 \cs_new:Nn
13491   \@@_plain_tex_default_input_raw_block:nn
13492   {
13493     \str_case:nn
13494       { #2 }
13495       {
13496         { md } { \markdownInput{#1} }
13497         { tex } { \markdownEscape{#1} }
13498       }
13499   }
13500 \cs_gset:Npn
13501   \markdownRendererInputRawInlinePrototype#1#2
13502   {
13503     \@@_plain_tex_default_input_raw_inline:nn
13504       { #1 }
13505       { #2 }
13506   }
13507 \cs_gset:Npn
13508   \markdownRendererInputRawBlockPrototype#1#2
13509   {
13510     \@@_plain_tex_default_input_raw_block:nn
13511       { #1 }
13512       { #2 }
13513   }
13514 \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_t1` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```
13515 \ExplSyntaxOn
13516 \seq_new:N \g_@@_jekyll_data_datatypes_seq
13517 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
13518 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
13519 \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.

```
13520 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
13521 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
13522 {
13523   \seq_if_empty:NF
13524     \g_@@_jekyll_data_datatypes_seq
13525     {
13526       \seq_get_right:NN
13527         \g_@@_jekyll_data_datatypes_seq
13528         \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.

```
13529   \str_if_eq:NNTF
13530     \l_tmpa_tl
13531     \c_@@_jekyll_data_sequence_tl
13532     {
13533       \seq_put_right:Nn
13534         \g_@@_jekyll_data_wildcard_absolute_address_seq
13535         { * }
13536     }
13537     {
13538       \seq_put_right:Nn
13539         \g_@@_jekyll_data_wildcard_absolute_address_seq
13540         { #1 }
13541     }
13542   }
13543 }
```

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.

```
13544 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
13545 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
13546 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
13547 {
13548   \seq_pop_left:NN #1 \l_tmpa_tl
13549   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
13550   \seq_put_left:NV #1 \l_tmpa_tl
13551 }
13552 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
13553 {
13554   \markdown_jekyll_data_concatenate_address:NN
13555   \g_@@_jekyll_data_wildcard_absolute_address_seq
13556   \g_@@_jekyll_data_wildcard_absolute_address_tl
13557   \seq_get_right:NN
13558   \g_@@_jekyll_data_wildcard_absolute_address_seq
13559   \g_@@_jekyll_data_wildcard_relative_address_tl
13560 }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:n` and `\markdown_jekyll_data_pop:` macros.

```
13561 \cs_new:Nn \markdown_jekyll_data_push:n
13562 {
13563   \markdown_jekyll_data_push_address_segment:n
```

```

13564     { #1 }
13565     \seq_put_right:NV
13566     \g_@@_jekyll_data_datatypes_seq
13567     #2
13568     \markdown_jekyll_data_update_address_tls:
13569   }
13570 \cs_new:Nn \markdown_jekyll_data_pop:
13571 {
13572   \seq_pop_right:NN
13573   \g_@@_jekyll_data_wildcard_absolute_address_seq
13574   \l_tmpa_tl
13575   \seq_pop_right:NN
13576   \g_@@_jekyll_data_datatypes_seq
13577   \l_tmpa_tl
13578   \markdown_jekyll_data_update_address_tls:
13579 }

```

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.

```

13580 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
13581 {
13582   \keys_set_known:nn
13583   { markdown/jekyllData }
13584   { { #1 } = { #2 } }
13585 }
13586 \cs_generate_variant:Nn
13587   \markdown_jekyll_data_set_keyval:nn
13588   { Vn }
13589 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
13590 {
13591   \markdown_jekyll_data_push:nN
13592   { #1 }
13593   \c_@@_jekyll_data_scalar_tl
13594   \markdown_jekyll_data_set_keyval:Vn
13595   \g_@@_jekyll_data_wildcard_absolute_address_tl
13596   { #2 }
13597   \markdown_jekyll_data_set_keyval:Vn
13598   \g_@@_jekyll_data_wildcard_relative_address_tl
13599   { #2 }
13600   \markdown_jekyll_data_pop:
13601 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

13602 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13603   \markdown_jekyll_data_push:nN
13604   { #1 }

```

```

13605     \c_@@_jekyll_data_sequence_tl
13606 }
13607 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13608     \markdown_jekyll_data_push:nN
13609     { #1 }
13610     \c_@@_jekyll_data_mapping_tl
13611 }
13612 \def\markdownRendererJekyllDataSequenceEndPrototype{
13613     \markdown_jekyll_data_pop:
13614 }
13615 \def\markdownRendererJekyllDataMappingEndPrototype{
13616     \markdown_jekyll_data_pop:
13617 }
13618 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13619     \markdown_jekyll_data_set_keyvals:nn
13620     { #1 }
13621     { #2 }
13622 }
13623 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13624 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13625     \markdown_jekyll_data_set_keyvals:nn
13626     { #1 }
13627     { #2 }
13628 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13629 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13630 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13631     \markdown_jekyll_data_set_keyvals:nn
13632     { #1 }
13633     { #2 }
13634 }
13635 \ExplSyntaxOff

```

If plain  $\text{T}_{\text{E}}\text{X}$  is the top layer, we load the [witiko/markdown/defaults](#) plain  $\text{T}_{\text{E}}\text{X}$  theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

13636 \ExplSyntaxOn
13637 \str_if_eq:VVT
13638     \c_@@_top_layer_tl
13639     \c_@@_option_layer_plain_tex_tl
13640     {
13641         \ExplSyntaxOff
13642         \@@_if_option:nF
13643             { noDefaults }
13644         {
13645             \@@_if_option:nTF

```

```

13646         { experimental }
13647     {
13648         \@@_setup:n
13649         { theme = witiko/markdown/defaults@experimental }
13650     }
13651     {
13652         \@@_setup:n
13653         { theme = witiko/markdown/defaults }
13654     }
13655 }
13656 \ExplSyntaxOn
13657 }
13658 \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).

```

13659 \ExplSyntaxOn
13660 \tl_new:N \g_@@_formatted_lua_options_tl
13661 \cs_new:Nn \@@_format_lua_options:
13662 {
13663     \tl_gclear:N
13664     \g_@@_formatted_lua_options_tl
13665     \seq_map_function:NN
13666     \g_@@_lua_options_seq
13667     \@@_format_lua_option:n
13668 }
13669 \cs_new:Nn \@@_format_lua_option:n
13670 {
13671     \@@_typecheck_option:n
13672     { #1 }
13673     \@@_get_option_type:nN
13674     { #1 }
13675     \l_tmpa_tl
13676     \bool_case_true:nF
13677     {
13678         {
13679             \str_if_eq_p:VV
13680             \l_tmpa_tl
13681             \c_@@_option_type_boolean_tl ||
13682             \str_if_eq_p:VV
13683             \l_tmpa_tl
13684             \c_@@_option_type_number_tl ||
13685             \str_if_eq_p:VV

```

```

13686         \l_tmpa_tl
13687         \c_@@_option_type_counter_tl
13688     }
13689     {
13690         \@@_get_option_value:nN
13691         { #1 }
13692         \l_tmpa_tl
13693         \tl_gput_right:Nx
13694         \g_@@_formatted_lua_options_tl
13695         { #1~::~ \l_tmpa_tl ,~ }
13696     }
13697     {
13698         \str_if_eq_p:VV
13699         \l_tmpa_tl
13700         \c_@@_option_type_clist_tl
13701     }
13702     {
13703         \@@_get_option_value:nN
13704         { #1 }
13705         \l_tmpa_tl
13706         \tl_gput_right:Nx
13707         \g_@@_formatted_lua_options_tl
13708         { #1~::~\c_left_brace_str }
13709         \clist_map_inline:Vn
13710         \l_tmpa_tl
13711         {
13712             \@@_lua_escape:xN
13713             { ##1 }
13714             \l_tmpb_tl
13715             \tl_gput_right:Nn
13716             \g_@@_formatted_lua_options_tl
13717             { " }
13718             \tl_gput_right:NV
13719             \g_@@_formatted_lua_options_tl
13720             \l_tmpb_tl
13721             \tl_gput_right:Nn
13722             \g_@@_formatted_lua_options_tl
13723             { " ,~ }
13724         }
13725         \tl_gput_right:Nx
13726         \g_@@_formatted_lua_options_tl
13727         { \c_right_brace_str ,~ }
13728     }
13729 }
13730 {
13731     \@@_get_option_value:nN
13732     { #1 }

```

```

13733     \l_tmpa_tl
13734     \@@_lua_escape:xN
13735     { \l_tmpa_tl }
13736     \l_tmpb_tl
13737     \tl_gput_right:Nn
13738     \g_@@_formatted_lua_options_tl
13739     { #1~::~ " }
13740     \tl_gput_right:NV
13741     \g_@@_formatted_lua_options_tl
13742     \l_tmpb_tl
13743     \tl_gput_right:Nn
13744     \g_@@_formatted_lua_options_tl
13745     { " ,~ }
13746   }
13747 }
13748 \cs_generate_variant:Nn
13749   \clist_map_inline:nn
13750   { Vn }
13751 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
13752 \def\markdownLuaOptions#{ \g_@@_formatted_lua_options_tl }
13753 \sys_if_engine luatex:TF
13754 {
13755   \cs_new:Nn
13756     \@@_lua_escape:nN
13757     {
13758       \tl_set:Nx
13759         #2
13760         {
13761           \lua_escape:n
13762             { #1 }
13763         }
13764     }
13765 }
13766 {
13767   \regex_const:Nn
13768     \c_@@_lua_escape_regex
13769     { [\\"' ] }
13770   \cs_new:Nn
13771     \@@_lua_escape:nN
13772     {
13773       \tl_set:Nn
13774         #2
13775         { #1 }
13776       \regex_replace_all:NnN
13777         \c_@@_lua_escape_regex
13778         { \u { c_backslash_str } \0 }
13779         #2

```

```

13780     }
13781   }
13782 \cs_generate_variant:Nn
13783   \@@_lua_escape:nN
13784   { xN }

```

After the `\markdownPrepareInputFilename` macro has been fully expanded, the `\markdownInputFilename` macro will expand to a Lua string that contains the input filename passed as the first argument.

```

13785 \tl_new:N
13786   \markdownInputFilename
13787 \cs_new:Npn
13788   \markdownPrepareInputFilename
13789   #1
13790   {
13791     \@@_lua_escape:xN
13792     { #1 }
13793     \markdownInputFilename
13794     \tl_gset:Nx
13795     \markdownInputFilename
13796     { " \markdownInputFilename " }
13797   }

```

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.

```

13798 \cs_new:Npn
13799   \markdownPrepare
13800   {

```

First, ensure that the `cacheDir` directory exists.

```

13801     local~lfs = require("lfs")
13802     local~options = \markdownLuaOptions
13803     if~not~lfs.isdir(options.cacheDir) then~
13804       assert(lfs.mkdir(options.cacheDir))
13805     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.

```

13806     local~md = require("markdown")
13807     local~convert = md.new(options)
13808   }

```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain  $\TeX$ . It opens the input file, converts it, and prints the conversion result.

```

13809 \cs_new:Npn
13810   \markdownConvert

```

```

13811 {
13812   local~filename = \markdownInputFilename
13813   local~file = assert(io.open(filename, "r"),
13814     [[Could~not~open~file~]] .. filename .. [[~for~reading]])
13815   local~input = assert(file:read("*a"))
13816   assert(file:close())
13817   print(convert(input))
13818 }
13819 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain  $\text{\TeX}$ .

```

13820 \def\markdownCleanup{%

```

Remove the `options.cacheDir` directory if it is empty.

```

13821   if options.cacheDir then
13822     lfs.rmdir(options.cacheDir)
13823   end
13824 }%

```

### 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.

```

13825 \csname newread\endcsname\markdownInputFileStream
13826 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

13827 \begingroup
13828   \catcode\^^I=12%
13829   \gdef\markdownReadAndConvertTab{^^I}%
13830 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}\epsilon$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

13831 \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.

```

13832   \catcode\^^M=13%
13833   \catcode\^^I=13%
13834   \catcode`|=0%
13835   \catcode`\=12%
13836   |catcode`@=14%

```



```

13837 |catcode`|=12@
13838 |gdef|markdownReadAndConvert#1#2{@
13839 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

13840 |markdownIfOption{frozenCache}{-}{@
13841 |immediate|openout|markdownOutputFileStream@
13842 |markdownOptionInputTempFileName|relax@
13843 |markdownInfo{@
13844 Buffering block-level markdown input into the temporary @
13845 input file "|markdownOptionInputTempFileName" and scanning @
13846 for the closing token sequence "#1"}@
13847 }@

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

13848 |def|do##1{|catcode`##1=12}|dospecials@
13849 |catcode`| =12@
13850 |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.

```

13851 |def|markdownReadAndConvertStripPercentSign##1{@
13852 |markdownIfOption{stripPercentSigns}{-}{@
13853 |if##1%@
13854 |expandafter|expandafter|expandafter@
13855 |markdownReadAndConvertProcessLine@
13856 |else@
13857 |expandafter|expandafter|expandafter@
13858 |markdownReadAndConvertProcessLine@
13859 |expandafter|expandafter|expandafter##1@
13860 |fi@
13861 }{-}{@
13862 |expandafter@
13863 |markdownReadAndConvertProcessLine@
13864 |expandafter##1@
13865 }@
13866 }@

```

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.

```

13867 |def|markdownReadAndConvertProcessLine##1##2#1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

13868     |ifx|relax##3|relax@
13869     |markdownIfOption{frozenCache}{-}{@
13870     |immediate|write|markdownOutputStream{##1}@
13871     }@
13872     |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T<sub>E</sub>X, `\input` the result of the conversion, and expand the ending control sequence.

```

13873     |def^^M{@
13874     |markdownInfo{The ending token sequence was found}@
13875     |markdownIfOption{frozenCache}{-}{@
13876     |immediate|closeout|markdownOutputStream@
13877     }@
13878     |endgroup@
13879     |markdownInput{@
13880     |markdownOptionOutputDir@
13881     /|markdownOptionInputTempFileName@
13882     }@
13883     #2}@
13884     |fi@

```

Repeat with the next line.

```

13885     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

13886     |catcode`|^I=13@
13887     |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.

```

13888     |catcode`|^M=13@
13889     |def^^M##1^^M{@
13890     |def^^M###1^^M{@
13891     |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
13892     ^^M}@
13893     ^^M}@

```

Reset the character categories back to the former state.

```

13894 |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.

```

13895 \ExplSyntaxOn
13896 \cs_new:Npn
13897   \markdownLuaExecute
13898   #1
13899   {
13900     \int_compare:nNnT
13901       { \g_luabridge_method_int }
13902       =
13903       { \c_luabridge_method_shell_int }
13904       {
13905         \sys_if_shell_unrestricted:F
13906         {
13907           \sys_if_shell:TF
13908           {
13909             \msg_error:nn
13910               { markdown }
13911               { restricted-shell-access }
13912           }
13913           {
13914             \msg_error:nn
13915               { markdown }
13916               { disabled-shell-access }
13917           }
13918         }
13919       }
13920     \str_gset:NV
13921       \g_luabridge_output_dirname_str
13922       \markdownOptionOutputDir
13923     \luabridge_now:e
13924     { #1 }
13925   }
13926 \cs_generate_variant:Nn
13927   \msg_new:nnnn
13928   { nnnV }
13929 \tl_set:Nn
13930   \l_tmpa_tl
13931   {
13932     You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
13933     --enable-write18~flag,~or~write~shell_escape=t~in~the~
13934     texmf.cnf~file.
13935   }
13936 \msg_new:nnnV
13937   { markdown }
13938   { restricted-shell-access }
13939   { Shell~escape~is~restricted }

```

```

13940 \l_tmpa_tl
13941 \msg_new:nnnV
13942 { markdown }
13943 { disabled-shell-access }
13944 { Shell-escape-is-disabled }
13945 \l_tmpa_tl
13946 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

13947 \ExplSyntaxOn
13948 \tl_new:N
13949 \g_@@_after_markinline_tl
13950 \tl_gset:Nn
13951 \g_@@_after_markinline_tl
13952 { \unskip }
13953 \cs_new:Npn
13954 \markinline
13955 {

```

Locally change the category of the special plain  $\TeX$  characters to *other* in order to prevent unwanted interpretation of the input markdown text as  $\TeX$  code.

```

13956 \group_begin:
13957 \cctab_select:N
13958 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

13959 \@@_if_option:nF
13960 { frozenCache }
13961 {
13962 \immediate
13963 \openout
13964 \markdownOutputFileStream
13965 \markdownOptionInputTempFileName
13966 \relax
13967 \msg_info:nne
13968 { markdown }
13969 { buffering-markinline }
13970 { \markdownOptionInputTempFileName }
13971 }

```

Peek ahead and extract the inline markdown text.

```

13972 \peek_regex_replace_once:nnF
13973 { { (.*) } }
13974 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```
13975     \c { @@_if_option:nF }
13976     \cB { frozenCache \cE }
13977     \cB {
13978         \c { immediate }
13979         \c { write }
13980         \c { markdownOutputFileStream }
13981         \cB { \1 \cE }
13982     \c { immediate }
13983         \c { closeout }
13984         \c { markdownOutputFileStream }
13985     \cE }
```

Reset the category codes and `\input` the result of the conversion.

```
13986     \c { group_end: }
13987     \c { group_begin: }
13988     \c { @@_setup:n }
13989     \cB { contentLevel = inline \cE }
13990     \c { markdownInput }
13991     \cB {
13992         \c { markdownOptionOutputDir } /
13993         \c { markdownOptionInputTempFileName }
13994     \cE }
13995     \c { group_end: }
13996     \c { tl_use:N }
13997     \c { g_@@_after_markinline_tl }
13998 }
13999 {
14000     \msg_error:nn
14001     { markdown }
14002     { markinline-peek-failure }
14003     \group_end:
14004     \tl_use:N
14005     \g_@@_after_markinline_tl
14006 }
14007 }
14008 \msg_new:nnn
14009 { markdown }
14010 { buffering-markinline }
14011 { Buffering~inline~markdown~input~into~
14012   the~temporary~input~file~"#1". }
14013 \msg_new:nnnn
14014 { markdown }
14015 { markinline-peek-failure }
14016 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
14017 { The~macro~should~be~followed~by~inline~
```

```

14018     markdown~text-in~curly~braces }
14019 \ExplSyntaxOff

```

### 3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T<sub>E</sub>X.

```

14020 \ExplSyntaxOn
14021 \cs_new:Npn
14022   \markdownInput
14023   #1
14024   {
14025     \@@_if_option:nTF
14026       { frozenCache }
14027       {
14028         \markdownInputRaw
14029           { #1 }
14030       }
14031   }

```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On L<sup>A</sup>T<sub>E</sub>X, this also includes the directories specified in `\input@path`.

```

14032     \tl_set:Nx
14033       \l_tmpa_tl
14034       { #1 }
14035     \file_get_full_name:VNTF
14036       \l_tmpa_tl
14037       \l_tmpb_tl
14038     {
14039       \exp_args:NV
14040         \markdownInputRaw
14041         \l_tmpb_tl
14042     }
14043     {
14044       \msg_error:nnV
14045         { markdown }
14046         { markdown-file-does-not-exist }
14047         \l_tmpa_tl
14048     }
14049   }
14050 }
14051 \msg_new:nnn
14052   { markdown }
14053   { markdown-file-does-not-exist }
14054   {

```

```

14055     Markdown~file~#1~does~not~exist
14056   }
14057   \ExplSyntaxOff
14058   \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.

```

14059   \catcode`\|=0%
14060   \catcode`\|=12%
14061   \catcode`\&=6%
14062   \gdef|markdownInputRaw#1{%

```

Change the category code of the percent sign (%) to other, so that a user of the [hybrid](#) Lua option or a malevolent actor can't produce TeX comments in the plain TeX output of the Markdown package.

```

14063     |begingroup
14064     |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.

```

14065     |catcode`\#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

14066     |markdownIfOption{frozenCache}{%
14067       |ifnum|markdownOptionFrozenCacheCounter=0|relax
14068         |markdownInfo{Reading frozen cache from
14069           "|markdownOptionFrozenCacheFileName"}%
14070         |input|markdownOptionFrozenCacheFileName|relax
14071       |fi
14072       |markdownInfo{Including markdown document number
14073         "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
14074       |csname markdownFrozenCache%
14075         |the|markdownOptionFrozenCacheCounter|endcsname
14076       |global|advance|markdownOptionFrozenCacheCounter by 1|relax
14077     }{%
14078     |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X<sub>M</sub>k to track changes to the markdown document.

```

14079     |openin|markdownInputFileStream&1
14080     |closein|markdownInputFileStream
14081     |markdownPrepareLuaOptions
14082     |markdownPrepareInputFilename{&1}%
14083     |markdownLuaExecute{%

```

```

14084     |markdownPrepare
14085     |markdownConvert
14086     |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

14087     |markdownIfOption{finalizeCache}{%
14088         |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
14089     }%
14090     |endgroup
14091 }%
14092 |endgroup

```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of  $\TeX$  to execute a  $\TeX$  document in the middle of a markdown document fragment.

```

14093 \gdef\markdownEscape#1{%
14094     \catcode`\%=14\relax
14095     \catcode`\#=6\relax
14096     \input #1\relax
14097     \catcode`\%=12\relax
14098     \catcode`\#=12\relax
14099 }%

```

### 3.3 $\LaTeX$ Implementation

The  $\LaTeX$  implementation makes use of the fact that, apart from some subtle differences,  $\LaTeX$  implements the majority of the plain  $\TeX$  format [15, Section 9]. As a consequence, we can directly reuse the existing plain  $\TeX$  implementation.

```

14100 \def\markdownVersionSpace{ }%
14101 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
14102     \markdownVersion\markdownVersionSpace markdown renderer]%

```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.3).

```

14103 \ExplSyntaxOn
14104 \cs_gset_eq:NN
14105     \markinlinePlainTeX
14106     \markinline
14107 \cs_gset:Npn
14108     \markinline
14109     {

```



```

14110 \peek_regex_replace_once:nn
14111 { ( \[ (.*) \] ) ? }
14112 {

```

Apply the options locally.

```

14113 \c { group_begin: }
14114 \c { @@_setup:n }
14115 \cB { \2 \cE }
14116 \c { tl_put_right:Nn }
14117 \c { g_@@_after_markinline_tl }
14118 \cB { \c { group_end: } \cE }
14119 \c { markinlinePlainTeX }
14120 }
14121 }
14122 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

14123 \let\markdownInputPlainTeX\markdownInput
14124 \renewcommand\markdownInput[2] [] {%
14125 \begingroup
14126 \markdownSetup{#1}%
14127 \markdownInputPlainTeX{#2}%
14128 \endgroup}%
14129 \renewcommand\yamlInput[2] [] {%
14130 \begingroup
14131 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
14132 \markdownInputPlainTeX{#2}%
14133 \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

14134 \ExplSyntaxOn
14135 \renewenvironment
14136 { markdown }
14137 {

```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in  $\LaTeX$  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.

```
14138 \group_begin:
14139 \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).

```
14140 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
14141 \peek_regex_replace_once:nnF
14142 { \ *\[r*([~])*\[ ]^r* }
14143 {
```

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.

```
14144 \c { group_end: }
14145 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
14146 \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.

We also make provision for using the `\markdown` command as a part of a different  $\LaTeX$  environment as follows:

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

```

14147     \c { exp_args:NV }
14148     \c { markdownReadAndConvert@ }
14149     \c { @currenvir }
14150   }
14151   {
14152     \group_end:
14153     \exp_args:NV
14154     \markdownReadAndConvert@
14155     \@currenvir
14156   }
14157 }
14158 { \markdownEnd }
14159 \renewenvironment
14160 { markdown* }
14161 [ 1 ]
14162 {
14163   \@@_if_option:nTF
14164   { experimental }
14165   {
14166     \msg_error:nnn
14167     { markdown }
14168     { latex-markdown-star-deprecated }
14169     { #1 }
14170   }
14171   {
14172     \msg_warning:nnn
14173     { markdown }
14174     { latex-markdown-star-deprecated }
14175     { #1 }
14176   }
14177   \@@_setup:n
14178   { #1 }
14179   \markdownReadAndConvert@
14180   { markdown* }
14181 }
14182 { \markdownEnd }
14183 \renewenvironment
14184 { yaml }
14185 {
14186   \group_begin:
14187   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
14188   \markdown
14189 }
14190 { \yamlEnd }
14191 \msg_new:nnn
14192 { markdown }
14193 { latex-markdown-star-deprecated }

```

```

14194 {
14195   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
14196   be~removed~in~the~next~major~version~of~the~Markdown~package.
14197 }
14198 \cs_generate_variant:Nn
14199   \@@_setup:n
14200   { V }
14201 \ExplSyntaxOff
14202 \begingroup

```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (f) and right brace (r) 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*.

```

14203 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
14204 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
14205 \gdef|markdownReadAndConvert@#1<%
14206   |markdownReadAndConvert<\end{#1}>%
14207   <|end<#1>>%
14208 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\TeX$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\LaTeX$  themes provided with the Markdown package.

```

14209 \ExplSyntaxOn
14210 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
14211 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
14212 \cs_gset:Nn
14213   \@@_load_theme:nnn
14214   {

```

If the Markdown package has not yet been loaded, determine whether either this is a built-in theme according to the prop `\g_@@_latex_built_in_themes_prop` or a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

14215   \ifmarkdownLaTeXLoaded
14216     \ifx\@onlypreamble\@notprerr

```

If both conditions are true, end with an error, since we cannot load  $\LaTeX$  themes after the preamble.

```

14217     \bool_if:nTF
14218     {
14219       \bool_lazy_or_p:nn
14220       {
14221         \prop_if_in_p:Nn

```

```

14222         \g_@@_latex_built_in_themes_prop
14223         { #1 }
14224     }
14225     {
14226         \file_if_exist_p:n
14227         { markdown theme #3.sty }
14228     }
14229 }
14230 {
14231     \msg_error:nnn
14232     { markdown }
14233     { latex-theme-after-preamble }
14234     { #1 }
14235 }

```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

14236     {
14237         \@@_plain_tex_load_theme:nnn
14238         { #1 }
14239         { #2 }
14240         { #3 }
14241     }
14242 \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```

14243     \bool_if:nTF
14244     {
14245         \bool_lazy_or_p:nn
14246         {
14247             \prop_if_in_p:Nn
14248             \g_@@_latex_built_in_themes_prop
14249             { #1 }
14250         }
14251         {
14252             \file_if_exist_p:n
14253             { markdown theme #3.sty }
14254         }
14255     }
14256     {
14257         \prop_get:NnNTF
14258         \g_@@_latex_loaded_themes_linenos_prop
14259         { #1 }
14260         \l_tmpa_tl
14261         {
14262             \prop_get:NnN
14263             \g_@@_latex_loaded_themes_versions_prop
14264             { #1 }

```

```

14265         \l_tmpb_tl
14266     \str_if_eq:nVTF
14267     { #2 }
14268     \l_tmpb_tl
14269     {
14270         \msg_warning:nnnVn
14271         { markdown }
14272         { repeatedly-loaded-latex-theme }
14273         { #1 }
14274         \l_tmpa_tl
14275         { #2 }
14276     }
14277     {
14278         \msg_error:nnnnVV
14279         { markdown }
14280         { different-versions-of-latex-theme }
14281         { #1 }
14282         { #2 }
14283         \l_tmpb_tl
14284         \l_tmpa_tl
14285     }
14286 }
14287 {
14288     \prop_gput:Nnx
14289     \g_@@_latex_loaded_themes_linenos_prop
14290     { #1 }
14291     { \tex_the:D \tex_inputlineno:D }
14292     \prop_gput:Nnn
14293     \g_@@_latex_loaded_themes_versions_prop
14294     { #1 }
14295     { #2 }

```

Load built-in plain T<sub>E</sub>X themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

14296     \prop_if_in:NnTF
14297     \g_@@_latex_built_in_themes_prop
14298     { #1 }
14299     {
14300         \msg_info:nnnn
14301         { markdown }
14302         { loading-built-in-latex-theme }
14303         { #1 }
14304         { #2 }
14305     }
14306     \prop_item:Nn
14307     \g_@@_latex_built_in_themes_prop
14308     { #1 }

```

```

14309         {
14310             \msg_info:nnnn
14311             { markdown }
14312             { loading-latex-theme }
14313             { #1 }
14314             { #2 }
14315             \RequirePackage
14316             { markdown theme #3 }
14317         }
14318     }
14319 }
14320 {
14321     \@@_plain_tex_load_theme:nnn
14322     { #1 }
14323     { #2 }
14324     { #3 }
14325 }
14326 \fi
14327 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

14328     \msg_info:nnnn
14329     { markdown }
14330     { theme-loading-postponed }
14331     { #1 }
14332     { #2 }
14333     \AtEndOfPackage
14334     {
14335         \@@_set_theme:n
14336         { #1 @ #2 }
14337     }
14338 \fi
14339 }
14340 \msg_new:nnn
14341 { markdown }
14342 { theme-loading-postponed }
14343 {
14344     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
14345     Markdown~package~has~finished~loading
14346 }
14347 \msg_new:nnn
14348 { markdown }
14349 { loading-built-in-latex-theme }
14350 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
14351 \msg_new:nnn
14352 { markdown }

```

```

14353 { loading-latex-theme }
14354 { Loading-version-#2-of-LaTeX-Markdown-theme-#1 }
14355 \msg_new:nnn
14356 { markdown }
14357 { repeatedly-loaded-latex-theme }
14358 {
14359     Version-#3-of-LaTeX-Markdown-theme-#1-was-previously-
14360     loaded-on-line-#2,~not~loading-it~again
14361 }
14362 \msg_new:nnn
14363 { markdown }
14364 { different-versions-of-latex-theme }
14365 {
14366     Tried-to-load-version-#2-of-LaTeX-Markdown-theme-#1~
14367     but-version-#3-has-already-been-loaded-on-line-#4
14368 }
14369 \cs_generate_variant:Nn
14370 \msg_new:nnnn
14371 { nnVV }
14372 \tl_set:Nn
14373 \l_tmpa_tl
14374 { Cannot-load-LaTeX-Markdown-theme-#1~after~ }
14375 \tl_put_right:NV
14376 \l_tmpa_tl
14377 \c_backslash_str
14378 \tl_put_right:Nn
14379 \l_tmpa_tl
14380 { begin{document} }
14381 \tl_set:Nn
14382 \l_tmpb_tl
14383 { Load-Markdown-theme-#1~before~ }
14384 \tl_put_right:NV
14385 \l_tmpb_tl
14386 \c_backslash_str
14387 \tl_put_right:Nn
14388 \l_tmpb_tl
14389 { begin{document} }
14390 \msg_new:nnVV
14391 { markdown }
14392 { latex-theme-after-preamble }
14393 \l_tmpa_tl
14394 \l_tmpb_tl

```

The [witiko/dot](#) and [witiko/graphicx/http](#) L<sup>A</sup>T<sub>E</sub>X themes load the package `graphicx`, see also Section 1.1.3. Then, they load the corresponding plain T<sub>E</sub>X themes.

```

14395 \tl_set:Nn
14396 \l_tmpa_tl

```



```

14397 {
14398   \RequirePackage
14399     { graphicx }
14400   \markdownLoadPlainTeXTheme
14401 }
14402 \prop_gput:NnV
14403   \g_@@_latex_built_in_themes_prop
14404   { witiko / dot }
14405   \l_tmpa_tl
14406 \prop_gput:NnV
14407   \g_@@_latex_built_in_themes_prop
14408   { witiko / graphicx / http }
14409   \l_tmpa_tl
14410 \ExplSyntaxOff

```

The `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>X theme also loads the corresponding plain T<sub>E</sub>X theme.

```
14411 \markdownLoadPlainTeXTheme
```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

14412 \DeclareOption*{%
14413   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
14414 \ProcessOptions\relax

```

### 3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```
14415 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

#### 3.3.4.1 Lists

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, use a package that provides support for tight and fancy lists.

If either the package `paralist` or the package `enumitem` have already been loaded, use them. Otherwise, if the option `experimental` or any test phase has been enabled, use the package `enumitem`. Otherwise, use the package `paralist`.

```

14416 \ExplSyntaxOn
14417 \bool_new:N
14418   \g_@@_tight_or_fancy_lists_bool
14419 \bool_gset_false:N

```

```

14420 \g_@@_tight_or_fancy_lists_bool
14421 \@@_if_option:nTF
14422 { tightLists }
14423 {
14424   \bool_gset_true:N
14425   \g_@@_tight_or_fancy_lists_bool
14426 }
14427 {
14428   \@@_if_option:nT
14429   { fancyLists }
14430   {
14431     \bool_gset_true:N
14432     \g_@@_tight_or_fancy_lists_bool
14433   }
14434 }
14435 \bool_new:N
14436 \g_@@_beamer_paralist_or_enumitem_bool
14437 \bool_gset_true:N
14438 \g_@@_beamer_paralist_or_enumitem_bool
14439 \@ifclassloaded
14440 { beamer }
14441 { }
14442 {
14443   \@ifpackageloaded
14444   { paralist }
14445   { }
14446   {
14447     \@ifpackageloaded
14448     { enumitem }
14449     { }
14450     {
14451       \bool_gset_false:N
14452       \g_@@_beamer_paralist_or_enumitem_bool
14453     }
14454   }
14455 }
14456 \bool_if:nT
14457 {
14458   \g_@@_tight_or_fancy_lists_bool &&
14459   ! \g_@@_beamer_paralist_or_enumitem_bool
14460 }
14461 {
14462   \bool_if:nTF
14463   {
14464     \bool_lazy_or_p:nn
14465     {
14466       \str_if_eq_p:en

```

```

14467         { \markdownThemeVersion }
14468         { experimental }
14469     }
14470     {
14471     \bool_lazy_and_p:nn
14472     {
14473     \prop_if_exist_p:N
14474     \g__pdfmanagement_documentproperties_prop
14475     }
14476     {
14477     \bool_lazy_any_p:n
14478     {
14479     {
14480     \prop_if_in_p:Nn
14481     \g__pdfmanagement_documentproperties_prop
14482     { document / testphase / phase-I }
14483     }
14484     {
14485     \prop_if_in_p:Nn
14486     \g__pdfmanagement_documentproperties_prop
14487     { document / testphase / phase-II }
14488     }
14489     {
14490     \prop_if_in_p:Nn
14491     \g__pdfmanagement_documentproperties_prop
14492     { document / testphase / phase-III }
14493     }
14494     {
14495     \prop_if_in_p:Nn
14496     \g__pdfmanagement_documentproperties_prop
14497     { document / testphase / phase-IV }
14498     }
14499     {
14500     \prop_if_in_p:Nn
14501     \g__pdfmanagement_documentproperties_prop
14502     { document / testphase / phase-V }
14503     }
14504     {
14505     \prop_if_in_p:Nn
14506     \g__pdfmanagement_documentproperties_prop
14507     { document / testphase / phase-VI }
14508     }
14509     }
14510     }
14511     }
14512     }
14513     {

```

```

14514     \RequirePackage
14515         { enumitem }
14516     }
14517     {
14518         \RequirePackage
14519             { paralist }
14520     }
14521 }
14522 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

14523 \ExplSyntaxOn
14524 \cs_new:Nn
14525   \@@_latex_fancy_list_item_label_number:nn
14526   {
14527     \str_case:nn
14528       { #1 }
14529       {
14530         { Decimal } { #2 }
14531         { LowerRoman } { \int_to_roman:n { #2 } }
14532         { UpperRoman } { \int_to_Roman:n { #2 } }
14533         { LowerAlpha } { \int_to_alph:n { #2 } }
14534         { UpperAlpha } { \int_to_Alph:n { #2 } }
14535       }
14536   }
14537 \cs_new:Nn
14538   \@@_latex_fancy_list_item_label_delimiter:n
14539   {
14540     \str_case:nn
14541       { #1 }
14542       {
14543         { Default } { . }
14544         { OneParen } { ) }
14545         { Period } { . }
14546       }
14547   }
14548 \cs_new:Nn
14549   \@@_latex_fancy_list_item_label:nnn
14550   {
14551     \@@_latex_fancy_list_item_label_number:nn
14552       { #1 }
14553       { #3 }
14554     \@@_latex_fancy_list_item_label_delimiter:n
14555       { #2 }
14556   }
14557 \cs_generate_variant:Nn

```

```

14558 \l@@_latex_fancy_list_item_label:nnn
14559 { VVn }
14560 \tl_new:N
14561 \l_@@_latex_fancy_list_item_label_number_style_tl
14562 \tl_new:N
14563 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14564 \@ifpackageloaded{enumitem}{
14565 \markdownSetup{rendererPrototypes={

```

First, let's define the tight list item renderer prototypes.

```

14566     ulBeginTight = {
14567         \begin
14568             { itemize }
14569             [ noitemsep ]
14570     },
14571     ulEndTight = {
14572         \end
14573             { itemize }
14574     },
14575     olBeginTight = {
14576         \begin
14577             { enumerate }
14578             [ noitemsep ]
14579     },
14580     olEndTight = {
14581         \end
14582             { enumerate }
14583     },
14584     dlBeginTight = {
14585         \begin
14586             { description }
14587             [ noitemsep ]
14588     },
14589     dlEndTight = {
14590         \end
14591             { description }
14592     },

```

Second, let's define the fancy list item renderer prototypes.

```

14593     fancyOlBegin = {
14594         \group_begin:
14595         \tl_set:Nn
14596             \l_@@_latex_fancy_list_item_label_number_style_tl
14597             { #1 }
14598         \tl_set:Nn
14599             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14600             { #2 }
14601         \begin

```

```

14602     { enumerate }
14603 },
14604 fancyOlBeginTight = {
14605   \group_begin:
14606   \tl_set:Nn
14607     \l_@@_latex_fancy_list_item_label_number_style_tl
14608     { #1 }
14609   \tl_set:Nn
14610     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14611     { #2 }
14612   \begin
14613     { enumerate }
14614     [ noitemsep ]
14615 },
14616 fancyOlEnd(|Tight) = {
14617   \end { enumerate }
14618   \group_end:
14619 },
14620 fancyOlItemWithNumber = {
14621   \item
14622   [
14623     \@@_latex_fancy_list_item_label:VVn
14624     \l_@@_latex_fancy_list_item_label_number_style_tl
14625     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14626     { #1 }
14627   ]
14628 },
14629 }}

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

14630 }{\@ifpackageloaded{paralist}{
14631   \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

14632   ulBeginTight = {%
14633     \group_begin:
14634     \pltopsep=\topsep
14635     \plpartopsep=\partopsep
14636     \begin{compactitem}
14637   },
14638   ulEndTight = {
14639     \end{compactitem}
14640     \group_end:
14641   },
14642   fancyOlBegin = {
14643     \group_begin:

```

```

14644     \tl_set:Nn
14645         \l_@@_latex_fancy_list_item_label_number_style_tl
14646         { #1 }
14647     \tl_set:Nn
14648         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14649         { #2 }
14650     \begin{enumerate}
14651 },
14652 fancyOlEnd = {
14653     \end{enumerate}
14654     \group_end:
14655 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

14656     olBeginTight = {%
14657         \group_begin:
14658         \plpartopsep=\partopsep
14659         \pltopsep=\topsep
14660         \begin{compactenum}
14661     },
14662     olEndTight = {
14663         \end{compactenum}
14664         \group_end:
14665     },
14666     fancyOlBeginTight = {
14667         \group_begin:
14668         \tl_set:Nn
14669             \l_@@_latex_fancy_list_item_label_number_style_tl
14670             { #1 }
14671         \tl_set:Nn
14672             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14673             { #2 }
14674         \plpartopsep=\partopsep
14675         \pltopsep=\topsep
14676         \begin{compactenum}
14677     },
14678     fancyOlEndTight = {
14679         \end{compactenum}
14680         \group_end:
14681     },
14682     fancyOlItemWithNumber = {
14683         \item
14684         [
14685             \@@_latex_fancy_list_item_label:VVn
14686             \l_@@_latex_fancy_list_item_label_number_style_tl
14687             \l_@@_latex_fancy_list_item_label_delimiter_style_tl

```

```

14688         { #1 }
14689     ]
14690 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

14691     dlBeginTight = {
14692         \group_begin:
14693         \plpartopsep=\partopsep
14694         \pltopsep=\topsep
14695         \begin{compactdesc}
14696     },
14697     dlEndTight = {
14698         \end{compactdesc}
14699         \group_end:
14700     }
14701 }}
14702 }{

```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

14703 \markdownSetup
14704 {
14705     rendererPrototypes = {
14706         ulBeginTight = \markdownRendererUlBegin,
14707         ulEndTight = \markdownRendererUlEnd,
14708         fancyOlBegin = \markdownRendererOlBegin,
14709         fancyOlEnd = \markdownRendererOlEnd,
14710         olBeginTight = \markdownRendererOlBegin,
14711         olEndTight = \markdownRendererOlEnd,
14712         fancyOlBeginTight = \markdownRendererOlBegin,
14713         fancyOlEndTight = \markdownRendererOlEnd,
14714         dlBeginTight = \markdownRendererDlBegin,
14715         dlEndTight = \markdownRendererDlEnd,
14716     },
14717 }
14718 }}
14719 \ExplSyntaxOff
14720 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

14721 \@ifpackageloaded{unicode-math}{
14722     \markdownSetup{rendererPrototypes={
14723         untickedBox = {\$ \mdlgwhtsquare$},
14724     }}
14725 }{

```



```

14726 \RequirePackage{amssymb}
14727 \markdownSetup{rendererPrototypes={
14728     untickedBox = {\square$},
14729 }}
14730 }
14731 \RequirePackage{csvsimple}
14732 \RequirePackage{fancyvrb}
14733 \RequirePackage{graphicx}
14734 \markdownSetup{rendererPrototypes={
14735     hardLineBreak = {\},
14736     leftBrace = {\textbraceleft},
14737     rightBrace = {\textbraceright},
14738     dollarSign = {\textdollar},
14739     underscore = {\textunderscore},
14740     circumflex = {\textasciicircum},
14741     backslash = {\textbackslash},
14742     tilde = {\textasciitilde},
14743     pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

14744 codeSpan = {%
14745     \ifmmode
14746         \text{#1}%
14747     \else
14748         \texttt{#1}%
14749     \fi
14750 }}}

```

### 3.3.4.2 Content Blocks

In content block renderer prototypes, display the content as a table using the package `csvsimple` when the raw attribute is `csv`, display the content using the default templates of the package `luaxml` when the raw attribute is `html`, execute the content with TeX when the raw attribute is `tex`, and display the content as markdown otherwise.

```

14751 \ExplSyntaxOn
14752 \markdownSetup{
14753     rendererPrototypes = {

```

---

<sup>34</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

14754     contentBlock = {
14755         \str_case:nnF
14756         { #1 }
14757         {
14758             { csv }
14759             {
14760                 \begin{table}
14761                 \begin{center}
14762                     \csvautotabular{#3}
14763                 \end{center}
14764                 \tl_if_empty:nF
14765                 { #4 }
14766                 { \caption{#4} }
14767                 \end{table}
14768             }
14769         { html }
14770         {

```

If we are using  $\text{T}_{\text{E}}\text{X}4\text{ht}$ <sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

14771         \cs_if_exist:NTF
14772         \HCode
14773         {
14774             \if_mode_vertical:
14775             \IgnorePar
14776             \fi:
14777             \EndP
14778             \special
14779             { t4ht* < #3 }
14780             \par
14781             \ShowPar
14782         }
14783         {
14784             \@@_luaxml_print_html:n
14785             { #3 }
14786         }
14787     }
14788 { tex }
14789 {
14790     \markdownEscape
14791     { #3 }
14792 }
14793 }
14794 {
14795     \markdownInput
14796     { #3 }

```

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

14797     }
14798   },
14799 },
14800 }
14801 \ExplSyntaxOff
14802 \markdownSetup{rendererPrototypes={
14803   ulBegin = {\begin{itemize}},
14804   ulEnd = {\end{itemize}},
14805   olBegin = {\begin{enumerate}},
14806   olItem = {\item{}},
14807   olItemWithNumber = {\item[#1.]},
14808   olEnd = {\end{enumerate}},
14809   dlBegin = {\begin{description}},
14810   dlItem = {\item[#1]},
14811   dlEnd = {\end{description}},
14812   emphasis = {\emph{#1}},
14813   tickedBox = {\$\boxtimes$},
14814   halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

14815 \ExplSyntaxOn
14816 \seq_new:N
14817   \g_@@_header_identifiers_seq
14818 \markdownSetup
14819   {
14820     rendererPrototypes = {
14821       headerAttributeContextBegin = {
14822         \markdownSetup
14823           {
14824             rendererPrototypes = {
14825               attributeIdentifier = {
14826                 \seq_gput_right:Nn
14827                   \g_@@_header_identifiers_seq
14828                   { ##1 }
14829               },
14830             },
14831           }
14832       },
14833       headerAttributeContextEnd = {
14834         \seq_map_inline:Nn
14835           \g_@@_header_identifiers_seq
14836           { \label { ##1 } }
14837         \seq_gclear:N
14838           \g_@@_header_identifiers_seq
14839       },
14840     },
14841   }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

14842 \bool_new:N
14843   \l_@@_header_unnumbered_bool
14844 \markdownSetup
14845   {
14846     rendererPrototypes = {
14847       headerAttributeContextBegin += {
14848         \markdownSetup
14849         {
14850           rendererPrototypes = {
14851             attributeClassName = {
14852               \bool_if:nT
14853               {
14854                 \str_if_eq_p:nn
14855                 { ##1 }
14856                 { unnumbered } &&
14857                 ! \l_@@_header_unnumbered_bool
14858               }
14859               {
14860                 \group_begin:
14861                 \bool_set_true:N
14862                 \l_@@_header_unnumbered_bool
14863                 \c@secnumdepth = 0
14864                 \markdownSetup
14865                 {
14866                   rendererPrototypes = {
14867                     sectionBegin = {
14868                       \group_begin:
14869                     },
14870                     sectionEnd = {
14871                       \group_end:
14872                     },
14873                   },
14874                 }
14875               }
14876             },
14877           },
14878         }
14879       },
14880     },
14881   }
14882 \ExplSyntaxOff
14883 \markdownSetup{rendererPrototypes={
14884   superscript = {\textsuperscript{#1}},
14885   subscript = {\textsubscript{#1}},
14886   blockQuoteBegin = {\begin{quotation}},

```

```

14887   blockQuoteEnd = {\end{quotation}},
14888   inputVerbatim = {\VerbatimInput{#1}},
14889   thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
14890   note = {\footnote{#1}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

14891 \RequirePackage{ltxcmds}
14892 \ExplSyntaxOn
14893 \cs_gset_protected:Npn
14894   \markdownRendererInputFencedCodePrototype#1#2#3
14895   {
14896     \tl_if_empty:nTF
14897       { #2 }
14898     { \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.

```

14899     {
14900       \regex_extract_once:nnN
14901         { \w* }
14902         { #2 }
14903         \l_tmpa_seq
14904       \seq_pop_left:NN
14905         \l_tmpa_seq
14906         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

14907     \ltx@ifpackageloaded
14908       { minted }
14909     {
14910       \catcode`\%=14\relax
14911       \catcode`\#=6\relax
14912       \exp_args:NV
14913         \inputminted
14914         \l_tmpa_tl
14915         { #1 }
14916       \catcode`\%=12\relax
14917       \catcode`\#=12\relax
14918     }
14919     {

```

When the listings package is loaded, use it for syntax highlighting.

```

14920     \ltx@ifpackageloaded
14921       { listings }
14922     { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

14923             { \markdownRendererInputFencedCode{#1}{}{} }
14924         }
14925     }
14926 }
14927 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

14928 \ExplSyntaxOn
14929 \def\markdownLATEXStrongEmphasis#1{%
14930     \str_if_in:NnTF
14931         \f@series
14932         { b }
14933         { \textnormal{#1} }
14934         { \textbf{#1} }
14935 }
14936 \ExplSyntaxOff
14937 \markdownSetup{rendererPrototypes={strongEmphasis={%
14938     \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```

14939 \@ifundefined{chapter}{%
14940     \markdownSetup{rendererPrototypes = {
14941         headingOne = {\section{#1}},
14942         headingTwo = {\subsection{#1}},
14943         headingThree = {\subsubsection{#1}},
14944         headingFour = {\paragraph{#1}},
14945         headingFive = {\subparagraph{#1}}}}
14946 }{%
14947     \markdownSetup{rendererPrototypes = {
14948         headingOne = {\chapter{#1}},
14949         headingTwo = {\section{#1}},
14950         headingThree = {\subsection{#1}},
14951         headingFour = {\subsubsection{#1}},
14952         headingFive = {\paragraph{#1}},
14953         headingSix = {\subparagraph{#1}}}}
14954 }%

```

#### 3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

14955 \markdownSetup{
14956     rendererPrototypes = {
14957         ulItem = {%
14958             \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
14959         },

```

```

14960   },
14961   }
14962   \def\markdownLaTeXUListItem{%
14963     \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
14964       \item[\markdownLaTeXCheckbox]%
14965       \expandafter\@gobble
14966     \else
14967       \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
14968         \item[\markdownLaTeXCheckbox]%
14969         \expandafter\expandafter\expandafter\@gobble
14970       \else
14971         \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
14972           \item[\markdownLaTeXCheckbox]%
14973           \expandafter\expandafter\expandafter\expandafter
14974             \expandafter\expandafter\expandafter\@gobble
14975         \else
14976           \item{}%
14977         \fi
14978       \fi
14979     \fi
14980   }

```

### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using `TeX4ht`<sup>36</sup>, we will pass HTML elements to the output HTML document unchanged.

```

14981 \@ifundefined{HCode}{}{
14982   \markdownSetup{
14983     rendererPrototypes = {
14984       inlineHtmlTag = {%
14985         \ifvmode
14986           \IgnorePar
14987           \EndP
14988         \fi
14989         \HCode{#1}%
14990       },
14991       inputBlockHtmlElement = {%
14992         \ifvmode
14993           \IgnorePar
14994           \fi
14995           \EndP
14996           \special{t4ht*#1}%
14997           \par
14998           \ShowPar
14999       },
15000     },

```

---

<sup>36</sup>See <https://tug.org/tex4ht/>.

```

15001 }
15002 }

```

### 3.3.4.6 Citations

Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

15003 \newcount\markdownLaTeXCitationsCounter
15004
15005 % Basic implementation
15006 \long\def@gobblethree#1#2#3{%
15007 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
15008 \advance\markdownLaTeXCitationsCounter by 1\relax
15009 \ifx\relax#4\relax
15010 \ifx\relax#5\relax
15011 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15012 \relax
15013 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
15014 \expandafter\expandafter\expandafter
15015 \expandafter\expandafter\expandafter\expandafter
15016 \@gobblethree
15017 \fi
15018 \else% Before a postnote (#5), dump the accumulator
15019 \ifx\relax#1\relax\else
15020 \cite{#1}%
15021 \fi
15022 \cite[#5]{#6}%
15023 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15024 \relax
15025 \else
15026 \expandafter\expandafter\expandafter
15027 \expandafter\expandafter\expandafter\expandafter
15028 \expandafter\expandafter\expandafter
15029 \expandafter\expandafter\expandafter\expandafter
15030 \markdownLaTeXBasicCitations
15031 \fi
15032 \expandafter\expandafter\expandafter
15033 \expandafter\expandafter\expandafter\expandafter{%
15034 \expandafter\expandafter\expandafter
15035 \expandafter\expandafter\expandafter\expandafter}%
15036 \expandafter\expandafter\expandafter
15037 \expandafter\expandafter\expandafter\expandafter{%
15038 \expandafter\expandafter\expandafter
15039 \expandafter\expandafter\expandafter\expandafter}%
15040 \expandafter\expandafter\expandafter

```



```

15041     \@gobblethree
15042     \fi
15043 \else% Before a prenote (#4), dump the accumulator
15044     \ifx\relax#1\relax\else
15045         \cite{#1}%
15046     \fi
15047     \ifnum\markdownLaTeXCitationsCounter>1\relax
15048         \space % Insert a space before the prenote in later citations
15049     \fi
15050     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
15051     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15052     \relax
15053 \else
15054     \expandafter\expandafter\expandafter
15055     \expandafter\expandafter\expandafter\expandafter
15056     \markdownLaTeXBasicCitations
15057 \fi
15058 \expandafter\expandafter\expandafter{%
15059 \expandafter\expandafter\expandafter}%
15060 \expandafter\expandafter\expandafter{%
15061 \expandafter\expandafter\expandafter}%
15062 \expandafter
15063 \@gobblethree
15064 \fi\markdownLaTeXBasicCitations{#1#2#6},}
15065 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
15066
15067 % Natbib implementation
15068 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
15069     \advance\markdownLaTeXCitationsCounter by 1\relax
15070     \ifx\relax#3\relax
15071         \ifx\relax#4\relax
15072             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15073             \relax
15074             \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
15075             \expandafter\expandafter\expandafter
15076             \expandafter\expandafter\expandafter\expandafter
15077             \@gobbletwo
15078         \fi
15079     \else% Before a postnote (#4), dump the accumulator
15080         \ifx\relax#1\relax\else
15081             \citep{#1}%
15082         \fi
15083         \citep[] [#4]{#5}%
15084         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15085         \relax
15086     \else
15087         \expandafter\expandafter\expandafter

```

```

15088     \expandafter\expandafter\expandafter\expandafter
15089     \expandafter\expandafter\expandafter
15090     \expandafter\expandafter\expandafter\expandafter
15091     \markdownLaTeXNatbibCitations
15092     \fi
15093     \expandafter\expandafter\expandafter
15094     \expandafter\expandafter\expandafter\expandafter{%
15095     \expandafter\expandafter\expandafter
15096     \expandafter\expandafter\expandafter\expandafter}%
15097     \expandafter\expandafter\expandafter
15098     \@gobbletwo
15099     \fi
15100 \else% Before a prenote (#3), dump the accumulator
15101     \ifx\relax#1\relax\relax\else
15102         \citep{#1}%
15103     \fi
15104     \citep[#3][#4]{#5}%
15105     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15106     \relax
15107     \else
15108         \expandafter\expandafter\expandafter
15109         \expandafter\expandafter\expandafter\expandafter
15110         \markdownLaTeXNatbibCitations
15111     \fi
15112     \expandafter\expandafter\expandafter{%
15113     \expandafter\expandafter\expandafter}%
15114     \expandafter
15115     \@gobbletwo
15116     \fi\markdownLaTeXNatbibCitations{#1,#5}}
15117 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
15118     \advance\markdownLaTeXCitationsCounter by 1\relax
15119     \ifx\relax#3\relax
15120         \ifx\relax#4\relax
15121             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15122             \relax
15123             \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
15124             \expandafter\expandafter\expandafter
15125             \expandafter\expandafter\expandafter\expandafter
15126             \@gobbletwo
15127         \fi
15128     \else% After a prenote or a postnote, dump the accumulator
15129         \ifx\relax#1\relax\else
15130             \citet{#1}%
15131         \fi
15132         , \citet[#3][#4]{#5}%
15133         \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15134         \relax

```

```

15135     ,
15136     \else
15137         \ifnum
15138             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15139         \relax
15140     ,
15141     \fi
15142 \fi
15143 \expandafter\expandafter\expandafter
15144 \expandafter\expandafter\expandafter\expandafter
15145 \markdownLaTeXNatbibTextCitations
15146 \expandafter\expandafter\expandafter
15147 \expandafter\expandafter\expandafter\expandafter{%
15148 \expandafter\expandafter\expandafter
15149 \expandafter\expandafter\expandafter\expandafter}%
15150 \expandafter\expandafter\expandafter
15151 \@gobbletwo
15152 \fi
15153 \else% After a prenote or a postnote, dump the accumulator
15154 \ifx\relax#1\relax\relax\else
15155     \citet{#1}%
15156 \fi
15157 , \citet[#3][#4]{#5}%
15158 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15159 \relax
15160 ,
15161 \else
15162     \ifnum
15163         \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15164     \relax
15165     ,
15166     \fi
15167 \fi
15168 \expandafter\expandafter\expandafter
15169 \markdownLaTeXNatbibTextCitations
15170 \expandafter\expandafter\expandafter{%
15171 \expandafter\expandafter\expandafter}%
15172 \expandafter
15173 \@gobbletwo
15174 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
15175
15176 % BibLaTeX implementation
15177 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
15178 \advance\markdownLaTeXCitationsCounter by 1\relax
15179 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15180 \relax
15181 \autocites#1[#3][#4]{#5}%

```

```

15182     \expandafter\@gobbletwo
15183     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
15184 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
15185     \advance\markdownLaTeXCitationsCounter by 1\relax
15186     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15187     \relax
15188     \textcites#1[#3][#4]{#5}%
15189     \expandafter\@gobbletwo
15190     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
15191
15192 \markdownSetup{rendererPrototypes = {
15193     cite = {%
15194         \markdownLaTeXCitationsCounter=1%
15195         \def\markdownLaTeXCitationsTotal{#1}%
15196         \@ifundefined{autocites}{%
15197             \@ifundefined{citep}{%
15198                 \expandafter\expandafter\expandafter
15199                 \markdownLaTeXBasicCitations
15200                 \expandafter\expandafter\expandafter{%
15201                 \expandafter\expandafter\expandafter}%
15202                 \expandafter\expandafter\expandafter{%
15203                 \expandafter\expandafter\expandafter}%
15204             }{%
15205                 \expandafter\expandafter\expandafter
15206                 \markdownLaTeXNatbibCitations
15207                 \expandafter\expandafter\expandafter{%
15208                 \expandafter\expandafter\expandafter}%
15209             }%
15210         }{%
15211             \expandafter\expandafter\expandafter
15212             \markdownLaTeXBibLaTeXCitations
15213             \expandafter{\expandafter}%
15214         }},
15215     textCite = {%
15216         \markdownLaTeXCitationsCounter=1%
15217         \def\markdownLaTeXCitationsTotal{#1}%
15218         \@ifundefined{autocites}{%
15219             \@ifundefined{citep}{%
15220                 \expandafter\expandafter\expandafter
15221                 \markdownLaTeXBasicTextCitations
15222                 \expandafter\expandafter\expandafter{%
15223                 \expandafter\expandafter\expandafter}%
15224                 \expandafter\expandafter\expandafter{%
15225                 \expandafter\expandafter\expandafter}%
15226             }{%
15227                 \expandafter\expandafter\expandafter
15228                 \markdownLaTeXNatbibTextCitations

```

```

15229     \expandafter\expandafter\expandafter{%
15230     \expandafter\expandafter\expandafter}%
15231     }%
15232   }{%
15233     \expandafter\expandafter\expandafter
15234     \markdownLaTeXBibLaTeXTextCitations
15235     \expandafter{\expandafter}%
15236   }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

15237 \RequirePackage{url}
15238 \RequirePackage{expl3}
15239 \ExplSyntaxOn
15240 \cs_gset_protected:Npn
15241   \markdownRendererLinkPrototype
15242   #1#2#3#4
15243   {
15244     \tl_set:Nn \l_tmpa_tl { #1 }
15245     \tl_set:Nn \l_tmpb_tl { #2 }
15246     \bool_set:Nn
15247       \l_tmpa_bool
15248       {
15249         \tl_if_eq_p:NN
15250           \l_tmpa_tl
15251           \l_tmpb_tl
15252       }
15253     \tl_set:Nn \l_tmpa_tl { #4 }
15254     \bool_set:Nn
15255       \l_tmpb_bool
15256       {
15257         \tl_if_empty_p:N
15258           \l_tmpa_tl
15259       }

```

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.

```

15260   \bool_if:nTF
15261     {
15262       \l_tmpa_bool && \l_tmpb_bool
15263     }
15264     {
15265       \markdownLaTeXRendererAutolink { #2 } { #3 }
15266     }{
15267       \markdownLaTeXRendererDirectOrIndirectLink
15268       { #1 } { #2 } { #3 } { #4 }

```

```

15269     }
15270   }
15271   \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.
15272   \tl_set:Nn
15273     \l_tmpa_tl
15274     { #2 }
15275   \tl_trim_spaces:N
15276     \l_tmpa_tl
15277   \tl_set:Nx
15278     \l_tmpb_tl
15279     {
15280       \tl_range:Nnn
15281         \l_tmpa_tl
15282         { 1 }
15283         { 1 }
15284     }
15285   \str_if_eq:NNTF
15286     \l_tmpb_tl
15287     \c_hash_str
15288     {
15289       \tl_set:Nx
15290         \l_tmpb_tl
15291         {
15292           \tl_range:Nnn
15293             \l_tmpa_tl
15294             { 2 }
15295             { -1 }
15296         }
15297       \exp_args:NV
15298         \ref
15299         \l_tmpb_tl
15300     }{
15301       \url { #2 }
15302     }
15303 }
15304 \ExplSyntaxOff
15305 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
15306   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

### 3.3.4.8 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

15307 \newcount\markdownLaTeXRowCount
15308 \newcount\markdownLaTeXRowTotal
```

```

15309 \newcount\markdownLaTeXColumnCounter
15310 \newcount\markdownLaTeXColumnTotal
15311 \newtoks\markdownLaTeXTable
15312 \newtoks\markdownLaTeXTableAlignment
15313 \newtoks\markdownLaTeXTableEnd
15314 \AtBeginDocument{%
15315   \@ifpackageloaded{booktabs}{%
15316     \def\markdownLaTeXTopRule{\toprule}%
15317     \def\markdownLaTeXMidRule{\midrule}%
15318     \def\markdownLaTeXBottomRule{\bottomrule}%
15319   }{%
15320     \def\markdownLaTeXTopRule{\hline}%
15321     \def\markdownLaTeXMidRule{\hline}%
15322     \def\markdownLaTeXBottomRule{\hline}%
15323   }%
15324 }
15325 \markdownSetup{rendererPrototypes={
15326   table = {%
15327     \markdownLaTeXTable={}%
15328     \markdownLaTeXTableAlignment={}%
15329     \markdownLaTeXTableEnd={%
15330       \markdownLaTeXBottomRule
15331       \end{tabular}}%
15332     \ifx\empty#1\empty\else
15333       \addto@hook\markdownLaTeXTable{%
15334         \begin{table}
15335         \centering}%
15336       \addto@hook\markdownLaTeXTableEnd{%
15337         \caption{#1}}%
15338     \fi
15339   }
15340 }}

```

If the `tableAttributes` option is enabled, we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing tables.

```

15341 \ExplSyntaxOn
15342 \seq_new:N
15343   \l_@@_table_identifiers_seq
15344 \markdownSetup {
15345   rendererPrototypes = {
15346     table += {
15347       \seq_map_inline:Nn
15348         \l_@@_table_identifiers_seq
15349         {
15350           \addto@hook
15351             \markdownLaTeXTableEnd
15352             { \label { ##1 } }

```

```

15353     }
15354   },
15355 }
15356 }
15357 \markdownSetup {
15358   rendererPrototypes = {
15359     tableAttributeContextBegin = {
15360       \group_begin:
15361       \markdownSetup {
15362         rendererPrototypes = {
15363           attributeIdentifier = {
15364             \seq_put_right:Nn
15365             \l_@@_table_identifiers_seq
15366             { ##1 }
15367           },
15368         },
15369       }
15370     },
15371     tableAttributeContextEnd = {
15372       \group_end:
15373     },
15374   },
15375 }
15376 \ExplSyntaxOff
15377 \markdownSetup{rendererPrototypes={
15378   table += {%
15379     \ifx\empty#1\empty\else
15380       \addto@hook\markdownLaTeXTableEnd{%
15381         \end{table}}%
15382     \fi
15383     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
15384     \markdownLaTeXRowCount=0%
15385     \markdownLaTeXRowTotal=#2%
15386     \markdownLaTeXColumnTotal=#3%
15387     \markdownLaTeXRenderTableRow
15388   }
15389 }}
15390 \def\markdownLaTeXRenderTableRow#1{%
15391   \markdownLaTeXColumnCounter=0%
15392   \ifnum\markdownLaTeXRowCount=0\relax
15393     \markdownLaTeXReadAlignments#1%
15394     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
15395       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
15396         \the\markdownLaTeXTableAlignment}}%
15397     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
15398   \else
15399     \markdownLaTeXRenderTableCell#1%

```



```

15400 \fi
15401 \ifnum\markdownLaTeXRowCount=1\relax
15402   \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
15403 \fi
15404 \advance\markdownLaTeXRowCount by 1\relax
15405 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
15406   \the\markdownLaTeXTable
15407   \the\markdownLaTeXTableEnd
15408   \expandafter\@gobble
15409 \fi\markdownLaTeXRenderTableRow}
15410 \def\markdownLaTeXReadAlignments#1{%
15411   \advance\markdownLaTeXColumnCounter by 1\relax
15412   \if#1d%
15413     \addto@hook\markdownLaTeXTableAlignment{1}%
15414   \else
15415     \addto@hook\markdownLaTeXTableAlignment{#1}%
15416   \fi
15417   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
15418     \expandafter\@gobble
15419   \fi\markdownLaTeXReadAlignments}
15420 \def\markdownLaTeXRenderTableCell#1{%
15421   \advance\markdownLaTeXColumnCounter by 1\relax
15422   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
15423     \addto@hook\markdownLaTeXTable{#1&}%
15424   \else
15425     \addto@hook\markdownLaTeXTable{#1\\}%
15426     \expandafter\@gobble
15427   \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.9 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

15428
15429 \markdownIfOption{lineBlocks}{%
15430   \RequirePackage{verse}
15431   \markdownSetup{rendererPrototypes={
15432     lineBlockBegin = {%
15433       \begingroup
15434         \def\markdownRendererHardLineBreak{\\}%
15435         \begin{verse}%
15436       },
15437     lineBlockEnd = {%
15438       \end{verse}%
15439     \endgroup
15440   },
15441 }}

```

```
15442 }-{}
15443
```

### 3.3.4.10 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
15444 \ExplSyntaxOn
15445 \keys_define:nn
15446   { markdown/jekyllData }
15447   {
15448     author .code:n = { \author{#1} },
15449     date   .code:n = { \date{#1}   },
15450     title  .code:n = { \title{#1}  },
15451   }
```

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.

```
15452 \markdownSetup{
15453   rendererPrototypes = {
15454     jekyllDataEnd = {
15455       \AddToHook{begindocument/end}{\maketitle}
15456     },
15457   },
15458 }
```

### 3.3.4.11 Marked Text

If the `mark` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement marked text.

```
15459 \@@_if_option:nT
15460   { mark }
15461   {
15462     \sys_if_engine luatex:TF
15463     {
15464       \RequirePackage
15465         { luacolor }
15466       \RequirePackage
15467         { lua-ul }
15468       \markdownSetup
15469         {
15470           rendererPrototypes = {
15471             mark = {
15472               \highLight
```

```

15473         { #1 }
15474     },
15475     }
15476 }
15477 }
15478 {
15479     \RequirePackage
15480     { xcolor }
15481     \RequirePackage
15482     { soul }
15483     \markdownSetup
15484     {
15485         rendererPrototypes = {
15486             mark = {
15487                 \hl
15488                 { #1 }
15489             },
15490         }
15491     }
15492 }
15493 }

```

### 3.3.4.12 Strike-Through

If the `strikeThrough` option is enabled, we will load either the `soul` package or the `lua-ul` package and use it to implement strike-throughs.

```

15494 \@@_if_option:nT
15495 { strikeThrough }
15496 {
15497     \sys_if_engine_luatex:TF
15498     {
15499         \RequirePackage
15500         { lua-ul }
15501         \markdownSetup
15502         {
15503             rendererPrototypes = {
15504                 strikeThrough = {
15505                     \strikeThrough
15506                     { #1 }
15507                 },
15508             }
15509         }
15510     }
15511     {
15512         \RequirePackage
15513         { soul }
15514         \markdownSetup

```

```

15515     {
15516         rendererPrototypes = {
15517             strikeThrough = {
15518                 \st
15519                 { #1 }
15520             },
15521         }
15522     }
15523 }
15524 }

```

### 3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form `<key>=<value>` set the corresponding keys of the `graphicx` package to the corresponding values and we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing figures.

```

15525 \seq_new:N
15526   \l_@@_image_identifiers_seq
15527 \markdownSetup {
15528   rendererPrototypes = {
15529     image = {
15530       \begin { figure }
15531         \begin { center }
15532           \includegraphics
15533             [ alt = { #1 } ]
15534             { #3 }
15535           \tl_if_empty:nF
15536             { #4 }
15537             { \caption { #4 } }
15538           \seq_map_inline:Nn
15539             \l_@@_image_identifiers_seq
15540             { \label { ##1 } }
15541         \end { center }
15542       \end { figure }
15543     },
15544   }
15545 }
15546 \@@_if_option:nT
15547   { linkAttributes }
15548   {
15549     \RequirePackage { graphicx }
15550     \markdownSetup {
15551       rendererPrototypes = {

```

```

15552     imageAttributeContextBegin = {
15553         \group_begin:
15554         \markdownSetup {
15555             rendererPrototypes = {
15556                 attributeIdentifier = {
15557                     \seq_put_right:Nn
15558                     \l_@@_image_identifiers_seq
15559                     { ##1 }
15560                 },
15561                 attributeKeyValue = {
15562                     \setkeys
15563                     { Gin }
15564                     { { ##1 } = { ##2 } }
15565                 },
15566             },
15567         }
15568     },
15569     imageAttributeContextEnd = {
15570         \group_end:
15571     },
15572 },
15573 }
15574 }
15575 \ExplSyntaxOff

```

### 3.3.4.14 Raw Attributes

In the raw block and inline raw span renderer prototypes, display the content using the default templates of the package `luaxml` when the raw attribute is `html` and default to the plain TeX renderer prototypes otherwise, translating raw attribute `latex` to `tex`.

```

15576 \ExplSyntaxOn
15577 \cs_new:Nn
15578   \@@_luaxml_print_html:n
15579   {
15580     \luabridge_now:n
15581     {
15582       local~input_file = assert(io.open(" #1 ", "r"))
15583       local~input = assert(input_file:read("*a"))
15584       assert(input_file:close())
15585       input = "<body>" .. input .. "</body>"
15586       local~dom = require("luaxml-domobject").html_parse(input)
15587       local~output = require("luaxml-htmltemplates"):process_dom(dom)
15588       print(output)
15589     }
15590   }
15591 \cs_gset_protected:Npn

```

```

15592 \markdownRendererInputRawInlinePrototype#1#2
15593 {
15594   \str_case:nnF
15595     { #2 }
15596     {
15597       { latex }
15598       {
15599         \@_plain_tex_default_input_raw_inline:nn
15600         { #1 }
15601         { tex }
15602       }
15603       { html }
15604       {

```

If we are using T<sub>E</sub>X4ht<sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15605         \cs_if_exist:NTF
15606           \HCode
15607           {
15608             \if_mode_vertical:
15609               \IgnorePar
15610               \EndP
15611             \fi:
15612             \special
15613               { t4ht* < #1 }
15614           }
15615           {
15616             \@_luaxml_print_html:n
15617               { #1 }
15618           }
15619       }
15620   }
15621 {
15622   \@_plain_tex_default_input_raw_inline:nn
15623     { #1 }
15624     { #2 }
15625 }
15626 }
15627 \cs_gset_protected:Npn
15628 \markdownRendererInputRawBlockPrototype#1#2
15629 {
15630   \str_case:nnF
15631     { #2 }
15632     {
15633       { latex }
15634       {

```

---

<sup>37</sup>See <https://tug.org/tex4ht/>.

```

15635         \@@_plain_tex_default_input_raw_block:nn
15636             { #1 }
15637             { tex }
15638         }
15639     { html }
15640     {

```

If we are using  $\text{T}\text{E}\text{X}4\text{ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15641         \cs_if_exist:NTF
15642         \HCode
15643         {
15644             \if_mode_vertical:
15645                 \IgnorePar
15646             \fi:
15647             \EndP
15648             \special
15649                 { t4ht* < #1 }
15650             \par
15651             \ShowPar
15652         }
15653     {
15654         \@@_luaxml_print_html:n
15655             { #1 }
15656     }
15657 }
15658 }
15659 {
15660     \@@_plain_tex_default_input_raw_block:nn
15661         { #1 }
15662         { #2 }
15663     }
15664 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{L}\text{A}\text{T}\text{E}\text{X}$  labels for referencing the last  $\text{L}\text{A}\text{T}\text{E}\text{X}$  counter that has been incremented in e.g. ordered lists.

```

15665 \seq_new:N
15666 \l_@@_bracketed_span_identifiers_seq
15667 \markdownSetup {
15668     rendererPrototypes = {
15669         bracketedSpanAttributeContextBegin = {
15670             \group_begin:
15671             \markdownSetup {

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

15672     rendererPrototypes = {
15673     attributeIdentifier = {
15674         \seq_put_right:Nn
15675         \l_@@_bracketed_span_identifiers_seq
15676         { ##1 }
15677     },
15678 },
15679 }
15680 },
15681 bracketedSpanAttributeContextEnd = {
15682     \seq_map_inline:Nn
15683     \l_@@_bracketed_span_identifiers_seq
15684     { \label { ##1 } }
15685     \group_end:
15686 },
15687 },
15688 }
15689 \ExplSyntaxOff
15690 \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.

```

15691 \newcommand\markdownMakeOther{%
15692     \count0=128\relax
15693     \loop
15694         \catcode\count0=11\relax
15695         \advance\count0 by 1\relax
15696     \ifnum\count0<256\repeat}%

```

## 3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the documentation is scarce) the majority of the plain TeX format required by the plain TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` L<sup>A</sup>T<sub>E</sub>X package.

```

15697 \def\markdownMakeOther{%

```



```

15698 \count0=128\relax
15699 \loop
15700 \catcode\count0=11\relax
15701 \advance\count0 by 1\relax
15702 \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.

```

15703 \catcode`|=12}%

```

### 3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```

15704 \long\def\inputmarkdown{%
15705 \dosingleempty
15706 \doinputmarkdown}%
15707 \long\def\doinputmarkdown[#1]#2{%
15708 \begingroup
15709 \iffirstargument
15710 \setupmarkdown[#1]%
15711 \fi
15712 \markdownInput{#2}%
15713 \endgroup}%
15714 \long\def\inputyaml{%
15715 \dosingleempty
15716 \doinputyaml}%
15717 \long\def\doinputyaml[#1]#2{%
15718 \doinputmarkdown
15719 [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s TeX, trailing spaces are removed very early on when a line is being put to the input buffer. [16, sec. 31]. According to Eijkhout [17, 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.

```

15720 \startluacode
15721 document.markdown_buffering = false
15722 local function preserve_trailing_spaces(line)
15723   if document.markdown_buffering then
15724     line = line:gsub("[ \t][ \t]$", "\t\t")
15725   end
15726   return line

```

```

15727   end
15728   resolvers.installinputlinehandler(preserve_trailing_spaces)
15729 \stoptluacode
15730 \begingroup
15731   \catcode`\|=0%
15732   \catcode`\|=12%
15733   |gdef|startmarkdown{%
15734     |ctxlua{document.markdown_buffering = true}%
15735     |markdownReadAndConvert{\stopmarkdown}%
15736       {|stopmarkdown}}%
15737   |gdef|stopmarkdown{%
15738     |ctxlua{document.markdown_buffering = false}%
15739     |markdownEnd}%
15740   |gdef|startyaml{%
15741     |begingroup
15742     |ctxlua{document.markdown_buffering = true}%
15743     |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
15744     |markdownReadAndConvert{\stopyaml}%
15745       {|stopyaml}}%
15746   |gdef|stopyaml{%
15747     |ctxlua{document.markdown_buffering = false}%
15748     |yamlEnd}%
15749 |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.

```

15750 \ExplSyntaxOn
15751 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
15752 \prop_new:N \g_@@_context_loaded_themes_versions_prop
15753 \cs_gset:Nn
15754   \@@_load_theme:nnn
15755   {

```

Determine whether either this is a built-in theme according to the prop `\g_@@_context_built_in_themes_prop` or a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain  $\TeX$  theme instead.

```

15756   \bool_if:nTF
15757     {
15758       \bool_lazy_or_p:nn
15759         {
15760           \prop_if_in_p:Nn
15761             \g_@@_context_built_in_themes_prop

```

```

15762         { #1 }
15763     }
15764     {
15765         \file_if_exist_p:n
15766         { t - markdown theme #3.tex }
15767     }
15768 }
15769 {
15770     \prop_get:NnNTF
15771     \g_@@_context_loaded_themes_linenos_prop
15772     { #1 }
15773     \l_tmpa_tl
15774     {
15775         \prop_get:NnN
15776         \g_@@_context_loaded_themes_versions_prop
15777         { #1 }
15778         \l_tmpb_tl
15779         \str_if_eq:nVTF
15780         { #2 }
15781         \l_tmpb_tl
15782         {
15783             \msg_warning:nnnVn
15784             { markdown }
15785             { repeatedly-loaded-context-theme }
15786             { #1 }
15787             \l_tmpa_tl
15788             { #2 }
15789         }
15790         {
15791             \msg_error:nnnnVV
15792             { markdown }
15793             { different-versions-of-context-theme }
15794             { #1 }
15795             { #2 }
15796             \l_tmpb_tl
15797             \l_tmpa_tl
15798         }
15799     }
15800 }
15801     \prop_gput:Nnx
15802     \g_@@_context_loaded_themes_linenos_prop
15803     { #1 }
15804     { \tex_the:D \tex_inputlineno:D }
15805     \prop_gput:Nnn
15806     \g_@@_context_loaded_themes_versions_prop
15807     { #1 }
15808     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

15809         \prop_if_in:NnTF
15810             \g_@@_context_built_in_themes_prop
15811             { #1 }
15812             {
15813                 \msg_info:nnnn
15814                 { markdown }
15815                 { loading-built-in-context-theme }
15816                 { #1 }
15817                 { #2 }
15818                 \prop_item:Nn
15819                 \g_@@_context_built_in_themes_prop
15820                 { #1 }
15821             }
15822             {
15823                 \msg_info:nnnn
15824                 { markdown }
15825                 { loading-context-theme }
15826                 { #1 }
15827                 { #2 }
15828                 \usemodule
15829                 [ t ]
15830                 [ markdown theme #3 ]
15831             }
15832         }
15833     }
15834     {
15835         \@@_plain_tex_load_theme:nnn
15836         { #1 }
15837         { #2 }
15838         { #3 }
15839     }
15840 }
15841 \msg_new:nnn
15842 { markdown }
15843 { loading-built-in-context-theme }
15844 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
15845 \msg_new:nnn
15846 { markdown }
15847 { loading-context-theme }
15848 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
15849 \msg_new:nnn
15850 { markdown }
15851 { repeatedly-loaded-context-theme }
15852 {
15853     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~

```

```

15854   loaded-on~line~#2,~not~loading~it~again
15855   }
15856   \msg_new:nnn
15857   { markdown }
15858   { different-versions-of-context-theme }
15859   {
15860     Tried-to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
15861     but~version~#3~has~already~been~loaded~on~line~#4
15862   }
15863   \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:

```
15864 \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.

```

15865 \markdownIfOption{plain}{\iffalse}{\iftrue}
15866 \def\markdownRendererHardLineBreakPrototype{\blank}%
15867 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
15868 \def\markdownRendererRightBracePrototype{\textbraceright}%
15869 \def\markdownRendererDollarSignPrototype{\textdollar}%
15870 \def\markdownRendererPercentSignPrototype{\percent}%
15871 \def\markdownRendererUnderscorePrototype{\textunderscore}%
15872 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
15873 \def\markdownRendererBackslashPrototype{\textbackslash}%
15874 \def\markdownRendererTildePrototype{\textasciitilde}%
15875 \def\markdownRendererPipePrototype{\char`|}%
15876 \def\markdownRendererLinkPrototype#1#2#3#4{%
15877   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
15878   \fi\tt<\hyphenatedurl{#3}>}}%
15879 \usemodule[database]
15880 \defineseparatedlist
15881   [MarkdownConTeXtCSV]
15882   [separator={,},
15883   before=\bTABLE,after=\eTABLE,
15884   first=\bTR,last=\eTR,
15885   left=\bTD,right=\eTD]
15886 \def\markdownConTeXtCSV{csv}
15887 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
15888   \def\markdownConTeXtCSV@arg{#1}%

```

```

15889 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
15890 \placetable[] [tab:#1]{#4}{%
15891 \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
15892 \else
15893 \markdownInput{#3}%
15894 \fi}%
15895 \def\markdownRendererImagePrototype#1#2#3#4{%
15896 \placefigure[] []{#4}{\externalfigure[#3]}%
15897 \def\markdownRendererUlBeginPrototype{\startitemize}%
15898 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
15899 \def\markdownRendererUlItemPrototype{\item}%
15900 \def\markdownRendererUlEndPrototype{\stopitemize}%
15901 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
15902 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
15903 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
15904 \def\markdownRendererOlItemPrototype{\item}%
15905 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
15906 \def\markdownRendererOlEndPrototype{\stopitemize}%
15907 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
15908 \definedescription
15909 [MarkdownConTeXtDlItemPrototype]
15910 [location=hanging,
15911 margin=standard,
15912 headstyle=bold]%
15913 \definestartstop
15914 [MarkdownConTeXtDlPrototype]
15915 [before=\blank,
15916 after=\blank]%
15917 \definestartstop
15918 [MarkdownConTeXtDlTightPrototype]
15919 [before=\blank\startpacked,
15920 after=\stoppacked\blank]%
15921 \def\markdownRendererDlBeginPrototype{%
15922 \startMarkdownConTeXtDlPrototype}%
15923 \def\markdownRendererDlBeginTightPrototype{%
15924 \startMarkdownConTeXtDlTightPrototype}%
15925 \def\markdownRendererDlItemPrototype#1{%
15926 \startMarkdownConTeXtDlItemPrototype{#1}}%
15927 \def\markdownRendererDlItemEndPrototype{%
15928 \stopMarkdownConTeXtDlItemPrototype}%
15929 \def\markdownRendererDlEndPrototype{%
15930 \stopMarkdownConTeXtDlPrototype}%
15931 \def\markdownRendererDlEndTightPrototype{%
15932 \stopMarkdownConTeXtDlTightPrototype}%
15933 \def\markdownRendererEmphasisPrototype#1{\em#1}%
15934 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
15935 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%

```

```

15936 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
15937 \def\markdownRendererLineBlockBeginPrototype{%
15938   \begingroup
15939     \def\markdownRendererHardLineBreak{
15940       }%
15941     \startlines
15942   }%
15943 \def\markdownRendererLineBlockEndPrototype{%
15944   \stoptlines
15945 \endgroup
15946 }%
15947 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

15948 \ExplSyntaxOn
15949 \cs_gset:Npn
15950 \markdownRendererInputFencedCodePrototype#1#2#3
15951 {
15952   \tl_if_empty:nTF
15953     { #2 }
15954     { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConT<sub>E</sub>Xt `\definetying` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
  ~~~ latex
  \documentclass{article}
  \begin{document}
  Hello world!
  \end{document}
  ~~~
  \stopmarkdown
\stoptext

```

```

15955   {
15956     \regex_extract_once:nnN

```

```

15957         { \w* }
15958         { #2 }
15959         \l_tmpa_seq
15960         \seq_pop_left:NN
15961         \l_tmpa_seq
15962         \l_tmpa_tl
15963         \typefile[\l_tmpa_tl] []{#1}
15964     }
15965 }
15966 \ExplSyntaxOff
15967 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
15968 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
15969 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
15970 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
15971 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
15972 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
15973 \def\markdownRendererThematicBreakPrototype{%
15974   \blackrule[height=1pt, width=\hsize]}%
15975 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
15976 \def\markdownRendererTickedBoxPrototype{\boxtimes$}
15977 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
15978 \def\markdownRendererUntickedBoxPrototype{\square$}
15979 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
15980 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
15981 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
15982 \def\markdownRendererDisplayMathPrototype#1{%
15983   \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

15984 \newcount\markdownConTeXtRowCounter
15985 \newcount\markdownConTeXtRowTotal
15986 \newcount\markdownConTeXtColumnCounter
15987 \newcount\markdownConTeXtColumnTotal
15988 \newtoks\markdownConTeXtTable
15989 \newtoks\markdownConTeXtTableFloat
15990 \def\markdownRendererTablePrototype#1#2#3{%
15991   \markdownConTeXtTable={}%
15992   \ifx\empty#1\empty
15993     \markdownConTeXtTableFloat={%
15994       \the\markdownConTeXtTable}%
15995   \else
15996     \markdownConTeXtTableFloat={%
15997       \placetable{#1}{\the\markdownConTeXtTable}}%
15998   \fi
15999   \begingroup

```



```

16000 \setupTABLE[r][each][topframe=off, bottomframe=off,
16001         leftframe=off, rightframe=off]
16002 \setupTABLE[c][each][topframe=off, bottomframe=off,
16003         leftframe=off, rightframe=off]
16004 \setupTABLE[r][1][topframe=on, bottomframe=on]
16005 \setupTABLE[r][#1][bottomframe=on]
16006 \markdownConTeXtRowCounter=0%
16007 \markdownConTeXtRowTotal=#2%
16008 \markdownConTeXtColumnTotal=#3%
16009 \markdownConTeXtRenderTableRow}
16010 \def\markdownConTeXtRenderTableRow#1{%
16011     \markdownConTeXtColumnCounter=0%
16012     \ifnum\markdownConTeXtRowCounter=0\relax
16013         \markdownConTeXtReadAlignments#1%
16014         \markdownConTeXtTable={\bTABLE}%
16015     \else
16016         \markdownConTeXtTable=\expandafter{%
16017             \the\markdownConTeXtTable\bTR}%
16018         \markdownConTeXtRenderTableCell#1%
16019         \markdownConTeXtTable=\expandafter{%
16020             \the\markdownConTeXtTable\eTR}%
16021     \fi
16022     \advance\markdownConTeXtRowCounter by 1\relax
16023     \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
16024         \markdownConTeXtTable=\expandafter{%
16025             \the\markdownConTeXtTable\eTABLE}%
16026         \the\markdownConTeXtTableFloat
16027         \endgroup
16028         \expandafter\gobbleoneargument
16029     \fi\markdownConTeXtRenderTableRow}
16030 \def\markdownConTeXtReadAlignments#1{%
16031     \advance\markdownConTeXtColumnCounter by 1\relax
16032     \if#1d%
16033         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16034     \fi\if#1l%
16035         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
16036     \fi\if#1c%
16037         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
16038     \fi\if#1r%
16039         \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
16040     \fi
16041     \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16042     \else
16043         \expandafter\gobbleoneargument
16044     \fi\markdownConTeXtReadAlignments}
16045 \def\markdownConTeXtRenderTableCell#1{%
16046     \advance\markdownConTeXtColumnCounter by 1\relax

```

```

16047 \markdownConTeXtTable=\expandafter{%
16048   \the\markdownConTeXtTable\bTD#1\eTD}%
16049 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16050 \else
16051   \expandafter\gobbleoneargument
16052 \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`.

```

16053 \ExplSyntaxOn
16054 \cs_gset:Npn
16055   \markdownRendererInputRawInlinePrototype#1#2
16056   {
16057     \str_case:nnF
16058       { #2 }
16059       {
16060         { latex }
16061         {
16062           \@@_plain_tex_default_input_raw_inline:nn
16063             { #1 }
16064             { context }
16065         }
16066       }
16067     {
16068       \@@_plain_tex_default_input_raw_inline:nn
16069         { #1 }
16070         { #2 }
16071     }
16072   }
16073 \cs_gset:Npn
16074   \markdownRendererInputRawBlockPrototype#1#2
16075   {
16076     \str_case:nnF
16077       { #2 }
16078       {
16079         { context }
16080         {
16081           \@@_plain_tex_default_input_raw_block:nn
16082             { #1 }
16083             { tex }
16084         }
16085       }
16086     {
16087       \@@_plain_tex_default_input_raw_block:nn
16088         { #1 }

```

```

16089         { #2 }
16090     }
16091 }
16092 \cs_gset_eq:NN
16093   \markdownRendererInputRawBlockPrototype
16094   \markdownRendererInputRawInlinePrototype
16095 \fi % Closes ` \markdownIfOption{plain}{\iffalse}{\iftrue}`
16096 \ExplSyntaxOff
16097 \stopmodule
16098 \protect

```

At the end of the ConT<sub>E</sub>Xt module, we load the `witiko/markdown/defaults` ConT<sub>E</sub>Xt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

16099 \ExplSyntaxOn
16100 \str_if_eq:VVT
16101   \c_@@_top_layer_tl
16102   \c_@@_option_layer_context_tl
16103   {
16104     \ExplSyntaxOff
16105     \@@_if_option:nF
16106       { noDefaults }
16107       {
16108         \@@_if_option:nTF
16109           { experimental }
16110           {
16111             \@@_setup:n
16112               { theme = witiko/markdown/defaults@experimental }
16113           }
16114         {
16115             \@@_setup:n
16116               { theme = witiko/markdown/defaults }
16117         }
16118       }
16119     \ExplSyntaxOn
16120   }
16121 \ExplSyntaxOff
16122 \stopmodule
16123 \protect

```

## References

- [1] LuaT<sub>E</sub>X development team. *LuaT<sub>E</sub>X 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] Frank Mittelbach, Ulrike Fischer, and L<sup>A</sup>T<sub>E</sub>X Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).
- [3] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [4] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [5] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [6] 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).
- [7] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [8] 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).
- [9] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [10] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [11] 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).
- [12] Frank Mittelbach. *L<sup>A</sup>T<sub>E</sub>X’s hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [13] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [14] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

- [15] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X<sub>2 $\epsilon$</sub>  Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [16] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [17] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

autoIdentifiers	21, 33, 82, 98
blankBeforeBlockquote	21
blankBeforeCodeFence	22
blankBeforeDivFence	22
blankBeforeHeading	22
blankBeforeList	22
bracketedSpans	23, 84, 447
breakableBlockquotes	23
cacheDir	4, 17, 19, 59, 156, 169, 370, 385, 399
citationNbsps	23
citations	24, 87, 88
codeSpans	24
contentBlocks	20, 25, 34
contentBlocksLanguageMap	20
contentLevel	25
debugExtensions	9, 20, 26, 314
debugExtensionsFileName	20, 26
defaultOptions	10, 51, 368, 370
definitionLists	26, 92
eagerCache	17, 368
ensureJekyllData	27
entities.char_entity	215
entities.dec_entity	215
entities.hex_entity	215
entities.hex_entity_with_x_char	215
escape_minimal	219
escape_programmatic_text	219
escape_typographic_text	219

expandtabs	274
expectJekyllData	27, 27
experimental	5, 17, 417
extensions	28, 164, 319
extensions.bracketed_spans	320
extensions.citations	321
extensions.content_blocks	325
extensions.definition_lists	328
extensions.fancy_lists	330
extensions.fenced_code	336
extensions.fenced_divs	342
extensions.header_attributes	346
extensions.inline_code_attributes	348
extensions.jekyll_data	364
extensions.line_blocks	348
extensions.link_attributes	350
extensions.mark	349
extensions.notes	351
extensions.pipe_table	353
extensions.raw_inline	358
extensions.strike_through	359
extensions.subscripts	359
extensions.superscripts	360
extensions.tex_math	361
fancyLists	30, 109–114, 417
fencedCode	30, 39, 89, 96, 115, 382
fencedCodeAttributes	31, 82, 96
fencedDiv	97
fencedDivs	31, 41
finalizeCache	17, 20, 32, 32, 59, 155, 368, 369
frozenCache	20, 32, 59, 74, 75, 155, 382, 385
frozenCacheCounter	32, 369, 407, 408
frozenCacheFileName	20, 32, 59, 369
gfmAutoIdentifiers	21, 32, 82, 98
hashEnumerators	33
headerAttributes	33, 41, 82, 98
html	34, 101, 431
hybrid	34, 34, 40, 46, 48, 62, 75, 116, 156, 220, 275, 407
inlineCodeAttributes	35, 82, 90

inlineNotes	36
\input	55, 56
\inputmarkdown	159, 160, 161, 449
inputTempFileName	60, 62, 401, 402, 404, 405
\inputyaml	159, 161, 449
iterlines	274
jeekyllData	3, 27, 28, 36, 125–128, 131
\l_file_search_path_seq	406
languages_json	325, 325
lineBlocks	37, 104
linkAttributes	37, 82, 102, 106, 295, 444
mark	38, 107, 442
\markdown	151, 152, 410
markdown	150, 150, 151, 409, 410
markdown*	150, 150, 151, 155, 409
\markdown_jeekyll_data_concatenate_address:NN	393
\markdown_jeekyll_data_pop:	393
\markdown_jeekyll_data_push:nN	393
\markdown_jeekyll_data_push_address_segment:n	392
\markdown_jeekyll_data_set_keyval:Nn	394
\markdown_jeekyll_data_set_keyvals:nn	394
\markdown_jeekyll_data_update_address_tls:	393
\markdownBegin	53, 53–56, 148–150, 152, 159, 160
\markdownCleanup	400
\markdownConvert	399
\markdownEnd	53, 53–56, 149–152, 159, 160
\markdownError	148, 148
\markdownEscape	53, 56, 408
\markdownIfOption	58
\markdownIfSnippetExists	77
\markdownInfo	148, 148
\markdownInput	53, 55, 56, 150, 153, 155, 160, 406, 409
\markdownInputFilename	399
\markdownInputFileStream	400
\markdownInputPlainTeX	409
\markdownLoadPlainTeXTheme	157, 163, 380
\markdownLuaExecute	403, 406
\markdownLuaOptions	396, 399
\markdownMakeOther	148, 448
\markdownOptionFinalizeCache	59

<code>\markdownOptionFrozenCache</code>	59
<code>\markdownOptionHybrid</code>	62
<code>\markdownOptionInputTempFileName</code>	59
<code>\markdownOptionNoDefaults</code>	61
<code>\markdownOptionOutputDir</code>	60, 60, 63
<code>\markdownOptionPlain</code>	60
<code>\markdownOptionStripPercentSigns</code>	61
<code>\markdownOutputFileStream</code>	400
<code>\markdownPrepare</code>	399
<code>\markdownPrepareInputFilename</code>	399
<code>\markdownPrepareLuaOptions</code>	396
<code>\markdownReadAndConvert</code>	148, 400, 409, 410, 449
<code>\markdownReadAndConvertProcessLine</code>	401, 402
<code>\markdownReadAndConvertStripPercentSigns</code>	401
<code>\markdownReadAndConvertTab</code>	400
<code>\markdownRendererAttributeName</code>	82
<code>\markdownRendererAttributeIdentifier</code>	82
<code>\markdownRendererAttributeKeyValue</code>	83
<code>\markdownRendererBlockQuoteBegin</code>	84
<code>\markdownRendererBlockQuoteEnd</code>	84
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	84
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	84
<code>\markdownRendererCite</code>	87, 88
<code>\markdownRendererCodeSpan</code>	89
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	90
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	90
<code>\markdownRendererContentBlock</code>	90, 91
<code>\markdownRendererContentBlockCode</code>	91
<code>\markdownRendererContentBlockOnlineImage</code>	91
<code>\markdownRendererDisplayMath</code>	122
<code>\markdownRendererDlBegin</code>	92
<code>\markdownRendererDlBeginTight</code>	92
<code>\markdownRendererDlDefinitionBegin</code>	94
<code>\markdownRendererDlDefinitionEnd</code>	94
<code>\markdownRendererDlEnd</code>	94
<code>\markdownRendererDlEndTight</code>	95
<code>\markdownRendererDlItem</code>	93
<code>\markdownRendererDlItemEnd</code>	93
<code>\markdownRendererDocumentBegin</code>	107
<code>\markdownRendererDocumentEnd</code>	107
<code>\markdownRendererEllipsis</code>	41, 95
<code>\markdownRendererEmphasis</code>	96, 135



<code>\markdownRendererError</code>	124
<code>\markdownRendererFancyOlBegin</code>	109, 110
<code>\markdownRendererFancyOlBeginTight</code>	110
<code>\markdownRendererFancyOlEnd</code>	114
<code>\markdownRendererFancyOlEndTight</code>	114
<code>\markdownRendererFancyOlItem</code>	112
<code>\markdownRendererFancyOlItemEnd</code>	112
<code>\markdownRendererFancyOlItemWithNumber</code>	112
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	96
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	96
<code>\markdownRendererFencedDivAttributeContextBegin</code>	97
<code>\markdownRendererFencedDivAttributeContextEnd</code>	97
<code>\markdownRendererHalfTickedBox</code>	123
<code>\markdownRendererHardLineBreak</code>	105
<code>\markdownRendererHeaderAttributeContextBegin</code>	98
<code>\markdownRendererHeaderAttributeContextEnd</code>	98
<code>\markdownRendererHeadingFive</code>	100
<code>\markdownRendererHeadingFour</code>	100
<code>\markdownRendererHeadingOne</code>	99
<code>\markdownRendererHeadingSix</code>	100
<code>\markdownRendererHeadingThree</code>	99
<code>\markdownRendererHeadingTwo</code>	99
<code>\markdownRendererImage</code>	102
<code>\markdownRendererImageAttributeContextBegin</code>	103
<code>\markdownRendererImageAttributeContextEnd</code>	103
<code>\markdownRendererInlineHtmlComment</code>	101
<code>\markdownRendererInlineHtmlTag</code>	101
<code>\markdownRendererInlineMath</code>	122
<code>\markdownRendererInputBlockHtmlElement</code>	101
<code>\markdownRendererInputFencedCode</code>	89
<code>\markdownRendererInputRawBlock</code>	115
<code>\markdownRendererInputRawInline</code>	114
<code>\markdownRendererInputVerbatim</code>	88
<code>\markdownRendererInterblockSeparator</code>	103
<code>\markdownRendererJekyllDataBegin</code>	125
<code>\markdownRendererJekyllDataBoolean</code>	127
<code>\markdownRendererJekyllDataEmpty</code>	131
<code>\markdownRendererJekyllDataEnd</code>	125
<code>\markdownRendererJekyllDataMappingBegin</code>	126
<code>\markdownRendererJekyllDataMappingEnd</code>	126
<code>\markdownRendererJekyllDataNumber</code>	128
<code>\markdownRendererJekyllDataProgrammaticString</code>	128, 128, 129

<code>\markdownRendererJekyllDataSequenceBegin</code>	127
<code>\markdownRendererJekyllDataSequenceEnd</code>	127
<code>\markdownRendererJekyllDataString</code>	129, 133
<code>\markdownRendererJekyllDataStringPrototype</code>	144
<code>\markdownRendererJekyllDataTypographicString</code>	128, 128, 129, 365
<code>\markdownRendererLineBlockBegin</code>	104
<code>\markdownRendererLineBlockEnd</code>	104
<code>\markdownRendererLink</code>	106, 135
<code>\markdownRendererLinkAttributeContextBegin</code>	106
<code>\markdownRendererLinkAttributeContextEnd</code>	106
<code>\markdownRendererMark</code>	107
<code>\markdownRendererNbsp</code>	108
<code>\markdownRendererNote</code>	108
<code>\markdownRendererOlBegin</code>	109
<code>\markdownRendererOlBeginTight</code>	109
<code>\markdownRendererOlEnd</code>	113
<code>\markdownRendererOlEndTight</code>	113
<code>\markdownRendererOlItem</code>	42, 110
<code>\markdownRendererOlItemEnd</code>	111
<code>\markdownRendererOlItemWithNumber</code>	42, 111
<code>\markdownRendererParagraphSeparator</code>	104
<code>\markdownRendererReplacementCharacter</code>	116
<code>\markdownRendererSectionBegin</code>	115
<code>\markdownRendererSectionEnd</code>	115
<code>\markdownRendererSoftLineBreak</code>	105
<code>\markdownRendererStrikeThrough</code>	119
<code>\markdownRendererStrongEmphasis</code>	96
<code>\markdownRendererSubscript</code>	120
<code>\markdownRendererSuperscript</code>	120
<code>\markdownRendererTable</code>	121
<code>\markdownRendererTableAttributeContextBegin</code>	121
<code>\markdownRendererTableAttributeContextEnd</code>	121
<code>\markdownRendererTextCite</code>	88
<code>\markdownRendererThematicBreak</code>	123
<code>\markdownRendererTickedBox</code>	123
<code>\markdownRendererUlBegin</code>	85
<code>\markdownRendererUlBeginTight</code>	85
<code>\markdownRendererUlEnd</code>	87
<code>\markdownRendererUlEndTight</code>	87
<code>\markdownRendererUlItem</code>	86
<code>\markdownRendererUlItemEnd</code>	86
<code>\markdownRendererUntickedBox</code>	123

<code>\markdownRendererWarning</code>	124
<code>\markdownSetup</code>	57, 58, 62, 154, 155, 161, 410, 417
<code>\markdownSetupSnippet</code>	76, 76
<code>\markdownThemeVersion</code>	69, 69
<code>\markdownWarning</code>	148, 148
<code>\markinline</code>	53, 55, 150, 153, 404, 408
<code>\markinlinePlainTeX</code>	408
<code>new</code>	7, 18, 368, 370
<code>notes</code>	38, 108
<code>parsers</code>	235, 273, 274
<code>parsers.commented_line</code>	254
<code>parsers.punctuation</code>	236
<code>pipeTables</code>	7, 39, 45, 121
<code>preserveTabs</code>	39, 43, 274
<code>rawAttribute</code>	35, 39, 40, 115
<code>reader</code>	8, 29, 164, 235, 273, 320
<code>reader-&gt;add_special_character</code>	8, 9, 29, 314
<code>reader-&gt;auto_link_email</code>	303
<code>reader-&gt;auto_link_url</code>	303
<code>reader-&gt;create_parser</code>	274
<code>reader-&gt;finalize_grammar</code>	309, 375
<code>reader-&gt;initialize_named_group</code>	314
<code>reader-&gt;insert_pattern</code>	8, 9, 29, 310, 316
<code>reader-&gt;lookup_note_reference</code>	288
<code>reader-&gt;lookup_reference</code>	288
<code>reader-&gt;normalize_tag</code>	274
<code>reader-&gt;options</code>	273
<code>reader-&gt;parser_functions</code>	274
<code>reader-&gt;parser_functions.name</code>	274
<code>reader-&gt;parsers</code>	273, 273, 274
<code>reader-&gt;register_link</code>	288
<code>reader-&gt;update_rule</code>	310, 313, 316
<code>reader-&gt;writer</code>	273
<code>reader.new</code>	273, 273, 375
<code>relativeReferences</code>	40
<code>\setupmarkdown</code>	161, 162
<code>\setupyaml</code>	162
<code>shiftHeadings</code>	7, 41
<code>singletonCache</code>	18

slice	7, 41, 216, 228, 229
smartEllipses	41, 95, 156
\startmarkdown	159, 159, 449
startNumber	42, 110–112
\startyaml	159, 159, 160, 449
\stopmarkdown	159, 159, 449
\stopyaml	159, 159, 160, 449
strikeThrough	42, 119, 443
stripIndent	43, 275
stripPercentSigns	400, 401
subscripts	43, 120
superscripts	44, 120
syntax	311, 315
tableAttributes	44, 121, 439
tableCaptions	7, 44, 45, 121
taskLists	45, 123, 430
texComments	46, 275
texMathDollars	35, 46, 122
texMathDoubleBackslash	35, 47, 122
texMathSingleBackslash	35, 47, 122
tightLists	47, 85, 87, 92, 95, 109, 110, 113, 114, 417
underscores	48
unicodeNormalization	18, 19
unicodeNormalizationForm	18, 19
util.cache	164, 165
util.cache_verbatim	165
util.encode_json_string	165
util.err	164
util.escaper	167
util.expand_tabs_in_line	165
util.flatten	166
util.intersperse	167
util.map	167
util.pathname	168
util.rope_last	167
util.rope_to_string	166
util.salt	169
util.table_copy	165
util.walk	165, 166, 167
util.warning	169

walkable_syntax	8, 20, 26, 309, 310, 313–315
writer	164, 164, 216, 320
writer->active_attributes	227, 227–229
writer->attribute_type_levels	227
writer->attributes	225
writer->block_html_element	223
writer->blockquote	224
writer->bulletitem	222
writer->bulletlist	222
writer->quotations	321
writer->code	220
writer->contentblock	326
writer->defer_call	235, 235
writer->definitionlist	328
writer->display_math	361
writer->div_begin	342
writer->div_end	342
writer->document	224
writer->ellipsis	218
writer->emphasis	224
writer->error	220
writer->escape	219
writer->escaped_chars	219, 219
writer->escaped_minimal_strings	218, 219
writer->escaped_strings	219
writer->escaped_uri_chars	218, 219
writer->fancyitem	332
writer->fancylist	331
writer->fencedCode	337
writer->flatten_inlines	216, 216
writer->get_state	234
writer->hard_line_break	218
writer->heading	232
writer->identifier	220
writer->image	221
writer->infostring	220
writer->inline_html_comment	223
writer->inline_html_tag	223
writer->inline_math	361
writer->interblocksep	217
writer->is_writing	216, 216
writer->jekyllData	365

writer->lineblock	348
writer->link	221
writer->mark	349
writer->math	220
writer->nbsp	217
writer->note	352
writer->options	216
writer->ordereditem	223
writer->orderedlist	222
writer->paragraph	217
writer->paragraphsep	217
writer->plain	217
writer->pop_attributes	228, 228, 229
writer->push_attributes	228, 228, 229
writer->rawBlock	337
writer->rawInline	358
writer->set_state	234
writer->slice_begin	216
writer->slice_end	216
writer->soft_line_break	218
writer->space	217
writer->span	320
writer->strike_through	359
writer->string	220
writer->strong	224
writer->subscript	359
writer->superscript	360
writer->table	355
writer->thematic_break	218
writer->checkbox	224
writer->undosep	218, 318
writer->uri	220
writer->verbatim	224
writer->warning	169, 220
writer.new	216, 216, 375
\yaml	152
yaml	150, 152, 409
\yamlBegin	53, 54, 148, 149, 152, 159
\yamlEnd	53, 54, 149, 152, 159
\yamlInput	53, 56, 150, 153, 161, 409
\yamlSetup	58