

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

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

Version 3.10.0-0-g626df6ad  
2025-01-27

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>	<b>3</b>	<b>Implementation</b>	<b>166</b>
1.1	Requirements . . . . .	2	3.1	Lua Implementation . . . . .	167
1.2	Feedback . . . . .	6	3.2	Plain T <sub>E</sub> X Implementation	380
1.3	Acknowledgements . . . . .	7	3.3	L <sup>A</sup> T <sub>E</sub> X Implementation . . . . .	421
<b>2</b>	<b>Interfaces</b>	<b>7</b>	3.4	ConT <sub>E</sub> Xt Implementation	462
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 . . . . .	152			
2.4	ConT <sub>E</sub> Xt Interface . . . . .	161			
				<b>References</b>	<b>473</b>
				<b>Index</b>	<b>475</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	An example directed graph . . . . .	75
5	An example mindmap . . . . .	76
6	An example UML sequence diagram . . . . .	77
7	The banner of the Markdown package . . . . .	78
8	A pushdown automaton that recognizes T <sub>E</sub> X comments . . . . .	258

## 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.

---

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

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

Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.<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 LuaTeX engine (though not necessarily in the LuaMetaTeX engine).

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

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

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

```
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 LuaTeX (TeX Live  $\geq$  2008).

---

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

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

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

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

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

```
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 Lua $\TeX$  engine [1, Section 4.3]. Beside these, we also include the following third-party Lua libraries:

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

```
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 $\TeX$ Requirements

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

**expl3** A package that enables the expl3 language [2] from the L $\text{\AA}$ T $\text{\E}$ X3 kernel in  $\TeX$  Live  $\leq$  2019. It is used to implement reflection capabilities that allow us to enumerate and inspect high-level concepts such as options, renderers, and renderer prototypes.

```
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 [3] 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 [4] is gratefully acknowledged.

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

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

## 2 Interfaces

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

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

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to  $\text{T}_{\text{E}}\text{X}$  *token renderers* is exposed by the Lua layer. The plain  $\text{T}_{\text{E}}\text{X}$  layer exposes the conversion capabilities of Lua as  $\text{T}_{\text{E}}\text{X}$  macros. The  $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$  and  $\text{ConT}_{\text{E}}\text{Xt}$  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

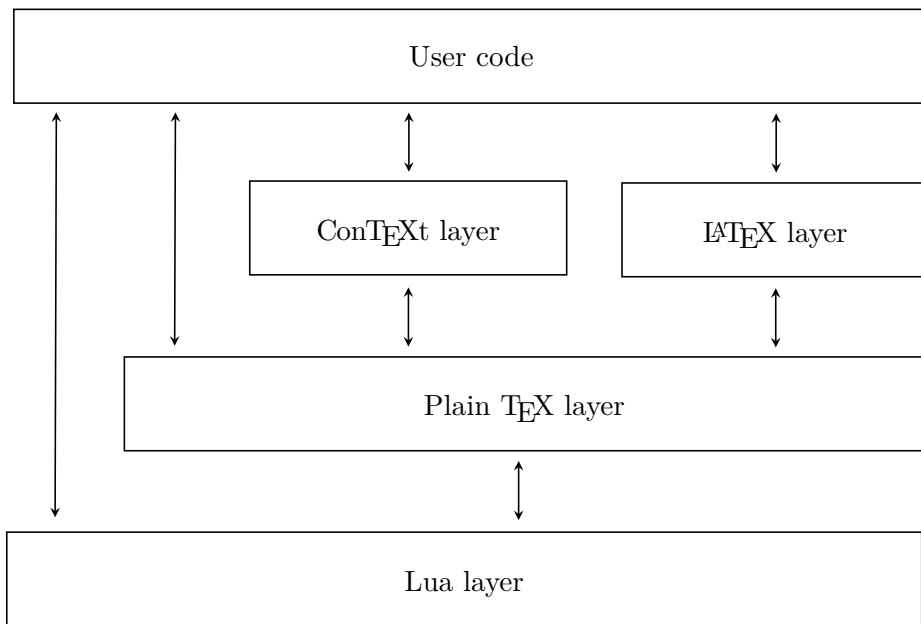
The Lua interface provides the conversion from UTF-8 encoded markdown to plain  $\text{T}_{\text{E}}\text{X}$ . This interface is used by the plain  $\text{T}_{\text{E}}\text{X}$  implementation (see Section 3.2) and will be of interest to the developers of other packages and Lua modules.

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

```

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

```

### 2.1.2 User-Defined Syntax Extensions

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

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

```

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: } % noqa: W200
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**=*<filename>*  
 default: `markdown-languages.json`

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

```

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

```

**debugExtensionsFileName**=*<filename>* default: `debug-extensions.json`

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

```

436 \@@_add_lua_option:nnn
437   { debugExtensionsFileName }
438   { path }
439   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
440 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

The frozen cache makes it possible to later typeset a plain T<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 [5]:

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

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

```
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`= $\langle$ *shift amount* $\rangle$  default: 0

All headings will be shifted by  $\langle$ *shift amount* $\rangle$ , 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  $\langle$ *shift amount* $\rangle$  is positive, and to level 1, when  $\langle$ *shift amount* $\rangle$  is negative.

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

`slice`= $\langle$ *the beginning and the end of a slice* $\rangle$  default:  $\wedge$   $\$$

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

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

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

`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 `\markdownRendererOListItemWithNumber` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOListItem` 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:nnn
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:nnn
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:nnn
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:nnn
732 { underscores }
733 { boolean }
734 { true }
735 \ExplSyntaxOff

736 defaultOptions.underscores = true

```

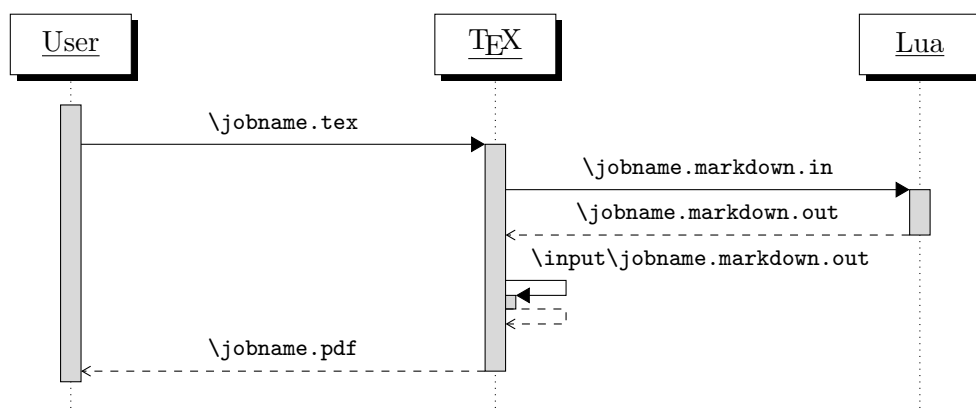


### 2.1.7 Command-Line Interface

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

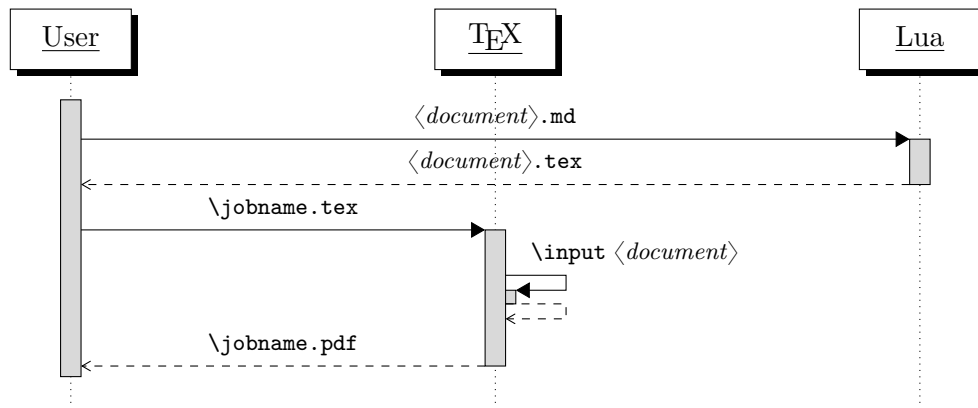
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted T<sub>E</sub>X documents are cached on the file system, taking up increasing amount of space. Unless the T<sub>E</sub>X engine includes a Lua interpreter, the package also requires 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 T<sub>E</sub>X is also provided, see Figure 3.



**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T<sub>E</sub>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 (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 [6] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-Case variants of options. As a bonus, studies [7] also show that `snake_case` is faster to read than `camelCase`.

```

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 [8, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

The `\markdownBegin` and `\markdownEnd` macros will also consume the rest of the lines at which they appear. In the following example plain 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 `\markdownEnd` macros:

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

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

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

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

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

```
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
933 \markdownIfOption
934 \@@_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 TeX 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 TeX document without invoking Lua. As a result, the plain TeX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected. It defaults to `false`.

```

935 \@@_add_plain_tex_option:nnn
936 { frozenCache }
937 { boolean }
938 { false }

```

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

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

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

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

```

939 \@@_add_plain_tex_option:nnn
940 { inputTempFileName }
941 { path }

```

```
942 { \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.

```
943 \@@_add_plain_tex_option:nnn
944 { outputDir }
945 { path }
946 { . }
```

### 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.

```
947 \@@_add_plain_tex_option:nnn
948   { plain }
949   { boolean }
950   { 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.

```
951 \@@_add_plain_tex_option:nnn
952   { noDefaults }
953   { boolean }
954   { false }
```

#### 2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing  $\TeX$  package documentation using the Doc  $\LaTeX$  package [9] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
955 \seq_gput_right:Nn
956   \g_@@_plain_tex_options_seq
957   { stripPercentSigns }
958 \prop_gput:Nnn
959   \g_@@_plain_tex_option_types_prop
960   { stripPercentSigns }
961   { boolean }
962 \prop_gput:Nnx
963   \g_@@_default_plain_tex_options_prop
964   { stripPercentSigns }
965   { 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.

```
966 \cs_new:Nn
967   \@@_define_option_commands_and_keyvals:
968   {
969     \seq_map_inline:Nn
970       \g_@@_option_layers_seq
971       {
972         \seq_map_inline:cn
973           { g_@@_ ##1 _options_seq }
974           {
975             \@@_define_option_command:n
976               { #####1 }
```

To make it easier to copy-and-paste options from Pandoc [6] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-Case variants of options. As a bonus, studies [7] also show that `snake_case` is faster to read than `camelCase`.

```
977         \@@_with_various_cases:nn
978           { #####1 }
979         {
980           \@@_define_option_keyval:nnn
981             { ##1 }
982             { #####1 }
983             { #####1 }
984         }
985     }
986 }
987 }
988 \cs_new:Nn
989   \@@_define_option_command:n
990   {
```

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.

```

991   \str_if_eq:nnTF
992     { #1 }
993     { outputDir }
994     { \@@_define_option_command_output_dir: }
995     {

```

Do not override options defined before loading the package.

```

996     \@@_option_tl_to_csname:nN
997     { #1 }
998     \l_tmpa_tl
999     \cs_if_exist:cF
1000    { \l_tmpa_tl }
1001    {
1002      \@@_get_default_option_value:nN
1003      { #1 }
1004      \l_tmpa_tl
1005      \@@_set_option_value:nV
1006      { #1 }
1007      \l_tmpa_tl
1008    }
1009  }
1010 }
1011 \ExplSyntaxOff
1012 \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.

```

1013 \ExplSyntaxOn
1014 \cs_new:Nn
1015   \@@_define_option_command_output_dir:
1016   {
1017     \cs_if_free:NT
1018       \markdownOptionOutputDir
1019       {
1020         \bool_if:nTF
1021           {
1022             \cs_if_exist_p:N
1023               \luabridge_tl_set:Nn &&
1024             (
1025               \int_compare_p:nNn
1026                 { \g_luabridge_method_int }
1027                 =
1028                 { \c_luabridge_method_directlua_int } ||

```

```

1029         \sys_if_shell_unrestricted_p:
1030     )
1031 }
1032 {

```

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.

```

1033     \group_begin:
1034     \cctab_select:N
1035     \c_str_cctab
1036     \luabridge_tl_set:Nn
1037     \l_tmpa_tl
1038     { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
1039     \tl_gset:NV
1040     \markdownOptionOutputDir
1041     \l_tmpa_tl
1042     \group_end:
1043 }
1044 {
1045     \tl_gset:Nn
1046     \markdownOptionOutputDir
1047     { . }
1048 }
1049 }
1050 }
1051 \cs_new:Nn
1052 \@@_set_option_value:nn
1053 {
1054     \@@_define_option:n
1055     { #1 }
1056     \@@_get_option_type:nN
1057     { #1 }
1058     \l_tmpa_tl
1059     \str_if_eq:NNTF
1060     \c_@@_option_type_counter_tl
1061     \l_tmpa_tl
1062     {
1063         \@@_option_tl_to_csname:nN
1064         { #1 }
1065         \l_tmpa_tl
1066         \int_gset:cn
1067         { \l_tmpa_tl }
1068         { #2 }
1069     }
1070     {
1071         \@@_option_tl_to_csname:nN

```



```

1072         { #1 }
1073         \l_tmpa_tl
1074     \cs_set:cpn
1075         { \l_tmpa_tl }
1076         { #2 }
1077     }
1078 }
1079 \cs_generate_variant:Nn
1080 \@@_set_option_value:nn
1081 { nV }
1082 \cs_new:Nn
1083 \@@_define_option:n
1084 {
1085     \@@_option_tl_to_csname:nN
1086     { #1 }
1087     \l_tmpa_tl
1088     \cs_if_free:cT
1089     { \l_tmpa_tl }
1090     {
1091         \@@_get_option_type:nN
1092         { #1 }
1093         \l_tmpb_tl
1094         \str_if_eq:NNT
1095         \c_@@_option_type_counter_tl
1096         \l_tmpb_tl
1097         {
1098             \@@_option_tl_to_csname:nN
1099             { #1 }
1100             \l_tmpa_tl
1101             \int_new:c
1102             { \l_tmpa_tl }
1103         }
1104     }
1105 }
1106 \cs_new:Nn
1107 \@@_define_option_keyval:nnn
1108 {
1109     \prop_get:cnN
1110     { g_@@_ #1 _option_types_prop }
1111     { #2 }
1112     \l_tmpa_tl
1113     \str_if_eq:VVTF
1114     \l_tmpa_tl
1115     \c_@@_option_type_boolean_tl
1116     {
1117         \keys_define:nn
1118         { markdown/options }

```

```
1119         {
```

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

```
1120         #3 .code:n = {
1121         \tl_set:Nx
1122         \l_tmpa_tl
1123         {
1124         \str_case:nnF
1125         { ##1 }
1126         {
1127         { yes } { true }
1128         { no } { false }
1129         }
1130         { ##1 }
1131         }
1132         \@@_set_option_value:nV
1133         { #2 }
1134         \l_tmpa_tl
1135         },
1136         #3 .default:n = { true },
1137         }
1138     }
1139     {
1140     \keys_define:nn
1141     { markdown/options }
1142     {
1143     #3 .code:n = {
1144     \@@_set_option_value:nn
1145     { #2 }
1146     { ##1 }
1147     },
1148     }
1149     }
```

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.

```
1150     \str_if_eq:VVT
1151     \l_tmpa_tl
1152     \c_@@_option_type_clist_tl
1153     {
1154     \tl_set:Nn
1155     \l_tmpa_tl
1156     { #3 }
```

```

1157     \tl_reverse:N
1158     \l_tmpa_tl
1159     \str_if_eq:enF
1160     {
1161         \tl_head:V
1162         \l_tmpa_tl
1163     }
1164     { s }
1165     {
1166         \msg_error:nnn
1167         { markdown }
1168         { malformed-name-for-clist-option }
1169         { #3 }
1170     }
1171     \tl_set:Nx
1172     \l_tmpa_tl
1173     {
1174         \tl_tail:V
1175         \l_tmpa_tl
1176     }
1177     \tl_reverse:N
1178     \l_tmpa_tl
1179     \tl_put_right:Nn
1180     \l_tmpa_tl
1181     {
1182         .code:n = {
1183             \@@_get_option_value:nN
1184             { #2 }
1185             \l_tmpa_tl
1186             \clist_set:NV
1187             \l_tmpa_clist
1188             { \l_tmpa_tl , { ##1 } }
1189             \@@_set_option_value:nV
1190             { #2 }
1191             \l_tmpa_clist
1192         }
1193     }
1194     \keys_define:nV
1195     { markdown/options }
1196     \l_tmpa_tl
1197 }
1198 }
1199 \cs_generate_variant:Nn
1200 \clist_set:Nn
1201 { NV }
1202 \cs_generate_variant:Nn
1203 \keys_define:nn

```

```

1204 { nV }
1205 \cs_generate_variant:Nn
1206 \@@_set_option_value:nn
1207 { nV }
1208 \prg_generate_conditional_variant:Nnn
1209 \str_if_eq:nn
1210 { en }
1211 { p, F }
1212 \msg_new:nnn
1213 { markdown }
1214 { malformed-name-for-clist-option }
1215 {
1216   Clist-option-name~#1~does~not~end~with~-s.
1217 }

```

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

```

1218 \str_if_eq:VVT
1219 \c_@@_top_layer_tl
1220 \c_@@_option_layer_plain_tex_tl
1221 {
1222   \@@_define_option_commands_and_keyvals:
1223 }
1224 \ExplSyntaxOff

```

### 2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a TeX document (further referred to as a *theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (\_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X package, which provides similar functionality with its `\usetheme` macro [10, Section 15.1].

A theme name is qualified if and only if it contains at least one forward slash. Theme names must be qualified to minimize naming conflicts between different themes with a similar purpose. The preferred format of a theme name is `<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` [11].

```

1225 \ExplSyntaxOn
1226 \keys_define:nn
1227   { markdown/options }
1228   {
1229     theme .code:n = {
1230       \@@_set_theme:n
1231       { #1 }
1232     },
1233     import .code:n = {
1234       \tl_set:Nn
1235       \l_tmpa_tl
1236       { #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.

```

1237     \tl_replace_all:NnV
1238     \l_tmpa_tl
1239     { / }
1240     \c_backslash_str
1241     \keys_set:nV
1242     { markdown/options/import }
1243     \l_tmpa_tl
1244   },
1245 }

```

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.

```

1246 \seq_new:N
1247 \g_@@_theme_names_seq
1248 \seq_new:N
1249 \g_@@_theme_versions_seq
1250 \tl_new:N

```

```

1251 \g_@@_current_theme_tl
1252 \tl_gset:Nn
1253 \g_@@_current_theme_tl
1254 { }
1255 \seq_gput_right:NV
1256 \g_@@_theme_names_seq
1257 \g_@@_current_theme_tl
1258 \cs_new:Npn
1259 \markdownThemeVersion
1260 { }
1261 \seq_gput_right:NV
1262 \g_@@_theme_versions_seq
1263 \g_@@_current_theme_tl
1264 \cs_new:Nn
1265 \@@_set_theme:n
1266 {

```

First, we validate the theme name.

```

1267 \str_if_in:nnF
1268 { #1 }
1269 { / }
1270 {
1271 \msg_error:nnn
1272 { markdown }
1273 { unqualified-theme-name }
1274 { #1 }
1275 }
1276 \str_if_in:nnT
1277 { #1 }
1278 { _ }
1279 {
1280 \msg_error:nnn
1281 { markdown }
1282 { underscores-in-theme-name }
1283 { #1 }
1284 }

```

Next, we extract the theme version.

```

1285 \str_if_in:nnTF
1286 { #1 }
1287 { @ }
1288 {
1289 \regex_extract_once:nnN
1290 { (.*?) @ (.*?) }
1291 { #1 }
1292 \l_tmpa_seq
1293 \seq_gpop_left:NN
1294 \l_tmpa_seq

```

```

1295         \l_tmpa_tl
1296     \seq_gpop_left:NN
1297         \l_tmpa_seq
1298         \l_tmpa_tl
1299     \tl_gset:NV
1300         \g_@@_current_theme_tl
1301         \l_tmpa_tl
1302     \seq_gpop_left:NN
1303         \l_tmpa_seq
1304         \l_tmpa_tl
1305     \cs_gset:Npe
1306         \markdownThemeVersion
1307         {
1308             \tl_use:N
1309                 \l_tmpa_tl
1310         }
1311     }
1312     {
1313         \tl_gset:Nn
1314             \g_@@_current_theme_tl
1315             { #1 }
1316         \cs_gset:Npn
1317             \markdownThemeVersion
1318             { latest }
1319     }

```

Next, we munge the theme name.

```

1320     \str_set:NV
1321         \l_tmpa_str
1322         \g_@@_current_theme_tl
1323     \str_replace_all:Nnn
1324         \l_tmpa_str
1325         { / }
1326         { _ }

```

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

```

1327     \tl_set:NV
1328         \l_tmpa_tl
1329         \g_@@_current_theme_tl
1330     \tl_put_right:Nn
1331         \g_@@_current_theme_tl
1332         { / }
1333     \seq_gput_right:NV
1334         \g_@@_theme_names_seq
1335         \g_@@_current_theme_tl
1336     \seq_gput_right:NV
1337         \g_@@_theme_versions_seq

```

```

1338     \markdownThemeVersion
1339     \@@_load_theme:VeV
1340     \l_tmpa_tl
1341     { \markdownThemeVersion }
1342     \l_tmpa_str

```

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

```

1343     \seq_gpop_right:NN
1344     \g_@@_theme_names_seq
1345     \l_tmpa_tl
1346     \seq_get_right:NN
1347     \g_@@_theme_names_seq
1348     \l_tmpa_tl
1349     \tl_gset:NV
1350     \g_@@_current_theme_tl
1351     \l_tmpa_tl
1352     \seq_gpop_right:NN
1353     \g_@@_theme_versions_seq
1354     \l_tmpa_tl
1355     \seq_get_right:NN
1356     \g_@@_theme_versions_seq
1357     \l_tmpa_tl
1358     \cs_gset:Npe
1359     \markdownThemeVersion
1360     {
1361         \tl_use:N
1362         \l_tmpa_tl
1363     }
1364 }
1365 \msg_new:nnnn
1366 { markdown }
1367 { unqualified-theme-name }
1368 { Won't~load~theme~with~unqualified~name~#1 }
1369 { Theme~names~must~contain~at~least~one~forward~slash }
1370 \msg_new:nnnn
1371 { markdown }
1372 { underscores-in-theme-name }
1373 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1374 { Theme~names~must~not~contain~underscores~in~their~names }
1375 \cs_generate_variant:Nn
1376 \tl_replace_all:Nnn
1377 { NnV }
1378 \cs_generate_variant:Nn
1379 \cs_gset:Npn
1380 { 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.

```
1381 \prop_new:N
1382   \g_@@_plain_tex_built_in_themes_prop
```

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

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

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

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

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

```


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

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

' Diagram title
title Simple Request-Response Flow

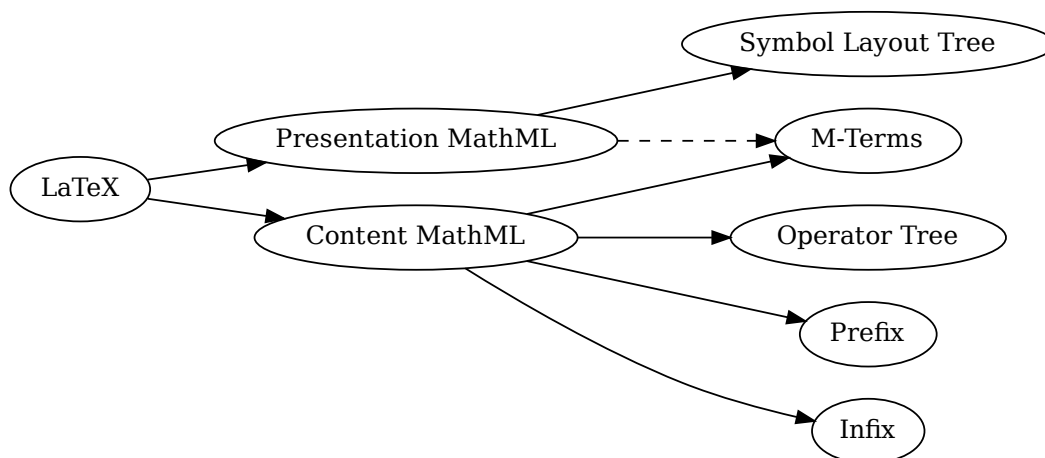
' Messages
C -> S: Send Request
note over S: Process request

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

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


```

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



**Figure 4: An example directed graph**

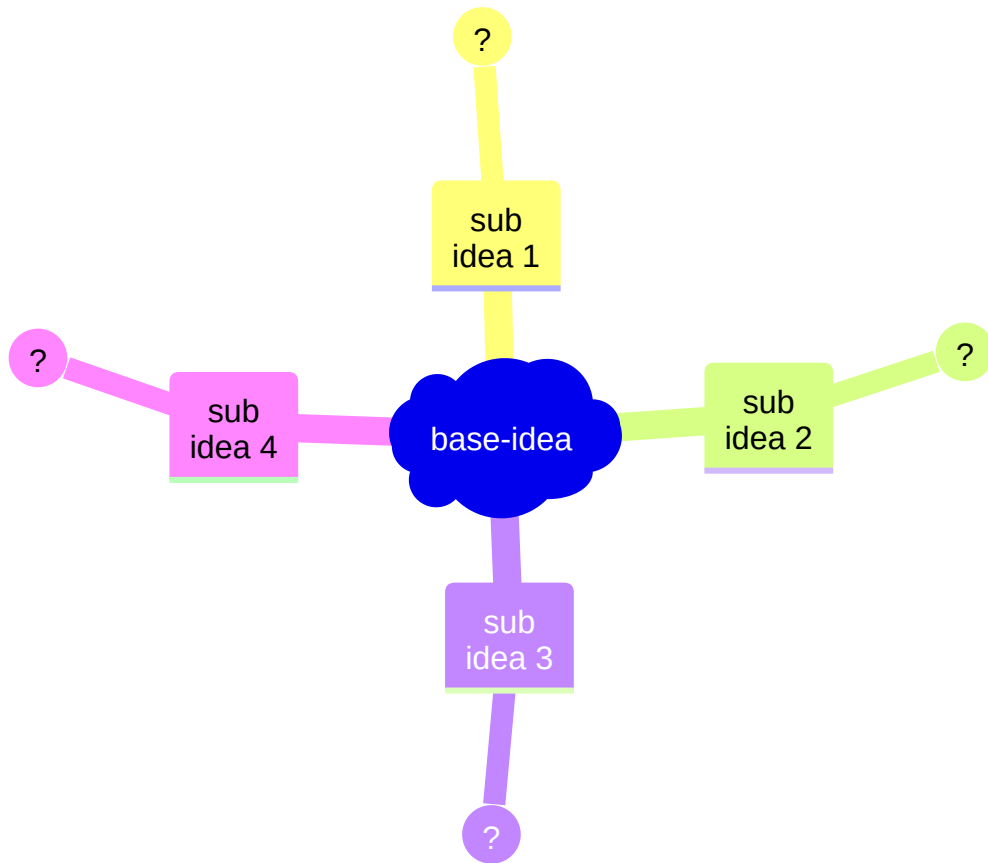
The theme requires a Unix-like operating system with GNU Diffutils, Graphviz, the npm package [@mermaid-js/mermaid-cli](#), and PlantUML installed. All these packages are already included in the Docker image [witiko/markdown](#); consult [Dockerfile](#) to see how they are installed. The theme also requires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

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

```

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

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



**Figure 5: An example mindmap**

## Simple Request-Response Flow

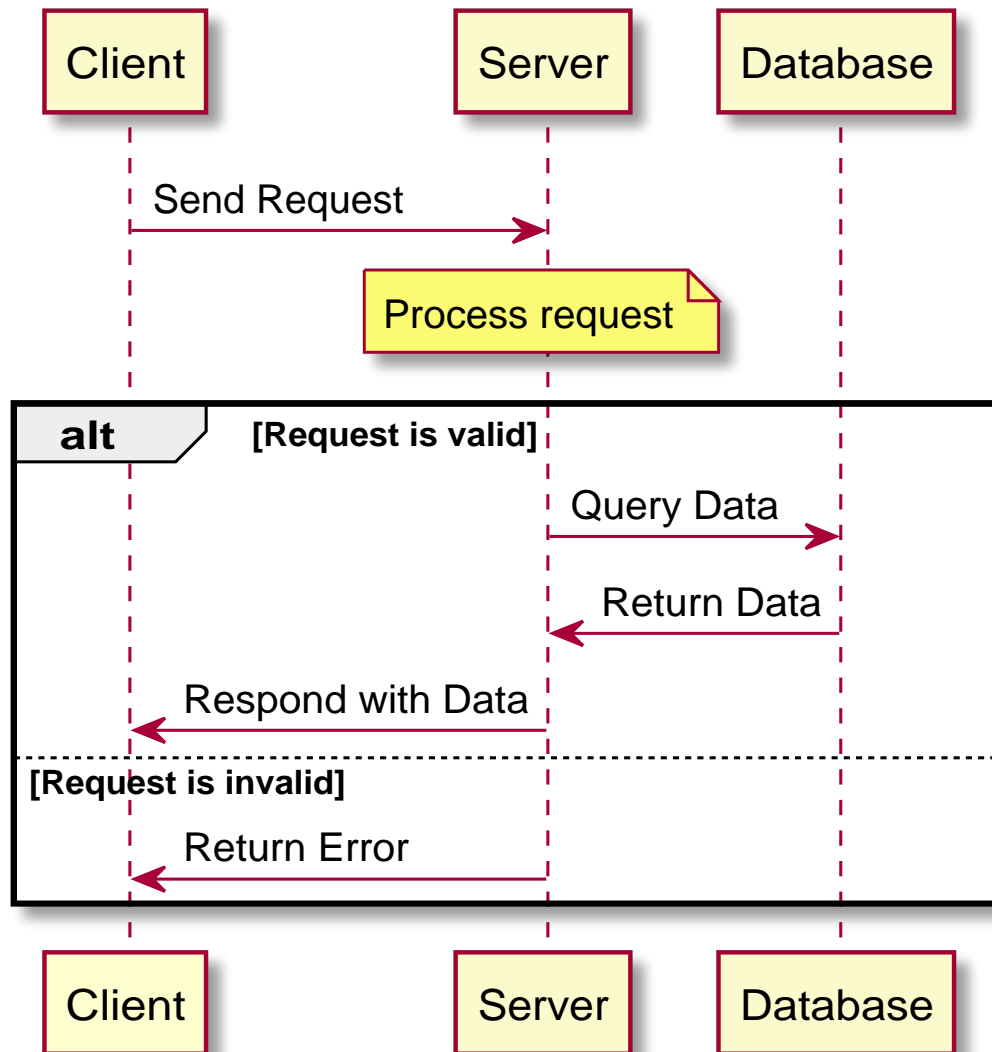


Figure 6: An example UML sequence diagram

```

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

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

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

```



# Chapter 1

## Introduction

1.1 Section  
1.1.1 Subsection  
Hello *Markdown*!

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

Table 1.1: Table

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

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

```

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

```

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

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

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

## 2.2.4 Snippets

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

```
1383 \prop_new:N
1384   \g_@@_snippets_prop
1385 \cs_new:Nn
1386   \@@_setup_snippet:nn
1387   {
1388     \tl_if_empty:nT
1389       { #1 }
1390     {
1391       \msg_error:nnn
1392         { markdown }
1393         { empty-snippet-name }
1394         { #1 }
1395     }
1396     \tl_set:NV
1397       \l_tmpa_tl
1398       \g_@@_current_theme_tl
1399     \tl_put_right:Nn
1400       \l_tmpa_tl
1401       { #1 }
1402     \@@_if_snippet_exists:nT
1403       { #1 }
1404     {
1405       \msg_warning:nnV
1406         { markdown }
1407         { redefined-snippet }
1408         \l_tmpa_tl
1409     }
1410     \keys_precompile:nnN
1411       { markdown/options }
1412       { #2 }
1413     \l_tmpb_tl
1414     \prop_gput:NVV
1415       \g_@@_snippets_prop
1416       \l_tmpa_tl
1417       \l_tmpb_tl
1418   }
1419 \cs_gset_eq:NN
1420   \markdownSetupSnippet
1421   \@@_setup_snippet:nn
1422 \msg_new:nnnn
1423   { markdown }
1424   { empty-snippet-name }
```

```

1425 { Empty~snippet~name~#1 }
1426 { Pick~a~non~empty~name~for~your~snippet }
1427 \msg_new:nnn
1428 { markdown }
1429 { redefined~snippet }
1430 { Redefined~snippet~#1 }

```

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

```

1431 \tl_new:N
1432   \l_@@_current_snippet_tl
1433 \prg_new_conditional:Nnn
1434   \@@_if_snippet_exists:n
1435   { TF, T, F }
1436   {
1437     \tl_set:NV
1438       \l_@@_current_snippet_tl
1439       \g_@@_current_theme_tl
1440     \tl_put_right:Nn
1441       \l_@@_current_snippet_tl
1442       { #1 }
1443     \prop_if_in:NVTF
1444       \g_@@_snippets_prop
1445       \l_@@_current_snippet_tl
1446       { \prg_return_true: }
1447       { \prg_return_false: }
1448   }
1449 \cs_gset_eq:NN
1450   \markdownIfSnippetExists
1451   \@@_if_snippet_exists:nTF

```

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

```

1452 \keys_define:nn
1453   { markdown/options }
1454   {
1455     snippet .code:n = {
1456       \tl_set:NV
1457         \l_tmpa_tl
1458         \g_@@_current_theme_tl
1459       \tl_put_right:Nn
1460         \l_tmpa_tl
1461         { #1 }
1462       \@@_if_snippet_exists:nTF
1463         { #1 }
1464         {
1465           \prop_get:NVN
1466             \g_@@_snippets_prop
1467             \l_tmpa_tl

```



```

1468         \l_tmpb_tl
1469         \tl_use:N
1470         \l_tmpb_tl
1471     }
1472     {
1473         \msg_error:nnV
1474         { markdown }
1475         { undefined-snippet }
1476         \l_tmpa_tl
1477     }
1478 }
1479 }
1480 \msg_new:nnn
1481 { markdown }
1482 { undefined-snippet }
1483 { Can't~invoke~undefined~snippet~#1 }
1484 \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 `jdoue/lists` theme, we could import the `jdoue/lists` theme and use the qualified name `jdoue/lists/romanNumerals` to invoke the snippet:

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:
```

```

3. tres
4. quattuor

\end{markdown}

```

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

```

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

```

```

1485 \ExplSyntaxOn
1486 \tl_new:N
1487   \l_@@_import_current_theme_tl
1488 \keys_define:nn
1489   { markdown/options/import }
1490   {

```

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

```

1491     unknown .default:n = {},
1492     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.

```

1493     \tl_set_eq:NN
1494     \l_@@_import_current_theme_tl
1495     \l_keys_key_str

```

```

1496     \tl_replace_all:NVn
1497     \l_@@_import_current_theme_tl
1498     \c_backslash_str
1499     { / }

```

Here, we import the snippets.

```

1500     \clist_map_inline:nn
1501     { #1 }
1502     {
1503         \regex_extract_once:nnNTF
1504         { ^(.*)\s+as\s+(.*)$ }
1505         { ##1 }
1506         \l_tmpa_seq
1507         {
1508             \seq_pop:NN
1509             \l_tmpa_seq
1510             \l_tmpa_tl
1511             \seq_pop:NN
1512             \l_tmpa_seq
1513             \l_tmpa_tl
1514             \seq_pop:NN
1515             \l_tmpa_seq
1516             \l_tmpb_tl
1517         }
1518         {
1519             \tl_set:Nn
1520             \l_tmpa_tl
1521             { ##1 }
1522             \tl_set:Nn
1523             \l_tmpb_tl
1524             { ##1 }
1525         }
1526         \tl_put_left:Nn
1527         \l_tmpa_tl
1528         { / }
1529         \tl_put_left:NV
1530         \l_tmpa_tl
1531         \l_@@_import_current_theme_tl
1532         \@@_setup_snippet:Vx
1533         \l_tmpb_tl
1534         { snippet = { \l_tmpa_tl } }
1535     }

```

Here, we load the theme.

```

1536     \@@_set_theme:V
1537     \l_@@_import_current_theme_tl
1538     },
1539     }

```

```

1540 \cs_generate_variant:Nn
1541   \tl_replace_all:Nnn
1542   { NVn }
1543 \cs_generate_variant:Nn
1544   \@@_set_theme:n
1545   { V }
1546 \cs_generate_variant:Nn
1547   \@@_setup_snippet:nn
1548   { 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.

```
1549 \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.

```
1550 \prop_new:N \g_@@_renderer_arities_prop
1551 \ExplSyntaxOff
```

### 2.2.5.1 Attribute Renderers

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

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

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

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

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

```

1552 \ExplSyntaxOn
1553 \cs_gset_protected:Npn
1554   \markdownRendererAttributeIdentifier
1555   {
1556     \markdownRendererAttributeIdentifierPrototype
1557   }
1558 \seq_gput_right:Nn
1559   \g_@@_renderers_seq
1560   { attributeIdentifier }
1561 \prop_gput:Nnn
1562   \g_@@_renderer_arities_prop
1563   { attributeIdentifier }
1564   { 1 }
1565 \cs_gset_protected:Npn
1566   \markdownRendererAttributeClassName
1567   {
1568     \markdownRendererAttributeClassNamePrototype
1569   }
1570 \seq_gput_right:Nn
1571   \g_@@_renderers_seq
1572   { attributeClassName }
1573 \prop_gput:Nnn
1574   \g_@@_renderer_arities_prop
1575   { attributeClassName }
1576   { 1 }
1577 \cs_gset_protected:Npn
1578   \markdownRendererAttributeKeyValue
1579   {
1580     \markdownRendererAttributeKeyValuePrototype
1581   }
1582 \seq_gput_right:Nn
1583   \g_@@_renderers_seq
1584   { attributeKeyValue }
1585 \prop_gput:Nnn
1586   \g_@@_renderer_arities_prop
1587   { attributeKeyValue }
1588   { 2 }
1589 \ExplSyntaxOff

```

### 2.2.5.2 Block Quote Renderers

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

```

1590 \ExplSyntaxOn
1591 \cs_gset_protected:Npn
1592   \markdownRendererBlockQuoteBegin
1593   {
1594     \markdownRendererBlockQuoteBeginPrototype
1595   }
1596 \seq_gput_right:Nn
1597   \g_@@_renderers_seq
1598   { blockQuoteBegin }
1599 \prop_gput:Nnn
1600   \g_@@_renderer_arities_prop
1601   { blockQuoteBegin }
1602   { 0 }
1603 \ExplSyntaxOff

```

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

```

1604 \ExplSyntaxOn
1605 \cs_gset_protected:Npn
1606   \markdownRendererBlockQuoteEnd
1607   {
1608     \markdownRendererBlockQuoteEndPrototype
1609   }
1610 \seq_gput_right:Nn
1611   \g_@@_renderers_seq
1612   { blockQuoteEnd }
1613 \prop_gput:Nnn
1614   \g_@@_renderer_arities_prop
1615   { blockQuoteEnd }
1616   { 0 }
1617 \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.

```

1618 \ExplSyntaxOn
1619 \cs_gset_protected:Npn
1620   \markdownRendererBracketedSpanAttributeContextBegin
1621   {
1622     \markdownRendererBracketedSpanAttributeContextBeginPrototype
1623   }
1624 \seq_gput_right:Nn

```

```

1625 \g_@@_renderers_seq
1626 { bracketedSpanAttributeContextBegin }
1627 \prop_gput:Nnn
1628 \g_@@_renderer_arities_prop
1629 { bracketedSpanAttributeContextBegin }
1630 { 0 }
1631 \cs_gset_protected:Npn
1632 \markdownRendererBracketedSpanAttributeContextEnd
1633 {
1634   \markdownRendererBracketedSpanAttributeContextEndPrototype
1635 }
1636 \seq_gput_right:Nn
1637 \g_@@_renderers_seq
1638 { bracketedSpanAttributeContextEnd }
1639 \prop_gput:Nnn
1640 \g_@@_renderer_arities_prop
1641 { bracketedSpanAttributeContextEnd }
1642 { 0 }
1643 \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.

```

1644 \ExplSyntaxOn
1645 \cs_gset_protected:Npn
1646 \markdownRendererUlBegin
1647 {
1648   \markdownRendererUlBeginPrototype
1649 }
1650 \seq_gput_right:Nn
1651 \g_@@_renderers_seq
1652 { ulBegin }
1653 \prop_gput:Nnn
1654 \g_@@_renderer_arities_prop
1655 { ulBegin }
1656 { 0 }
1657 \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.

```

1658 \ExplSyntaxOn
1659 \cs_gset_protected:Npn

```



```

1660 \markdownRendererUlBeginTight
1661 {
1662   \markdownRendererUlBeginTightPrototype
1663 }
1664 \seq_gput_right:Nn
1665 \g_@@_renderers_seq
1666 { ulBeginTight }
1667 \prop_gput:Nnn
1668 \g_@@_renderer_arities_prop
1669 { ulBeginTight }
1670 { 0 }
1671 \ExplSyntaxOff

```

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

```

1672 \ExplSyntaxOn
1673 \cs_gset_protected:Npn
1674   \markdownRendererUlItem
1675   {
1676     \markdownRendererUlItemPrototype
1677   }
1678 \seq_gput_right:Nn
1679 \g_@@_renderers_seq
1680 { ulItem }
1681 \prop_gput:Nnn
1682 \g_@@_renderer_arities_prop
1683 { ulItem }
1684 { 0 }
1685 \ExplSyntaxOff

```

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

```

1686 \ExplSyntaxOn
1687 \cs_gset_protected:Npn
1688   \markdownRendererUlItemEnd
1689   {
1690     \markdownRendererUlItemEndPrototype
1691   }
1692 \seq_gput_right:Nn
1693 \g_@@_renderers_seq
1694 { ulItemEnd }
1695 \prop_gput:Nnn
1696 \g_@@_renderer_arities_prop
1697 { ulItemEnd }
1698 { 0 }
1699 \ExplSyntaxOff

```

The `\markdownRendererUPEnd` 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.

```

1700 \ExplSyntaxOn
1701 \cs_gset_protected:Npn
1702   \markdownRendererUPEnd
1703   {
1704     \markdownRendererUPEndPrototype
1705   }
1706 \seq_gput_right:Nn
1707   \g_@@_renderers_seq
1708   { ulEnd }
1709 \prop_gput:Nnn
1710   \g_@@_renderer_arities_prop
1711   { ulEnd }
1712   { 0 }
1713 \ExplSyntaxOff

```

The `\markdownRendererUPEndTight` 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.

```

1714 \ExplSyntaxOn
1715 \cs_gset_protected:Npn
1716   \markdownRendererUPEndTight
1717   {
1718     \markdownRendererUPEndTightPrototype
1719   }
1720 \seq_gput_right:Nn
1721   \g_@@_renderers_seq
1722   { ulEndTight }
1723 \prop_gput:Nnn
1724   \g_@@_renderer_arities_prop
1725   { ulEndTight }
1726   { 0 }
1727 \ExplSyntaxOff

```

### 2.2.5.5 Citation Renderers

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

```

1728 \ExplSyntaxOn
1729 \cs_gset_protected:Npn
1730   \markdownRendererCite
1731   {
1732     \markdownRendererCitePrototype
1733   }
1734 \seq_gput_right:Nn
1735   \g_@@_renderers_seq
1736   { cite }
1737 \prop_gput:Nnn
1738   \g_@@_renderer_arities_prop
1739   { cite }
1740   { 1 }
1741 \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.

```

1742 \ExplSyntaxOn
1743 \cs_gset_protected:Npn
1744   \markdownRendererTextCite
1745   {
1746     \markdownRendererTextCitePrototype
1747   }
1748 \seq_gput_right:Nn
1749   \g_@@_renderers_seq
1750   { textCite }
1751 \prop_gput:Nnn
1752   \g_@@_renderer_arities_prop
1753   { textCite }
1754   { 1 }
1755 \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.

```

1756 \ExplSyntaxOn
1757 \cs_gset_protected:Npn
1758   \markdownRendererInputVerbatim
1759   {
1760     \markdownRendererInputVerbatimPrototype
1761   }
1762 \seq_gput_right:Nn

```

```

1763 \g_@@_renderers_seq
1764 { inputVerbatim }
1765 \prop_gput:Nnn
1766 \g_@@_renderer_arities_prop
1767 { inputVerbatim }
1768 { 1 }
1769 \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.

```

1770 \ExplSyntaxOn
1771 \cs_gset_protected:Npn
1772 \markdownRendererInputFencedCode
1773 {
1774   \markdownRendererInputFencedCodePrototype
1775 }
1776 \seq_gput_right:Nn
1777 \g_@@_renderers_seq
1778 { inputFencedCode }
1779 \prop_gput:Nnn
1780 \g_@@_renderer_arities_prop
1781 { inputFencedCode }
1782 { 3 }
1783 \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.

```

1784 \ExplSyntaxOn
1785 \cs_gset_protected:Npn
1786 \markdownRendererCodeSpan
1787 {
1788   \markdownRendererCodeSpanPrototype
1789 }
1790 \seq_gput_right:Nn
1791 \g_@@_renderers_seq
1792 { codeSpan }
1793 \prop_gput:Nnn
1794 \g_@@_renderer_arities_prop
1795 { codeSpan }
1796 { 1 }
1797 \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.

```
1798 \ExplSyntaxOn
1799 \cs_gset_protected:Npn
1800   \markdownRendererCodeSpanAttributeContextBegin
1801   {
1802     \markdownRendererCodeSpanAttributeContextBeginPrototype
1803   }
1804 \seq_gput_right:Nn
1805   \g_@@_renderers_seq
1806   { codeSpanAttributeContextBegin }
1807 \prop_gput:Nnn
1808   \g_@@_renderer_arities_prop
1809   { codeSpanAttributeContextBegin }
1810   { 0 }
1811 \cs_gset_protected:Npn
1812   \markdownRendererCodeSpanAttributeContextEnd
1813   {
1814     \markdownRendererCodeSpanAttributeContextEndPrototype
1815   }
1816 \seq_gput_right:Nn
1817   \g_@@_renderers_seq
1818   { codeSpanAttributeContextEnd }
1819 \prop_gput:Nnn
1820   \g_@@_renderer_arities_prop
1821   { codeSpanAttributeContextEnd }
1822   { 0 }
1823 \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.

```
1824 \ExplSyntaxOn
1825 \cs_gset_protected:Npn
1826   \markdownRendererContentBlock
1827   {
1828     \markdownRendererContentBlockPrototype
1829   }
1830 \seq_gput_right:Nn
```

```

1831 \g_@@_renderers_seq
1832 { contentBlock }
1833 \prop_gput:Nnn
1834 \g_@@_renderer_arities_prop
1835 { contentBlock }
1836 { 4 }
1837 \ExplSyntaxOff

```

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

```

1838 \ExplSyntaxOn
1839 \cs_gset_protected:Npn
1840 \markdownRendererContentBlockOnlineImage
1841 {
1842 \markdownRendererContentBlockOnlineImagePrototype
1843 }
1844 \seq_gput_right:Nn
1845 \g_@@_renderers_seq
1846 { contentBlockOnlineImage }
1847 \prop_gput:Nnn
1848 \g_@@_renderer_arities_prop
1849 { contentBlockOnlineImage }
1850 { 4 }
1851 \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  $\text{T}_\text{E}_\text{X}$  directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [5] is a good starting point.

```

1852 \ExplSyntaxOn
1853 \cs_gset_protected:Npn

```

---

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

```

1854 \markdownRendererContentBlockCode
1855 {
1856   \markdownRendererContentBlockCodePrototype
1857 }
1858 \seq_gput_right:Nn
1859 \g_@@_renderers_seq
1860 { contentBlockCode }
1861 \prop_gput:Nnn
1862 \g_@@_renderer_arities_prop
1863 { contentBlockCode }
1864 { 5 }
1865 \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.

```

1866 \ExplSyntaxOn
1867 \cs_gset_protected:Npn
1868 \markdownRendererDlBegin
1869 {
1870   \markdownRendererDlBeginPrototype
1871 }
1872 \seq_gput_right:Nn
1873 \g_@@_renderers_seq
1874 { dlBegin }
1875 \prop_gput:Nnn
1876 \g_@@_renderer_arities_prop
1877 { dlBegin }
1878 { 0 }
1879 \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.

```

1880 \ExplSyntaxOn
1881 \cs_gset_protected:Npn
1882 \markdownRendererDlBeginTight
1883 {
1884   \markdownRendererDlBeginTightPrototype
1885 }
1886 \seq_gput_right:Nn

```

```

1887 \g_@@_renderers_seq
1888 { dlBeginTight }
1889 \prop_gput:Nnn
1890 \g_@@_renderer_arities_prop
1891 { dlBeginTight }
1892 { 0 }
1893 \ExplSyntaxOff

```

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

```

1894 \ExplSyntaxOn
1895 \cs_gset_protected:Npn
1896 \markdownRendererDlItem
1897 {
1898   \markdownRendererDlItemPrototype
1899 }
1900 \seq_gput_right:Nn
1901 \g_@@_renderers_seq
1902 { dlItem }
1903 \prop_gput:Nnn
1904 \g_@@_renderer_arities_prop
1905 { dlItem }
1906 { 1 }
1907 \ExplSyntaxOff

```

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

```

1908 \ExplSyntaxOn
1909 \cs_gset_protected:Npn
1910 \markdownRendererDlItemEnd
1911 {
1912   \markdownRendererDlItemEndPrototype
1913 }
1914 \seq_gput_right:Nn
1915 \g_@@_renderers_seq
1916 { dlItemEnd }
1917 \prop_gput:Nnn
1918 \g_@@_renderer_arities_prop
1919 { dlItemEnd }
1920 { 0 }
1921 \ExplSyntaxOff

```

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

```

1922 \ExplSyntaxOn
1923 \cs_gset_protected:Npn
1924 \markdownRendererDlDefinitionBegin

```



```

1925 {
1926   \markdownRendererDlDefinitionBeginPrototype
1927 }
1928 \seq_gput_right:Nn
1929   \g_@@_renderers_seq
1930 { dlDefinitionBegin }
1931 \prop_gput:Nnn
1932   \g_@@_renderer_arities_prop
1933 { dlDefinitionBegin }
1934 { 0 }
1935 \ExplSyntaxOff

```

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

```

1936 \ExplSyntaxOn
1937 \cs_gset_protected:Npn
1938   \markdownRendererDlDefinitionEnd
1939   {
1940     \markdownRendererDlDefinitionEndPrototype
1941   }
1942 \seq_gput_right:Nn
1943   \g_@@_renderers_seq
1944   { dlDefinitionEnd }
1945 \prop_gput:Nnn
1946   \g_@@_renderer_arities_prop
1947   { dlDefinitionEnd }
1948   { 0 }
1949 \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.

```

1950 \ExplSyntaxOn
1951 \cs_gset_protected:Npn
1952   \markdownRendererDlEnd
1953   {
1954     \markdownRendererDlEndPrototype
1955   }
1956 \seq_gput_right:Nn
1957   \g_@@_renderers_seq
1958   { dlEnd }
1959 \prop_gput:Nnn
1960   \g_@@_renderer_arities_prop
1961   { dlEnd }
1962   { 0 }
1963 \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.

```

1964 \ExplSyntaxOn
1965 \cs_gset_protected:Npn
1966   \markdownRendererDlEndTight
1967   {
1968     \markdownRendererDlEndTightPrototype
1969   }
1970 \seq_gput_right:Nn
1971   \g_@@_renderers_seq
1972   { dlEndTight }
1973 \prop_gput:Nnn
1974   \g_@@_renderers_arities_prop
1975   { dlEndTight }
1976   { 0 }
1977 \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.

```

1978 \ExplSyntaxOn
1979 \cs_gset_protected:Npn
1980   \markdownRendererEllipsis
1981   {
1982     \markdownRendererEllipsisPrototype
1983   }
1984 \seq_gput_right:Nn
1985   \g_@@_renderers_seq
1986   { ellipsis }
1987 \prop_gput:Nnn
1988   \g_@@_renderers_arities_prop
1989   { ellipsis }
1990   { 0 }
1991 \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.

```

1992 \ExplSyntaxOn
1993 \cs_gset_protected:Npn

```

```

1994 \markdownRendererEmphasis
1995 {
1996   \markdownRendererEmphasisPrototype
1997 }
1998 \seq_gput_right:Nn
1999 \g_@@_renderers_seq
2000 { emphasis }
2001 \prop_gput:Nnn
2002 \g_@@_renderer_arities_prop
2003 { emphasis }
2004 { 1 }
2005 \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.

```

2006 \ExplSyntaxOn
2007 \cs_gset_protected:Npn
2008   \markdownRendererStrongEmphasis
2009   {
2010     \markdownRendererStrongEmphasisPrototype
2011   }
2012 \seq_gput_right:Nn
2013 \g_@@_renderers_seq
2014 { strongEmphasis }
2015 \prop_gput:Nnn
2016 \g_@@_renderer_arities_prop
2017 { strongEmphasis }
2018 { 1 }
2019 \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.

```

2020 \ExplSyntaxOn
2021 \cs_gset_protected:Npn
2022   \markdownRendererFencedCodeAttributeContextBegin
2023   {
2024     \markdownRendererFencedCodeAttributeContextBeginPrototype
2025   }
2026 \seq_gput_right:Nn
2027 \g_@@_renderers_seq

```

```

2028 { fencedCodeAttributeContextBegin }
2029 \prop_gput:Nnn
2030 \g_@@_renderer_arities_prop
2031 { fencedCodeAttributeContextBegin }
2032 { 0 }
2033 \cs_gset_protected:Npn
2034 \markdownRendererFencedCodeAttributeContextEnd
2035 {
2036   \markdownRendererFencedCodeAttributeContextEndPrototype
2037 }
2038 \seq_gput_right:Nn
2039 \g_@@_renderers_seq
2040 { fencedCodeAttributeContextEnd }
2041 \prop_gput:Nnn
2042 \g_@@_renderer_arities_prop
2043 { fencedCodeAttributeContextEnd }
2044 { 0 }
2045 \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.

```

2046 \ExplSyntaxOn
2047 \cs_gset_protected:Npn
2048   \markdownRendererFencedDivAttributeContextBegin
2049   {
2050     \markdownRendererFencedDivAttributeContextBeginPrototype
2051   }
2052 \seq_gput_right:Nn
2053   \g_@@_renderers_seq
2054   { fencedDivAttributeContextBegin }
2055 \prop_gput:Nnn
2056   \g_@@_renderer_arities_prop
2057   { fencedDivAttributeContextBegin }
2058   { 0 }
2059 \cs_gset_protected:Npn
2060   \markdownRendererFencedDivAttributeContextEnd
2061   {
2062     \markdownRendererFencedDivAttributeContextEndPrototype
2063   }
2064 \seq_gput_right:Nn
2065   \g_@@_renderers_seq
2066   { fencedDivAttributeContextEnd }
2067 \prop_gput:Nnn

```

```

2068 \g_@@_renderer_arities_prop
2069 { fencedDivAttributeContextEnd }
2070 { 0 }
2071 \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.

```

2072 \ExplSyntaxOn
2073 \cs_gset_protected:Npn
2074 \markdownRendererHeaderAttributeContextBegin
2075 {
2076   \markdownRendererHeaderAttributeContextBeginPrototype
2077 }
2078 \seq_gput_right:Nn
2079 \g_@@_renderers_seq
2080 { headerAttributeContextBegin }
2081 \prop_gput:Nnn
2082 \g_@@_renderer_arities_prop
2083 { headerAttributeContextBegin }
2084 { 0 }
2085 \cs_gset_protected:Npn
2086 \markdownRendererHeaderAttributeContextEnd
2087 {
2088   \markdownRendererHeaderAttributeContextEndPrototype
2089 }
2090 \seq_gput_right:Nn
2091 \g_@@_renderers_seq
2092 { headerAttributeContextEnd }
2093 \prop_gput:Nnn
2094 \g_@@_renderer_arities_prop
2095 { headerAttributeContextEnd }
2096 { 0 }
2097 \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.

```

2098 \ExplSyntaxOn
2099 \cs_gset_protected:Npn
2100 \markdownRendererHeadingOne

```

```

2101 {
2102   \markdownRendererHeadingOnePrototype
2103 }
2104 \seq_gput_right:Nn
2105   \g_@@_renderers_seq
2106   { headingOne }
2107 \prop_gput:Nnn
2108   \g_@@_renderer_arities_prop
2109   { headingOne }
2110   { 1 }
2111 \ExplSyntaxOff

```

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

```

2112 \ExplSyntaxOn
2113 \cs_gset_protected:Npn
2114   \markdownRendererHeadingTwo
2115   {
2116     \markdownRendererHeadingTwoPrototype
2117   }
2118 \seq_gput_right:Nn
2119   \g_@@_renderers_seq
2120   { headingTwo }
2121 \prop_gput:Nnn
2122   \g_@@_renderer_arities_prop
2123   { headingTwo }
2124   { 1 }
2125 \ExplSyntaxOff

```

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

```

2126 \ExplSyntaxOn
2127 \cs_gset_protected:Npn
2128   \markdownRendererHeadingThree
2129   {
2130     \markdownRendererHeadingThreePrototype
2131   }
2132 \seq_gput_right:Nn
2133   \g_@@_renderers_seq
2134   { headingThree }
2135 \prop_gput:Nnn
2136   \g_@@_renderer_arities_prop
2137   { headingThree }
2138   { 1 }
2139 \ExplSyntaxOff

```

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

```
2140 \ExplSyntaxOn
2141 \cs_gset_protected:Npn
2142   \markdownRendererHeadingFour
2143   {
2144     \markdownRendererHeadingFourPrototype
2145   }
2146 \seq_gput_right:Nn
2147   \g_@@_renderers_seq
2148   { headingFour }
2149 \prop_gput:Nnn
2150   \g_@@_renderer_arities_prop
2151   { headingFour }
2152   { 1 }
2153 \ExplSyntaxOff
```

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

```
2154 \ExplSyntaxOn
2155 \cs_gset_protected:Npn
2156   \markdownRendererHeadingFive
2157   {
2158     \markdownRendererHeadingFivePrototype
2159   }
2160 \seq_gput_right:Nn
2161   \g_@@_renderers_seq
2162   { headingFive }
2163 \prop_gput:Nnn
2164   \g_@@_renderer_arities_prop
2165   { headingFive }
2166   { 1 }
2167 \ExplSyntaxOff
```

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

```
2168 \ExplSyntaxOn
2169 \cs_gset_protected:Npn
2170   \markdownRendererHeadingSix
2171   {
2172     \markdownRendererHeadingSixPrototype
2173   }
2174 \seq_gput_right:Nn
2175   \g_@@_renderers_seq
2176   { headingSix }
2177 \prop_gput:Nnn
```

```

2178 \g_@@_renderer_arities_prop
2179 { headingSix }
2180 { 1 }
2181 \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.

```

2182 \ExplSyntaxOn
2183 \cs_gset_protected:Npn
2184 \markdownRendererInlineHtmlComment
2185 {
2186 \markdownRendererInlineHtmlCommentPrototype
2187 }
2188 \seq_gput_right:Nn
2189 \g_@@_renderers_seq
2190 { inlineHtmlComment }
2191 \prop_gput:Nnn
2192 \g_@@_renderer_arities_prop
2193 { inlineHtmlComment }
2194 { 1 }
2195 \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.

```

2196 \ExplSyntaxOn
2197 \cs_gset_protected:Npn
2198 \markdownRendererInlineHtmlTag
2199 {
2200 \markdownRendererInlineHtmlTagPrototype
2201 }
2202 \seq_gput_right:Nn
2203 \g_@@_renderers_seq
2204 { inlineHtmlTag }
2205 \prop_gput:Nnn

```



```

2206 \g_@@_renderer_arities_prop
2207 { inlineHtmlTag }
2208 { 1 }
2209 \cs_gset_protected:Npn
2210 \markdownRendererInputBlockHtmlElement
2211 {
2212   \markdownRendererInputBlockHtmlElementPrototype
2213 }
2214 \seq_gput_right:Nn
2215 \g_@@_renderers_seq
2216 { inputBlockHtmlElement }
2217 \prop_gput:Nnn
2218 \g_@@_renderer_arities_prop
2219 { inputBlockHtmlElement }
2220 { 1 }
2221 \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.

```

2222 \ExplSyntaxOn
2223 \cs_gset_protected:Npn
2224 \markdownRendererImage
2225 {
2226   \markdownRendererImagePrototype
2227 }
2228 \seq_gput_right:Nn
2229 \g_@@_renderers_seq
2230 { image }
2231 \prop_gput:Nnn
2232 \g_@@_renderer_arities_prop
2233 { image }
2234 { 4 }
2235 \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.

```

2236 \ExplSyntaxOn
2237 \cs_gset_protected:Npn

```

```

2238 \markdownRendererImageAttributeContextBegin
2239 {
2240   \markdownRendererImageAttributeContextBeginPrototype
2241 }
2242 \seq_gput_right:Nn
2243 \g_@@_renderers_seq
2244 { imageAttributeContextBegin }
2245 \prop_gput:Nnn
2246 \g_@@_renderer_arities_prop
2247 { imageAttributeContextBegin }
2248 { 0 }
2249 \cs_gset_protected:Npn
2250 \markdownRendererImageAttributeContextEnd
2251 {
2252   \markdownRendererImageAttributeContextEndPrototype
2253 }
2254 \seq_gput_right:Nn
2255 \g_@@_renderers_seq
2256 { imageAttributeContextEnd }
2257 \prop_gput:Nnn
2258 \g_@@_renderer_arities_prop
2259 { imageAttributeContextEnd }
2260 { 0 }
2261 \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.

```

2262 \ExplSyntaxOn
2263 \cs_gset_protected:Npn
2264 \markdownRendererInterblockSeparator
2265 {
2266   \markdownRendererInterblockSeparatorPrototype
2267 }
2268 \seq_gput_right:Nn
2269 \g_@@_renderers_seq
2270 { interblockSeparator }
2271 \prop_gput:Nnn
2272 \g_@@_renderer_arities_prop
2273 { interblockSeparator }
2274 { 0 }
2275 \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.

```
2276 \ExplSyntaxOn
2277 \cs_gset_protected:Npn
2278   \markdownRendererParagraphSeparator
2279   {
2280     \markdownRendererParagraphSeparatorPrototype
2281   }
2282 \seq_gput_right:Nn
2283   \g_@@_renderers_seq
2284   { paragraphSeparator }
2285 \prop_gput:Nnn
2286   \g_@@_renderer_arities_prop
2287   { paragraphSeparator }
2288   { 0 }
2289 \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.

```
2290 \ExplSyntaxOn
2291 \cs_gset_protected:Npn
2292   \markdownRendererLineBlockBegin
2293   {
2294     \markdownRendererLineBlockBeginPrototype
2295   }
2296 \seq_gput_right:Nn
2297   \g_@@_renderers_seq
2298   { lineBlockBegin }
2299 \prop_gput:Nnn
2300   \g_@@_renderer_arities_prop
2301   { lineBlockBegin }
2302   { 0 }
2303 \cs_gset_protected:Npn
2304   \markdownRendererLineBlockEnd
2305   {
2306     \markdownRendererLineBlockEndPrototype
2307   }
2308 \seq_gput_right:Nn
2309   \g_@@_renderers_seq
2310   { lineBlockEnd }
```

```

2311 \prop_gput:Nnn
2312   \g_@@_renderer_arities_prop
2313   { lineBlockEnd }
2314   { 0 }
2315 \ExplSyntaxOff

```

### 2.2.5.23 Line Break Renderers

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

```

2316 \ExplSyntaxOn
2317 \cs_gset_protected:Npn
2318   \markdownRendererSoftLineBreak
2319   {
2320     \markdownRendererSoftLineBreakPrototype
2321   }
2322 \seq_gput_right:Nn
2323   \g_@@_renderers_seq
2324   { softLineBreak }
2325 \prop_gput:Nnn
2326   \g_@@_renderer_arities_prop
2327   { softLineBreak }
2328   { 0 }
2329 \ExplSyntaxOff

```

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

```

2330 \ExplSyntaxOn
2331 \cs_gset_protected:Npn
2332   \markdownRendererHardLineBreak
2333   {
2334     \markdownRendererHardLineBreakPrototype
2335   }
2336 \seq_gput_right:Nn
2337   \g_@@_renderers_seq
2338   { hardLineBreak }
2339 \prop_gput:Nnn
2340   \g_@@_renderer_arities_prop
2341   { hardLineBreak }
2342   { 0 }
2343 \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.

```

2344 \ExplSyntaxOn
2345 \cs_gset_protected:Npn
2346   \markdownRendererLink
2347   {
2348     \markdownRendererLinkPrototype
2349   }
2350 \seq_gput_right:Nn
2351   \g_@@_renderers_seq
2352   { link }
2353 \prop_gput:Nnn
2354   \g_@@_renderer_arities_prop
2355   { link }
2356   { 4 }
2357 \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.

```

2358 \ExplSyntaxOn
2359 \cs_gset_protected:Npn
2360   \markdownRendererLinkAttributeContextBegin
2361   {
2362     \markdownRendererLinkAttributeContextBeginPrototype
2363   }
2364 \seq_gput_right:Nn
2365   \g_@@_renderers_seq
2366   { linkAttributeContextBegin }
2367 \prop_gput:Nnn
2368   \g_@@_renderer_arities_prop
2369   { linkAttributeContextBegin }
2370   { 0 }
2371 \cs_gset_protected:Npn
2372   \markdownRendererLinkAttributeContextEnd
2373   {
2374     \markdownRendererLinkAttributeContextEndPrototype
2375   }
2376 \seq_gput_right:Nn
2377   \g_@@_renderers_seq
2378   { linkAttributeContextEnd }
2379 \prop_gput:Nnn
2380   \g_@@_renderer_arities_prop
2381   { linkAttributeContextEnd }
2382   { 0 }

```

```
2383 \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.

```
2384 \ExplSyntaxOn
2385 \cs_gset_protected:Npn
2386   \markdownRendererMark
2387   {
2388     \markdownRendererMarkPrototype
2389   }
2390 \seq_gput_right:Nn
2391   \g_@@_renderers_seq
2392   { mark }
2393 \prop_gput:Nnn
2394   \g_@@_renderer_arities_prop
2395   { mark }
2396   { 1 }
2397 \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.

```
2398 \ExplSyntaxOn
2399 \cs_gset_protected:Npn
2400   \markdownRendererDocumentBegin
2401   {
2402     \markdownRendererDocumentBeginPrototype
2403   }
2404 \seq_gput_right:Nn
2405   \g_@@_renderers_seq
2406   { documentBegin }
2407 \prop_gput:Nnn
2408   \g_@@_renderer_arities_prop
2409   { documentBegin }
2410   { 0 }
2411 \cs_gset_protected:Npn
2412   \markdownRendererDocumentEnd
```

```

2413 {
2414   \markdownRendererDocumentEndPrototype
2415 }
2416 \seq_gput_right:Nn
2417   \g_@@_renderers_seq
2418   { documentEnd }
2419 \prop_gput:Nnn
2420   \g_@@_renderer_arities_prop
2421   { documentEnd }
2422   { 0 }
2423 \ExplSyntaxOff

```

### 2.2.5.28 Non-Breaking Space Renderer

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

```

2424 \ExplSyntaxOn
2425 \cs_gset_protected:Npn
2426   \markdownRendererNbsp
2427   {
2428     \markdownRendererNbspPrototype
2429   }
2430 \seq_gput_right:Nn
2431   \g_@@_renderers_seq
2432   { nbsp }
2433 \prop_gput:Nnn
2434   \g_@@_renderer_arities_prop
2435   { nbsp }
2436   { 0 }
2437 \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.

```

2438 \def\markdownRendererNote{%
2439   \markdownRendererNotePrototype}%
2440 \ExplSyntaxOn
2441 \seq_gput_right:Nn
2442   \g_@@_renderers_seq
2443   { note }
2444 \prop_gput:Nnn
2445   \g_@@_renderer_arities_prop
2446   { note }
2447   { 1 }
2448 \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.

```
2449 \ExplSyntaxOn
2450 \cs_gset_protected:Npn
2451   \markdownRendererOlBegin
2452   {
2453     \markdownRendererOlBeginPrototype
2454   }
2455 \seq_gput_right:Nn
2456   \g_@@_renderers_seq
2457   { olBegin }
2458 \prop_gput:Nnn
2459   \g_@@_renderer_arities_prop
2460   { olBegin }
2461   { 0 }
2462 \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.

```
2463 \ExplSyntaxOn
2464 \cs_gset_protected:Npn
2465   \markdownRendererOlBeginTight
2466   {
2467     \markdownRendererOlBeginTightPrototype
2468   }
2469 \seq_gput_right:Nn
2470   \g_@@_renderers_seq
2471   { olBeginTight }
2472 \prop_gput:Nnn
2473   \g_@@_renderer_arities_prop
2474   { olBeginTight }
2475   { 0 }
2476 \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`).



```

2477 \ExplSyntaxOn
2478 \cs_gset_protected:Npn
2479   \markdownRendererFancyOlBegin
2480   {
2481     \markdownRendererFancyOlBeginPrototype
2482   }
2483 \seq_gput_right:Nn
2484   \g_@@_renderers_seq
2485   { fancyOlBegin }
2486 \prop_gput:Nnn
2487   \g_@@_renderer_arities_prop
2488   { fancyOlBegin }
2489   { 2 }
2490 \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.

```

2491 \ExplSyntaxOn
2492 \cs_gset_protected:Npn
2493   \markdownRendererFancyOlBeginTight
2494   {
2495     \markdownRendererFancyOlBeginTightPrototype
2496   }
2497 \seq_gput_right:Nn
2498   \g_@@_renderers_seq
2499   { fancyOlBeginTight }
2500 \prop_gput:Nnn
2501   \g_@@_renderer_arities_prop
2502   { fancyOlBeginTight }
2503   { 2 }
2504 \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.

```

2505 \ExplSyntaxOn
2506 \cs_gset_protected:Npn
2507   \markdownRendererOlItem
2508   {
2509     \markdownRendererOlItemPrototype
2510   }
2511 \seq_gput_right:Nn

```

```

2512 \g_@@_renderers_seq
2513 { olItem }
2514 \prop_gput:Nnn
2515 \g_@@_renderer_arities_prop
2516 { olItem }
2517 { 0 }
2518 \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.

```

2519 \ExplSyntaxOn
2520 \cs_gset_protected:Npn
2521 \markdownRendererOlItemEnd
2522 {
2523 \markdownRendererOlItemEndPrototype
2524 }
2525 \seq_gput_right:Nn
2526 \g_@@_renderers_seq
2527 { olItemEnd }
2528 \prop_gput:Nnn
2529 \g_@@_renderer_arities_prop
2530 { olItemEnd }
2531 { 0 }
2532 \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.

```

2533 \ExplSyntaxOn
2534 \cs_gset_protected:Npn
2535 \markdownRendererOlItemWithNumber
2536 {
2537 \markdownRendererOlItemWithNumberPrototype
2538 }
2539 \seq_gput_right:Nn
2540 \g_@@_renderers_seq
2541 { olItemWithNumber }
2542 \prop_gput:Nnn
2543 \g_@@_renderer_arities_prop
2544 { olItemWithNumber }
2545 { 1 }
2546 \ExplSyntaxOff

```

The `\markdownRendererFancy01Item` macro represents an item in a fancy ordered list. This macro will only be produced, when the `startNumber` option is disabled and the `fancyLists` option is enabled. The macro receives no arguments.

```

2547 \ExplSyntaxOn
2548 \cs_gset_protected:Npn
2549   \markdownRendererFancy01Item
2550   {
2551     \markdownRendererFancy01ItemPrototype
2552   }
2553 \seq_gput_right:Nn
2554   \g_@@_renderers_seq
2555   { fancy01Item }
2556 \prop_gput:Nnn
2557   \g_@@_renderer_arities_prop
2558   { fancy01Item }
2559   { 0 }
2560 \ExplSyntaxOff

```

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

```

2561 \ExplSyntaxOn
2562 \cs_gset_protected:Npn
2563   \markdownRendererFancy01ItemEnd
2564   {
2565     \markdownRendererFancy01ItemEndPrototype
2566   }
2567 \seq_gput_right:Nn
2568   \g_@@_renderers_seq
2569   { fancy01ItemEnd }
2570 \prop_gput:Nnn
2571   \g_@@_renderer_arities_prop
2572   { fancy01ItemEnd }
2573   { 0 }
2574 \ExplSyntaxOff

```

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

```

2575 \ExplSyntaxOn
2576 \cs_gset_protected:Npn
2577   \markdownRendererFancy01ItemWithNumber
2578   {
2579     \markdownRendererFancy01ItemWithNumberPrototype
2580   }

```

```

2581 \seq_gput_right:Nn
2582   \g_@@_renderers_seq
2583   { fancyO1ItemWithNumber }
2584 \prop_gput:Nnn
2585   \g_@@_renderer_arities_prop
2586   { fancyO1ItemWithNumber }
2587   { 1 }
2588 \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.

```

2589 \ExplSyntaxOn
2590 \cs_gset_protected:Npn
2591   \markdownRendererO1End
2592   {
2593     \markdownRendererO1EndPrototype
2594   }
2595 \seq_gput_right:Nn
2596   \g_@@_renderers_seq
2597   { olEnd }
2598 \prop_gput:Nnn
2599   \g_@@_renderer_arities_prop
2600   { olEnd }
2601   { 0 }
2602 \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.

```

2603 \ExplSyntaxOn
2604 \cs_gset_protected:Npn
2605   \markdownRendererO1EndTight
2606   {
2607     \markdownRendererO1EndTightPrototype
2608   }
2609 \seq_gput_right:Nn
2610   \g_@@_renderers_seq
2611   { olEndTight }
2612 \prop_gput:Nnn
2613   \g_@@_renderer_arities_prop
2614   { olEndTight }
2615   { 0 }
2616 \ExplSyntaxOff

```

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

```

2617 \ExplSyntaxOn
2618 \cs_gset_protected:Npn
2619   \markdownRendererFancyO1End
2620   {
2621     \markdownRendererFancyO1EndPrototype
2622   }
2623 \seq_gput_right:Nn
2624   \g_@@_renderers_seq
2625   { fancyO1End }
2626 \prop_gput:Nnn
2627   \g_@@_renderer_arities_prop
2628   { fancyO1End }
2629   { 0 }
2630 \ExplSyntaxOff

```

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

```

2631 \ExplSyntaxOn
2632 \cs_gset_protected:Npn
2633   \markdownRendererFancyO1EndTight
2634   {
2635     \markdownRendererFancyO1EndTightPrototype
2636   }
2637 \seq_gput_right:Nn
2638   \g_@@_renderers_seq
2639   { fancyO1EndTight }
2640 \prop_gput:Nnn
2641   \g_@@_renderer_arities_prop
2642   { fancyO1EndTight }
2643   { 0 }
2644 \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.

```

2645 \ExplSyntaxOn

```

```

2646 \cs_gset_protected:Npn
2647   \markdownRendererInputRawInline
2648   {
2649     \markdownRendererInputRawInlinePrototype
2650   }
2651 \seq_gput_right:Nn
2652   \g_@@_renderers_seq
2653   { inputRawInline }
2654 \prop_gput:Nnn
2655   \g_@@_renderer_arities_prop
2656   { inputRawInline }
2657   { 2 }
2658 \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.

```

2659 \ExplSyntaxOn
2660 \cs_gset_protected:Npn
2661   \markdownRendererInputRawBlock
2662   {
2663     \markdownRendererInputRawBlockPrototype
2664   }
2665 \seq_gput_right:Nn
2666   \g_@@_renderers_seq
2667   { inputRawBlock }
2668 \prop_gput:Nnn
2669   \g_@@_renderer_arities_prop
2670   { inputRawBlock }
2671   { 2 }
2672 \ExplSyntaxOff

```

### 2.2.5.32 Section Renderers

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

```

2673 \ExplSyntaxOn
2674 \cs_gset_protected:Npn
2675   \markdownRendererSectionBegin
2676   {
2677     \markdownRendererSectionBeginPrototype
2678   }
2679 \seq_gput_right:Nn
2680   \g_@@_renderers_seq
2681   { sectionBegin }
2682 \prop_gput:Nnn

```

```

2683 \g_@@_renderer_arities_prop
2684 { sectionBegin }
2685 { 0 }
2686 \cs_gset_protected:Npn
2687 \markdownRendererSectionEnd
2688 {
2689   \markdownRendererSectionEndPrototype
2690 }
2691 \seq_gput_right:Nn
2692 \g_@@_renderers_seq
2693 { sectionEnd }
2694 \prop_gput:Nnn
2695 \g_@@_renderer_arities_prop
2696 { sectionEnd }
2697 { 0 }
2698 \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.

```

2699 \ExplSyntaxOn
2700 \cs_gset_protected:Npn
2701 \markdownRendererReplacementCharacter
2702 {
2703   \markdownRendererReplacementCharacterPrototype
2704 }
2705 \seq_gput_right:Nn
2706 \g_@@_renderers_seq
2707 { replacementCharacter }
2708 \prop_gput:Nnn
2709 \g_@@_renderer_arities_prop
2710 { replacementCharacter }
2711 { 0 }
2712 \ExplSyntaxOff

```

### 2.2.5.34 Special Character Renderers

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

```

2713 \ExplSyntaxOn
2714 \cs_gset_protected:Npn
2715 \markdownRendererLeftBrace
2716 {
2717   \markdownRendererLeftBracePrototype
2718 }

```

```

2719 \seq_gput_right:Nn
2720   \g_@@_renderers_seq
2721   { leftBrace }
2722 \prop_gput:Nnn
2723   \g_@@_renderer_arities_prop
2724   { leftBrace }
2725   { 0 }
2726 \cs_gset_protected:Npn
2727   \markdownRendererRightBrace
2728   {
2729     \markdownRendererRightBracePrototype
2730   }
2731 \seq_gput_right:Nn
2732   \g_@@_renderers_seq
2733   { rightBrace }
2734 \prop_gput:Nnn
2735   \g_@@_renderer_arities_prop
2736   { rightBrace }
2737   { 0 }
2738 \cs_gset_protected:Npn
2739   \markdownRendererDollarSign
2740   {
2741     \markdownRendererDollarSignPrototype
2742   }
2743 \seq_gput_right:Nn
2744   \g_@@_renderers_seq
2745   { dollarSign }
2746 \prop_gput:Nnn
2747   \g_@@_renderer_arities_prop
2748   { dollarSign }
2749   { 0 }
2750 \cs_gset_protected:Npn
2751   \markdownRendererPercentSign
2752   {
2753     \markdownRendererPercentSignPrototype
2754   }
2755 \seq_gput_right:Nn
2756   \g_@@_renderers_seq
2757   { percentSign }
2758 \prop_gput:Nnn
2759   \g_@@_renderer_arities_prop
2760   { percentSign }
2761   { 0 }
2762 \cs_gset_protected:Npn
2763   \markdownRendererAmpersand
2764   {
2765     \markdownRendererAmpersandPrototype

```



```

2766 }
2767 \seq_gput_right:Nn
2768 \g_@@_renderers_seq
2769 { ampersand }
2770 \prop_gput:Nnn
2771 \g_@@_renderer_arities_prop
2772 { ampersand }
2773 { 0 }
2774 \cs_gset_protected:Npn
2775 \markdownRendererUnderscore
2776 {
2777   \markdownRendererUnderscorePrototype
2778 }
2779 \seq_gput_right:Nn
2780 \g_@@_renderers_seq
2781 { underscore }
2782 \prop_gput:Nnn
2783 \g_@@_renderer_arities_prop
2784 { underscore }
2785 { 0 }
2786 \cs_gset_protected:Npn
2787 \markdownRendererHash
2788 {
2789   \markdownRendererHashPrototype
2790 }
2791 \seq_gput_right:Nn
2792 \g_@@_renderers_seq
2793 { hash }
2794 \prop_gput:Nnn
2795 \g_@@_renderer_arities_prop
2796 { hash }
2797 { 0 }
2798 \cs_gset_protected:Npn
2799 \markdownRendererCircumflex
2800 {
2801   \markdownRendererCircumflexPrototype
2802 }
2803 \seq_gput_right:Nn
2804 \g_@@_renderers_seq
2805 { circumflex }
2806 \prop_gput:Nnn
2807 \g_@@_renderer_arities_prop
2808 { circumflex }
2809 { 0 }
2810 \cs_gset_protected:Npn
2811 \markdownRendererBackslash
2812 {

```

```

2813     \markdownRendererBackslashPrototype
2814   }
2815 \seq_gput_right:Nn
2816   \g_@@_renderers_seq
2817   { backslash }
2818 \prop_gput:Nnn
2819   \g_@@_renderer_arities_prop
2820   { backslash }
2821   { 0 }
2822 \cs_gset_protected:Npn
2823   \markdownRendererTilde
2824   {
2825     \markdownRendererTildePrototype
2826   }
2827 \seq_gput_right:Nn
2828   \g_@@_renderers_seq
2829   { tilde }
2830 \prop_gput:Nnn
2831   \g_@@_renderer_arities_prop
2832   { tilde }
2833   { 0 }
2834 \cs_gset_protected:Npn
2835   \markdownRendererPipe
2836   {
2837     \markdownRendererPipePrototype
2838   }
2839 \seq_gput_right:Nn
2840   \g_@@_renderers_seq
2841   { pipe }
2842 \prop_gput:Nnn
2843   \g_@@_renderer_arities_prop
2844   { pipe }
2845   { 0 }
2846 \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.

```

2847 \ExplSyntaxOn
2848 \cs_gset_protected:Npn
2849   \markdownRendererStrikeThrough
2850   {
2851     \markdownRendererStrikeThroughPrototype
2852   }

```

```

2853 \seq_gput_right:Nn
2854   \g_@@_renderers_seq
2855   { strikeThrough }
2856 \prop_gput:Nnn
2857   \g_@@_renderer_arities_prop
2858   { strikeThrough }
2859   { 1 }
2860 \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.

```

2861 \ExplSyntaxOn
2862 \cs_gset_protected:Npn
2863   \markdownRendererSubscript
2864   {
2865     \markdownRendererSubscriptPrototype
2866   }
2867 \seq_gput_right:Nn
2868   \g_@@_renderers_seq
2869   { subscript }
2870 \prop_gput:Nnn
2871   \g_@@_renderer_arities_prop
2872   { subscript }
2873   { 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.

```

2874 \cs_gset_protected:Npn
2875   \markdownRendererSuperscript
2876   {
2877     \markdownRendererSuperscriptPrototype
2878   }
2879 \seq_gput_right:Nn
2880   \g_@@_renderers_seq
2881   { superscript }
2882 \prop_gput:Nnn
2883   \g_@@_renderer_arities_prop
2884   { superscript }
2885   { 1 }
2886 \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.

```
2887 \ExplSyntaxOn
2888 \cs_gset_protected:Npn
2889   \markdownRendererTableAttributeContextBegin
2890   {
2891     \markdownRendererTableAttributeContextBeginPrototype
2892   }
2893 \seq_gput_right:Nn
2894   \g_@@_renderers_seq
2895   { tableAttributeContextBegin }
2896 \prop_gput:Nnn
2897   \g_@@_renderer_arities_prop
2898   { tableAttributeContextBegin }
2899   { 0 }
2900 \cs_gset_protected:Npn
2901   \markdownRendererTableAttributeContextEnd
2902   {
2903     \markdownRendererTableAttributeContextEndPrototype
2904   }
2905 \seq_gput_right:Nn
2906   \g_@@_renderers_seq
2907   { tableAttributeContextEnd }
2908 \prop_gput:Nnn
2909   \g_@@_renderer_arities_prop
2910   { tableAttributeContextEnd }
2911   { 0 }
2912 \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.

```

2913 \ExplSyntaxOn
2914 \cs_gset_protected:Npn
2915   \markdownRendererTable
2916   {
2917     \markdownRendererTablePrototype
2918   }
2919 \seq_gput_right:Nn
2920   \g_@@_renderers_seq
2921   { table }
2922 \prop_gput:Nnn
2923   \g_@@_renderer_arities_prop
2924   { table }
2925   { 3 }
2926 \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.

```

2927 \ExplSyntaxOn
2928 \cs_gset_protected:Npn
2929   \markdownRendererInlineMath
2930   {
2931     \markdownRendererInlineMathPrototype
2932   }
2933 \seq_gput_right:Nn
2934   \g_@@_renderers_seq
2935   { inlineMath }
2936 \prop_gput:Nnn
2937   \g_@@_renderer_arities_prop
2938   { inlineMath }
2939   { 1 }
2940 \cs_gset_protected:Npn
2941   \markdownRendererDisplayMath
2942   {
2943     \markdownRendererDisplayMathPrototype
2944   }
2945 \seq_gput_right:Nn
2946   \g_@@_renderers_seq
2947   { displayMath }
2948 \prop_gput:Nnn
2949   \g_@@_renderer_arities_prop

```

```

2950 { displayMath }
2951 { 1 }
2952 \ExplSyntaxOff

```

### 2.2.5.41 Thematic Break Renderer

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

```

2953 \ExplSyntaxOn
2954 \cs_gset_protected:Npn
2955   \markdownRendererThematicBreak
2956   {
2957     \markdownRendererThematicBreakPrototype
2958   }
2959 \seq_gput_right:Nn
2960   \g_@@_renderers_seq
2961   { thematicBreak }
2962 \prop_gput:Nnn
2963   \g_@@_renderer_arities_prop
2964   { thematicBreak }
2965   { 0 }
2966 \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.

```

2967 \ExplSyntaxOn
2968 \cs_gset_protected:Npn
2969   \markdownRendererTickedBox
2970   {
2971     \markdownRendererTickedBoxPrototype
2972   }
2973 \seq_gput_right:Nn
2974   \g_@@_renderers_seq
2975   { tickedBox }
2976 \prop_gput:Nnn
2977   \g_@@_renderer_arities_prop
2978   { tickedBox }
2979   { 0 }
2980 \cs_gset_protected:Npn
2981   \markdownRendererHalfTickedBox
2982   {

```

```

2983     \markdownRendererHalfTickedBoxPrototype
2984   }
2985 \seq_gput_right:Nn
2986   \g_@@_renderers_seq
2987   { halfTickedBox }
2988 \prop_gput:Nnn
2989   \g_@@_renderer_arities_prop
2990   { halfTickedBox }
2991   { 0 }
2992 \cs_gset_protected:Npn
2993   \markdownRendererUntickedBox
2994   {
2995     \markdownRendererUntickedBoxPrototype
2996   }
2997 \seq_gput_right:Nn
2998   \g_@@_renderers_seq
2999   { untickedBox }
3000 \prop_gput:Nnn
3001   \g_@@_renderer_arities_prop
3002   { untickedBox }
3003   { 0 }
3004 \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.

```

3005 \ExplSyntaxOn
3006 \cs_gset_protected:Npn
3007   \markdownRendererWarning
3008   {
3009     \markdownRendererWarningPrototype
3010   }
3011 \cs_gset_protected:Npn
3012   \markdownRendererError
3013   {

```

```

3014     \markdownRendererErrorPrototype
3015   }
3016 \seq_gput_right:Nn
3017   \g_@@_renderers_seq
3018   { warning }
3019 \prop_gput:Nnn
3020   \g_@@_renderer_arities_prop
3021   { warning }
3022   { 4 }
3023 \seq_gput_right:Nn
3024   \g_@@_renderers_seq
3025   { error }
3026 \prop_gput:Nnn
3027   \g_@@_renderer_arities_prop
3028   { error }
3029   { 4 }
3030 \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.

```

3031 \ExplSyntaxOn
3032 \cs_gset_protected:Npn
3033   \markdownRendererJekyllDataBegin
3034   {
3035     \markdownRendererJekyllDataBeginPrototype
3036   }
3037 \seq_gput_right:Nn
3038   \g_@@_renderers_seq
3039   { jekyllDataBegin }
3040 \prop_gput:Nnn
3041   \g_@@_renderer_arities_prop
3042   { jekyllDataBegin }
3043   { 0 }
3044 \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.

```

3045 \ExplSyntaxOn
3046 \cs_gset_protected:Npn
3047   \markdownRendererJekyllDataEnd
3048   {
3049     \markdownRendererJekyllDataEndPrototype
3050   }

```



```

3051 \seq_gput_right:Nn
3052   \g_@@_renderers_seq
3053   { jekyllDataEnd }
3054 \prop_gput:Nnn
3055   \g_@@_renderer_arities_prop
3056   { jekyllDataEnd }
3057   { 0 }
3058 \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.

```

3059 \ExplSyntaxOn
3060 \cs_gset_protected:Npn
3061   \markdownRendererJekyllDataMappingBegin
3062   {
3063     \markdownRendererJekyllDataMappingBeginPrototype
3064   }
3065 \seq_gput_right:Nn
3066   \g_@@_renderers_seq
3067   { jekyllDataMappingBegin }
3068 \prop_gput:Nnn
3069   \g_@@_renderer_arities_prop
3070   { jekyllDataMappingBegin }
3071   { 2 }
3072 \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.

```

3073 \ExplSyntaxOn
3074 \cs_gset_protected:Npn
3075   \markdownRendererJekyllDataMappingEnd
3076   {
3077     \markdownRendererJekyllDataMappingEndPrototype
3078   }
3079 \seq_gput_right:Nn
3080   \g_@@_renderers_seq
3081   { jekyllDataMappingEnd }
3082 \prop_gput:Nnn
3083   \g_@@_renderer_arities_prop
3084   { jekyllDataMappingEnd }
3085   { 0 }
3086 \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.

```

3087 \ExplSyntaxOn
3088 \cs_gset_protected:Npn
3089   \markdownRendererJekyllDataSequenceBegin
3090   {
3091     \markdownRendererJekyllDataSequenceBeginPrototype
3092   }
3093 \seq_gput_right:Nn
3094   \g_@@_renderers_seq
3095   { jekyllDataSequenceBegin }
3096 \prop_gput:Nnn
3097   \g_@@_renderer_arities_prop
3098   { jekyllDataSequenceBegin }
3099   { 2 }
3100 \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.

```

3101 \ExplSyntaxOn
3102 \cs_gset_protected:Npn
3103   \markdownRendererJekyllDataSequenceEnd
3104   {
3105     \markdownRendererJekyllDataSequenceEndPrototype
3106   }
3107 \seq_gput_right:Nn
3108   \g_@@_renderers_seq
3109   { jekyllDataSequenceEnd }
3110 \prop_gput:Nnn
3111   \g_@@_renderer_arities_prop
3112   { jekyllDataSequenceEnd }
3113   { 0 }
3114 \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.

```

3115 \ExplSyntaxOn
3116 \cs_gset_protected:Npn

```

```

3117 \markdownRendererJekyllDataBoolean
3118 {
3119   \markdownRendererJekyllDataBooleanPrototype
3120 }
3121 \seq_gput_right:Nn
3122 \g_@@_renderers_seq
3123 { jekyllDataBoolean }
3124 \prop_gput:Nnn
3125 \g_@@_renderer_arities_prop
3126 { jekyllDataBoolean }
3127 { 2 }
3128 \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.

```

3129 \ExplSyntaxOn
3130 \cs_gset_protected:Npn
3131 \markdownRendererJekyllDataNumber
3132 {
3133   \markdownRendererJekyllDataNumberPrototype
3134 }
3135 \seq_gput_right:Nn
3136 \g_@@_renderers_seq
3137 { jekyllDataNumber }
3138 \prop_gput:Nnn
3139 \g_@@_renderer_arities_prop
3140 { jekyllDataNumber }
3141 { 2 }
3142 \ExplSyntaxOff

```

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

For each string scalar value, both macros are produced. Whereas `\markdownRendererJekyllDataTypographicString` receives the scalar value after all markdown markup and special  $\TeX$  characters in the string have been replaced by  $\TeX$  macros, `\markdownRendererJekyllDataProgrammaticString` receives the raw scalar value. Therefore, whereas the `\markdownRendererJekyllDataTypographicString` macro is more appropriate for texts that are supposed to be typeset with  $\TeX$ , such as document titles, author names, or exam questions, the

`\markdownRendererJekyllDataProgrammaticString` macro is more appropriate for identifiers and other programmatic text that won't be typeset by T<sub>E</sub>X.

```

3143 \ExplSyntaxOn
3144 \cs_gset_protected:Npn
3145   \markdownRendererJekyllDataTypographicString
3146   {
3147     \markdownRendererJekyllDataTypographicStringPrototype
3148   }
3149 \cs_gset_protected:Npn
3150   \markdownRendererJekyllDataProgrammaticString
3151   {
3152     \markdownRendererJekyllDataProgrammaticStringPrototype
3153   }
3154 \seq_gput_right:Nn
3155   \g_@@_renderers_seq
3156   { jekyllDataTypographicString }
3157 \prop_gput:Nnn
3158   \g_@@_renderer_arities_prop
3159   { jekyllDataTypographicString }
3160   { 2 }
3161 \seq_gput_right:Nn
3162   \g_@@_renderers_seq
3163   { jekyllDataProgrammaticString }
3164 \prop_gput:Nnn
3165   \g_@@_renderer_arities_prop
3166   { jekyllDataProgrammaticString }
3167   { 2 }
3168 \ExplSyntaxOff

```

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

```

3169 \ExplSyntaxOn
3170 \cs_gset:Npn
3171   \markdownRendererJekyllDataTypographicString
3172   {
3173     \cs_if_exist:NTF
3174       \markdownRendererJekyllDataString
3175       {
3176         \@@_if_option:nTF
3177           { experimental }
3178           {
3179             \markdownError
3180             {
3181               The~jekyllDataString~renderer~has~been~deprecated,~

```

```

3182         to-be-removed-in-Markdown-4.0.0
3183     }
3184 }
3185 {
3186     \markdownWarning
3187     {
3188         The~jekyllDataString~renderer~has~been~deprecated,~
3189         to-be-removed-in-Markdown-4.0.0
3190     }
3191     \markdownRendererJekyllDataString
3192 }
3193 }
3194 {
3195     \cs_if_exist:NTF
3196     \markdownRendererJekyllDataStringPrototype
3197     {
3198         \@@_if_option:nTF
3199         { experimental }
3200         {
3201             \markdownError
3202             {
3203                 The~jekyllDataString~renderer~prototype~
3204                 has~been~deprecated,~
3205                 to-be-removed-in-Markdown-4.0.0
3206             }
3207         }
3208         {
3209             \markdownWarning
3210             {
3211                 The~jekyllDataString~renderer~prototype~
3212                 has~been~deprecated,~
3213                 to-be-removed-in-Markdown-4.0.0
3214             }
3215             \markdownRendererJekyllDataStringPrototype
3216         }
3217     }
3218     {
3219         \markdownRendererJekyllDataTypographicStringPrototype
3220     }
3221 }
3222 }
3223 \seq_gput_right:Nn
3224 \g_@@_renderers_seq
3225 { jekyllDataString }
3226 \prop_gput:Nnn
3227 \g_@@_renderer_arities_prop
3228 { jekyllDataString }

```

```

3229 { 2 }
3230 \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.

```

3231 \ExplSyntaxOn
3232 \cs_gset_protected:Npn
3233   \markdownRendererJekyllDataEmpty
3234   {
3235     \markdownRendererJekyllDataEmptyPrototype
3236   }
3237 \seq_gput_right:Nn
3238   \g_@@_renderers_seq
3239   { jekyllDataEmpty }
3240 \prop_gput:Nnn
3241   \g_@@_renderer_arities_prop
3242   { jekyllDataEmpty }
3243   { 1 }
3244 \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.

```

3245 \ExplSyntaxOn
3246 \cs_new:Nn \@@_define_renderers:
3247   {
3248     \seq_map_inline:Nn
3249       \g_@@_renderers_seq
3250       {
3251         \@@_define_renderer:n
3252         { ##1 }
3253       }
3254   }
3255 \cs_new:Nn \@@_define_renderer:n
3256   {

```

```

3257 \@@_renderer_tl_to_csname:nN
3258 { #1 }
3259 \l_tmpa_tl
3260 \prop_get:NnN
3261 \g_@@_renderer_arities_prop
3262 { #1 }
3263 \l_tmpb_tl
3264 \@@_define_renderer:ncV
3265 { #1 }
3266 { \l_tmpa_tl }
3267 \l_tmpb_tl
3268 }
3269 \cs_new:Nn \@@_renderer_tl_to_csname:nN
3270 {
3271 \tl_set:Nn
3272 \l_tmpa_tl
3273 { \str_uppercase:n { #1 } }
3274 \tl_set:Nx
3275 #2
3276 {
3277 markdownRenderer
3278 \tl_head:f { \l_tmpa_tl }
3279 \tl_tail:n { #1 }
3280 }
3281 }
3282 \tl_new:N
3283 \l_@@_renderer_definition_tl
3284 \bool_new:N
3285 \g_@@_appending_renderer_bool
3286 \bool_new:N
3287 \g_@@_unprotected_renderer_bool
3288 \cs_new:Nn \@@_define_renderer:nNn
3289 {
3290 \keys_define:nn
3291 { markdown/options/renderers }
3292 {
3293 #1 .code:n = {
3294 \tl_set:Nn
3295 \l_@@_renderer_definition_tl
3296 { ##1 }
3297 \regex_replace_all:nnN
3298 { \cP\#0 }
3299 { #1 }
3300 \l_@@_renderer_definition_tl
3301 \bool_if:NT
3302 \g_@@_appending_renderer_bool
3303 {

```

```

3304         \@@_tl_set_from_cs:NNn
3305         \l_tmpa_tl
3306         #2
3307         { #3 }
3308         \tl_put_left:NV
3309         \l_@@_renderer_definition_tl
3310         \l_tmpa_tl
3311     }
3312 \bool_if:NTF
3313 \g_@@_unprotected_renderer_bool
3314 {
3315     \tl_set:Nn
3316     \l_tmpa_tl
3317     { \cs_set:Npn }
3318 }
3319 {
3320     \tl_set:Nn
3321     \l_tmpa_tl
3322     { \cs_set_protected:Npn }
3323 }
3324 \exp_last_unbraced:NNV
3325 \cs_generate_from_arg_count:NNnV
3326 #2
3327 \l_tmpa_tl
3328 { #3 }
3329 \l_@@_renderer_definition_tl
3330 },
3331 }

```

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.

```

3332 \str_if_eq:nnT
3333 { #1 }
3334 { jekyllDataString }
3335 {
3336     \cs_undefine:N
3337     #2
3338 }
3339 }

```

We define the function `\@@_tl_set_from_cs:NNn` [12]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```

3340 \cs_new_protected:Nn
3341 \@@_tl_set_from_cs:NNn

```



```

3342 {
3343   \tl_set:Nn
3344     \l_tmpa_tl
3345     { #2 }
3346   \int_step_inline:nn
3347     { #3 }
3348     {
3349       \exp_args:Nnc
3350         \tl_put_right:Nn
3351         \l_tmpa_tl
3352         { @@_tl_set_from_cs_parameter_ ##1 }
3353     }
3354   \exp_args:NNV
3355     \tl_set:No
3356     \l_tmpb_tl
3357     \l_tmpa_tl
3358   \regex_replace_all:nnN
3359     { \cP. }
3360     { \0\0 }
3361     \l_tmpb_tl
3362   \int_step_inline:nn
3363     { #3 }
3364     {
3365       \regex_replace_all:nnN
3366         { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3367         { \cP\# ##1 }
3368         \l_tmpb_tl
3369     }
3370   \tl_set:NV
3371     #1
3372     \l_tmpb_tl
3373 }
3374 \cs_generate_variant:Nn
3375   \@_define_renderer:nNn
3376   { ncV }
3377 \cs_generate_variant:Nn
3378   \cs_generate_from_arg_count:NNnn
3379   { NNnV }
3380 \cs_generate_variant:Nn
3381   \tl_put_left:Nn
3382   { Nv }
3383 \keys_define:nn
3384   { markdown/options }
3385   {
3386     renderers .code:n = {
3387       \bool_gset_false:N
3388         \g_@@_unprotected_renderer_bool

```

```

3389     \keys_set:nn
3390     { markdown/options/renderers }
3391     { #1 }
3392 },
3393 unprotectedRenderers .code:n = {
3394     \bool_gset_true:N
3395     \g_@@_unprotected_renderer_bool
3396     \keys_set:nn
3397     { markdown/options/renderers }
3398     { #1 }
3399 },
3400 }

```

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.
  }
}

```

```

3401 \tl_new:N
3402 \l_@@_renderer_glob_definition_tl
3403 \seq_new:N
3404 \l_@@_renderer_glob_results_seq
3405 \regex_const:Nn
3406 \c_@@_appending_key_regex
3407 { \s*+& }
3408 \keys_define:nn
3409 { markdown/options/renderers }
3410 {
3411     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 += {...},
    },
  },
},
}

```

```

3412   \regex_match:NVTF
3413   \c_@@_appending_key_regex
3414   \l_keys_key_str
3415   {
3416     \bool_gset_true:N
3417     \g_@@_appending_renderer_bool
3418     \tl_set:NV
3419     \l_tmpa_tl
3420     \l_keys_key_str
3421     \regex_replace_once:NnN
3422     \c_@@_appending_key_regex
3423     { }
3424     \l_tmpa_tl
3425     \tl_set:Nx
3426     \l_tmpb_tl
3427     { { \l_tmpa_tl } = }
3428     \tl_put_right:Nn
3429     \l_tmpb_tl
3430     { { #1 } }
3431     \keys_set:nV
3432     { markdown/options/renderers }
3433     \l_tmpb_tl
3434     \bool_gset_false:N
3435     \g_@@_appending_renderer_bool
3436   }

```

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.
  }
}
```

```
3437     {
3438         \@@_glob_seq:VnN
3439         \l_keys_key_str
3440         { g_@@_renderers_seq }
3441         \l_@@_renderer_glob_results_seq
3442     \seq_if_empty:NTF
3443     \l_@@_renderer_glob_results_seq
3444     {
3445         \msg_error:nnV
3446         { markdown }
3447         { undefined-renderer }
3448         \l_keys_key_str
3449     }
3450     {
3451         \tl_set:Nn
```

```

3452         \l_@@_renderer_glob_definition_tl
3453         { \exp_not:n { #1 } }
3454     \seq_map_inline:Nn
3455         \l_@@_renderer_glob_results_seq
3456         {
3457             \tl_set:Nn
3458                 \l_tmpa_tl
3459                 { { ##1 } = }
3460             \tl_put_right:Nx
3461                 \l_tmpa_tl
3462                 { { \l_@@_renderer_glob_definition_tl } }
3463             \keys_set:nV
3464                 { markdown/options/renderers }
3465                 \l_tmpa_tl
3466         }
3467     }
3468 }
3469 },
3470 }
3471 \msg_new:nnn
3472     { markdown }
3473     { undefined-renderer }
3474     {
3475         Renderer~#1~is~undefined.
3476     }
3477 \cs_generate_variant:Nn
3478     \@@_glob_seq:nnN
3479     { VnN }
3480 \cs_generate_variant:Nn
3481     \cs_generate_from_arg_count:NNnn
3482     { cNVV }
3483 \cs_generate_variant:Nn
3484     \msg_error:nnn
3485     { nnV }
3486 \prg_generate_conditional_variant:Nnn
3487     \regex_match:Nn
3488     { NV }
3489     { TF }
3490 \prop_new:N
3491     \g_@@_glob_cache_prop
3492 \tl_new:N
3493     \l_@@_current_glob_tl
3494 \cs_new:Nn
3495     \@@_glob_seq:nnN
3496     {
3497         \tl_set:Nn
3498             \l_@@_current_glob_tl

```

```

3499     { ^ #1 $ }
3500   \prop_get:NeNTF
3501     \g_@@_glob_cache_prop
3502     { #2 / \l_@@_current_glob_tl }
3503     \l_tmpa_clist
3504     {
3505       \seq_set_from_clist:NN
3506       #3
3507       \l_tmpa_clist
3508     }
3509     {
3510       \seq_clear:N
3511       #3
3512       \regex_replace_all:nnN
3513       { \* }
3514       { .* }
3515       \l_@@_current_glob_tl
3516       \regex_set:NV
3517       \l_tmpa_regex
3518       \l_@@_current_glob_tl
3519       \seq_map_inline:cn
3520       { #2 }
3521       {
3522         \regex_match:NnT
3523         \l_tmpa_regex
3524         { ##1 }
3525         {
3526           \seq_put_right:Nn
3527           #3
3528           { ##1 }
3529         }
3530       }
3531     \clist_set_from_seq:NN
3532     \l_tmpa_clist
3533     #3
3534     \prop_gput:NeV
3535     \g_@@_glob_cache_prop
3536     { #2 / \l_@@_current_glob_tl }
3537     \l_tmpa_clist
3538   }
3539 }
3540 \cs_generate_variant:Nn
3541   \regex_set:Nn
3542   { NV }
3543 \cs_generate_variant:Nn
3544   \prop_gput:Nnn
3545   { 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.

```

3546 \str_if_eq:VVT
3547   \c_@@_top_layer_tl
3548   \c_@@_option_layer_plain_tex_tl
3549   {
3550     \@@_define_renderers:
3551   }
3552 \ExplSyntaxOff

```

## 2.2.6 Token Renderer Prototypes

### 2.2.6.1 YAML Metadata Renderer Prototypes

For simple YAML metadata, a simple high-level interface is provided that can be programmed by setting the expl3 key-values [2] for the module `markdown/jekyllData`.

```

3553 \ExplSyntaxOn
3554 \keys_define:nn
3555   { markdown/jekyllData }
3556   { }
3557 \ExplSyntaxOff

```

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

```

3558 \ExplSyntaxOn
3559 \@@_with_various_cases:nn
3560   { jekyllDataRenderers }
3561   {
3562     \keys_define:nn
3563       { markdown/options }
3564       {
3565         #1 .code:n = {
3566           \tl_set:Nn
3567             \l_tmpa_tl
3568             { ##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.

```

3569         \tl_replace_all:NnV
3570         \l_tmpa_tl
3571         { / }
3572         \c_backslash_str
3573         \keys_set:nV

```

```

3574         { markdown/options/jekyll-data-renderers }
3575         \l_tmpa_tl
3576     },
3577 }
3578 }
3579 \keys_define:nn
3580 { markdown/options/jekyll-data-renderers }
3581 {
3582     unknown .code:n = {
3583         \tl_set_eq:NN
3584         \l_tmpa_tl
3585         \l_keys_key_str
3586         \tl_replace_all:NVn
3587         \l_tmpa_tl
3588         \c_backslash_str
3589         { / }
3590         \tl_put_right:Nn
3591         \l_tmpa_tl
3592         {
3593             .code:n = { #1 }
3594         }
3595         \keys_define:nV
3596         { markdown/jekyllData }
3597         \l_tmpa_tl
3598     }
3599 }
3600 \cs_generate_variant:Nn
3601 \keys_define:nn
3602 { nV }
3603 \ExplSyntaxOff

```

For complex YAML metadata, the option `jekyllDataKeyValue=<module>` [13] can be used to route the processing of all YAML metadata in the current  $\TeX$  group to the key-values from  $\langle module \rangle$ .

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

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

Whereas the key `rendererPrototypes` defines protected functions, which are usually preferable for typesetting, the key `unprotectedRendererPrototypes` de-



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

```

3604 \ExplSyntaxOn
3605 \cs_new:Nn \@@_define_renderer_prototypes:
3606   {
3607     \seq_map_inline:Nn
3608       \g_@@_renderers_seq
3609       {
3610         \@@_define_renderer_prototype:n
3611         { ##1 }
3612       }
3613   }
3614 \cs_new:Nn \@@_define_renderer_prototype:n
3615   {
3616     \@@_renderer_prototype_tl_to_csname:nN
3617     { #1 }
3618     \l_tmpa_tl
3619     \prop_get:NnN
3620       \g_@@_renderer_arities_prop
3621       { #1 }
3622       \l_tmpb_tl
3623     \@@_define_renderer_prototype:ncV
3624     { #1 }
3625     { \l_tmpa_tl }
3626     \l_tmpb_tl
3627   }
3628 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3629   {
3630     \tl_set:Nn
3631       \l_tmpa_tl
3632       { \str_uppercase:n { #1 } }
3633     \tl_set:Nx
3634       #2
3635       {
3636         markdownRenderer
3637         \tl_head:f { \l_tmpa_tl }
3638         \tl_tail:n { #1 }
3639         Prototype
3640       }
3641   }
3642 \tl_new:N
3643   \l_@@_renderer_prototype_definition_tl
3644 \bool_new:N
3645   \g_@@_appending_renderer_prototype_bool
3646 \bool_new:N
3647   \g_@@_unprotected_renderer_prototype_bool
3648 \cs_new:Nn \@@_define_renderer_prototype:nNn

```

```

3649 {
3650   \keys_define:nn
3651     { markdown/options/renderer-prototypes }
3652     {
3653       #1 .code:n = {
3654         \tl_set:Nn
3655           \l_@@_renderer_prototype_definition_tl
3656           { ##1 }
3657         \regex_replace_all:nnN
3658           { \cP\#0 }
3659           { #1 }
3660         \l_@@_renderer_prototype_definition_tl
3661         \bool_if:NT
3662           \g_@@_appending_renderer_prototype_bool
3663           {
3664             \@@_tl_set_from_cs:NNn
3665             \l_tmpa_tl
3666             #2
3667             { #3 }
3668             \tl_put_left:NV
3669             \l_@@_renderer_prototype_definition_tl
3670             \l_tmpa_tl
3671           }
3672         \bool_if:NTF
3673           \g_@@_unprotected_renderer_prototype_bool
3674           {
3675             \tl_set:Nn
3676             \l_tmpa_tl
3677             { \cs_set:Npn }
3678           }
3679           {
3680             \tl_set:Nn
3681             \l_tmpa_tl
3682             { \cs_set_protected:Npn }
3683           }
3684         \exp_last_unbraced:NNV
3685         \cs_generate_from_arg_count:NNnV
3686         #2
3687         \l_tmpa_tl
3688         { #3 }
3689         \l_@@_renderer_prototype_definition_tl
3690       },
3691     }

```

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.

```

3692   \str_if_eq:nnF
3693     { #1 }
3694     { jekyllDataString }
3695     {
3696       \cs_if_free:NT
3697         #2
3698         {
3699           \cs_generate_from_arg_count:NNnn
3700             #2
3701             \cs_gset_protected:Npn
3702               { #3 }
3703               { }
3704           }
3705       }
3706   }
3707 \cs_generate_variant:Nn
3708   \@@_define_renderer_prototype:nNn
3709   { 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.
  }
}

```

```

3710 \keys_define:nn
3711   { markdown/options/renderer-prototypes }
3712   {
3713     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 += {...},
    },
  }
},
}
}

```

```

3714 \regex_match:NVTF
3715 \c_@@_appending_key_regex
3716 \l_keys_key_str
3717 {
3718   \bool_gset_true:N
3719   \g_@@_appending_renderer_prototype_bool
3720   \tl_set:NV
3721   \l_tmpa_tl
3722   \l_keys_key_str
3723   \regex_replace_once:NnN
3724   \c_@@_appending_key_regex
3725   { }
3726   \l_tmpa_tl
3727   \tl_set:Nx
3728   \l_tmpb_tl
3729   { { \l_tmpa_tl } = }
3730   \tl_put_right:Nn
3731   \l_tmpb_tl
3732   { { #1 } }
3733   \keys_set:nV
3734   { markdown/options/renderer-prototypes }
3735   \l_tmpb_tl
3736   \bool_gset_false:N
3737   \g_@@_appending_renderer_prototype_bool

```

```
3738     }
```

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.
}
```

```
3739     {
3740         \@@_glob_seq:VnN
3741         \l_keys_key_str
3742         { g_@@_renderers_seq }
3743         \l_@@_renderer_glob_results_seq
3744         \seq_if_empty:NTF
3745         \l_@@_renderer_glob_results_seq
3746         {
3747             \msg_error:nnV
3748             { markdown }
3749             { undefined-renderer-prototype }
3750             \l_keys_key_str
```

```

3751         }
3752     {
3753         \tl_set:Nn
3754             \l_@@_renderer_glob_definition_tl
3755             { \exp_not:n { #1 } }
3756         \seq_map_inline:Nn
3757             \l_@@_renderer_glob_results_seq
3758             {
3759                 \tl_set:Nn
3760                     \l_tmpa_tl
3761                     { { ##1 } = }
3762                 \tl_put_right:Nx
3763                     \l_tmpa_tl
3764                     { { \l_@@_renderer_glob_definition_tl } }
3765                 \keys_set:nV
3766                     { markdown/options/renderer-prototypes }
3767                     \l_tmpa_tl
3768             }
3769         }
3770     }
3771 },
3772 }
3773 \msg_new:nnn
3774 { markdown }
3775 { undefined-renderer-prototype }
3776 {
3777     Renderer~prototype~#1~is~undefined.
3778 }
3779 \@@_with_various_cases:nn
3780 { rendererPrototypes }
3781 {
3782     \keys_define:nn
3783         { markdown/options }
3784         {
3785             #1 .code:n = {
3786                 \bool_gset_false:N
3787                     \g_@@_unprotected_renderer_prototype_bool
3788                 \keys_set:nn
3789                     { markdown/options/renderer-prototypes }
3790                     { ##1 }
3791             },
3792         }
3793     }
3794 \@@_with_various_cases:nn
3795 { unprotectedRendererPrototypes }
3796 {
3797     \keys_define:nn

```

```

3798     { markdown/options }
3799     {
3800     #1 .code:n = {
3801         \bool_gset_true:N
3802         \g_@@_unprotected_renderer_prototype_bool
3803         \keys_set:nn
3804         { markdown/options/renderer-prototypes }
3805         { ##1 }
3806     },
3807     }
3808 }

```

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

```

3809 \str_if_eq:VVT
3810 \c_@@_top_layer_tl
3811 \c_@@_option_layer_plain_tex_tl
3812 {
3813     \@@_define_renderer_prototypes:
3814 }
3815 \ExplSyntaxOff

```

### 2.2.7 Logging Facilities

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

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

### 2.2.8 Miscellanea

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

```

3816 \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.

```
3817 \let\markdownReadAndConvert\relax
3818 \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*.

```
3819 \catcode`\|=0\catcode`\=12%
3820 |gdef|markdownBegin{%
3821   |markdownReadAndConvert{\markdownEnd}%
3822                               {\markdownEnd}}%
3823 |gdef|yamlBegin{%
3824   |begingroup
3825   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3826   |markdownReadAndConvert{\yamlEnd}%
3827                               {\yamlEnd}}%
3828 |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.

```
3829 \ExplSyntaxOn
3830 \keys_define:nn
3831   { markdown/options }
3832   {
3833     code .code:n = { #1 },
3834   }
3835 \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.

```
3836 \ExplSyntaxOn
```



```

3837 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3838 \cs_generate_variant:Nn
3839   \tl_const:Nn
3840   { NV }
3841 \tl_if_exist:NF
3842   \c_@@_top_layer_tl
3843   {
3844     \tl_const:NV
3845       \c_@@_top_layer_tl
3846       \c_@@_option_layer_latex_tl
3847   }
3848 \ExplSyntaxOff
3849 \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.

```

3850 \newenvironment{markdown}\relax\relax
3851 \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:

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

You can't directly extend the `markdown` 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.

```
3852 \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. [14, 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 [15, Section 5.1], which supports the `finalizcache` and `frozencache` 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 `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

```

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

```

We hope that other packages will support the `finalizcache` and `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.

```
3853 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3854 \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.

```
3855 \ExplSyntaxOn
3856 \str_if_eq:VVT
3857   \c_@@_top_layer_tl
3858   \c_@@_option_layer_latex_tl
3859   {
3860     \@@_define_option_commands_and_keyvals:
3861     \@@_define_renderers:
3862     \@@_define_renderer_prototypes:
3863   }
3864 \ExplSyntaxOff
```

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

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

### 2.3.4 Themes

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

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

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

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

```
3865 \newif\ifmarkdownLaTeXLoaded
3866 \markdownLaTeXLoadedfalse
```

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

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.

```
3867 \ExplSyntaxOn
3868 \prop_new:N
3869 \g_@@_latex_built_in_themes_prop
3870 \ExplSyntaxOff
```

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

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

```
3871 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
```

At the end of the L<sup>A</sup>T<sub>E</sub>X module, we load the `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>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).

```
3872 \ExplSyntaxOn
3873 \str_if_eq:VVT
3874 \c_@@_top_layer_tl
3875 \c_@@_option_layer_latex_tl
3876 {
```



```

3877 \use:c
3878   { ExplSyntaxOff }
3879 \AtEndOfPackage
3880   {
3881     \@@_if_option:nF
3882     { noDefaults }
3883     {
3884       \@@_if_option:nTF
3885       { experimental }
3886       {
3887         \@@_setup:n
3888         { theme = witiko/markdown/defaults@experimental }
3889       }
3890       {
3891         \@@_setup:n
3892         { theme = witiko/markdown/defaults }
3893       }
3894     }
3895   }
3896 \use:c
3897   { ExplSyntaxOn }
3898 }
3899 \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.

```

3900 \ExplSyntaxOn
3901 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3902 \cs_generate_variant:Nn
3903   \tl_const:Nn
3904   { NV }
3905 \tl_if_exist:NF
3906   \c_@@_top_layer_tl
3907   {
3908     \tl_const:NV
3909     \c_@@_top_layer_tl
3910     \c_@@_option_layer_context_tl
3911   }
3912 \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

ConTEXt options used during the conversion from markdown to plain TEX. The rest of the interface is inherited from the plain TEX interface (see Section 2.2).

```

3913 \writestatus{loading}{ConTEXt User Module / markdown}%
3914 \startmodule[markdown]
3915 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3916 \do#\do\^\do\_do\%do\~}%
3917 \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.

```

3918 \let\startmarkdown\relax
3919 \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.

```
3920 \let\startyaml\relax
3921 \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.

```
3922 \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.

```
3923 \let\inputyaml\relax
```

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

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

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

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

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

## 2.4.2 Options

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

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

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

```

3924 \ExplSyntaxOn
3925 \cs_new:Npn
3926   \setupmarkdown
3927   [ #1 ]
3928   {
3929     \@@_setup:n
3930     { #1 }
3931   }

```

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

```

3932 \cs_gset_eq:NN
3933   \setupyaml
3934   \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.

```

3935 \cs_new:Nn \@@_caseless:N
3936   {
3937     \regex_replace_all:nnN
3938     { ([a-z])([A-Z]) }
3939     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3940     #1
3941     \tl_set:Nx
3942     #1
3943     { #1 }
3944   }
3945 \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.

```

3946 \str_if_eq:VVT
3947   \c_@@_top_layer_tl
3948   \c_@@_option_layer_context_tl
3949   {
3950     \@@_define_option_commands_and_keyvals:
3951     \@@_define_renderers:
3952     \@@_define_renderer_prototypes:
3953   }

```

### 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.

```
3954 \prop_new:N
3955   \g_@@_context_built_in_themes_prop
3956 \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.

```
3957 \startmodule[markdownthemewitiko_markdown_defaults]
3958 \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  $\LaTeX$  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).

```
3959 local upper, format, length =
3960   string.upper, string.format, string.len
3961 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3962   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3963   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.

```
3964 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.

```
3965 function util.err(msg, exit_code)
3966   io.stderr:write("markdown.lua: " .. msg .. "\n")
3967   os.exit(exit_code or 1)
3968 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.

```
3969 function util.cache(dir, string, salt, transform, suffix)
3970   local digest = md5.sumhexa(string .. (salt or ""))
3971   local name = util.pathname(dir, digest .. suffix)
3972   local file = io.open(name, "r")
3973   if file == nil then -- If no cache entry exists, create a new one.
3974     file = assert(io.open(name, "w"),
3975       [[Could not open file ]] .. name .. [[ for writing]])
3976     local result = string
```

```

3977     if transform ~= nil then
3978         result = transform(result)
3979     end
3980     assert(file:write(result))
3981     assert(file:close())
3982 end
3983 return name
3984 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.

```

3985 function util.cache_verbatim(dir, string)
3986     local name = util.cache(dir, string, nil, nil, ".verbatim")
3987     return name
3988 end

```

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

```

3989 function util.table_copy(t)
3990     local u = { }
3991     for k, v in pairs(t) do u[k] = v end
3992     return setmetatable(u, getmetatable(t))
3993 end

```

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

```

3994 function util.encode_json_string(s)
3995     s = s:gsub([[\\]], [[\\]])
3996     s = s:gsub([[\"]], [[\"]])
3997     return [[\"]] .. s .. [[\"]]
3998 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 [16, Chapter 21].

```

3999 function util.expand_tabs_in_line(s, tabstop)
4000     local tab = tabstop or 4
4001     local corr = 0
4002     return (s:gsub(")\t", function(p)
4003         local sp = tab - (p - 1 + corr) % tab
4004         corr = corr - 1 + sp
4005         return string.rep(" ", sp)
4006     end))
4007 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.

```
4008 function util.walk(t, f)
4009   local typ = type(t)
4010   if typ == "string" then
4011     f(t)
4012   elseif typ == "table" then
4013     local i = 1
4014     local n
4015     n = t[i]
4016     while n do
4017       util.walk(n, f)
4018       i = i + 1
4019       n = t[i]
4020     end
4021   elseif typ == "function" then
4022     local ok, val = pcall(t)
4023     if ok then
4024       util.walk(val, f)
4025     end
4026   else
4027     f(tostring(t))
4028   end
4029 end
```

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

```
4030 function util.flatten(ary)
4031   local new = {}
4032   for _,v in ipairs(ary) do
4033     if type(v) == "table" then
4034       for _,w in ipairs(util.flatten(v)) do
4035         new[#new + 1] = w
4036       end
4037     else
4038       new[#new + 1] = v
4039     end
4040   end
4041   return new
4042 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.

```
4043 function util.rope_to_string(rope)
4044   local buffer = {}
4045   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4046   return table.concat(buffer)
```

```
4047 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.

```
4048 function util.rope_last(rope)
4049   if #rope == 0 then
4050     return nil
4051   else
4052     local l = rope[#rope]
4053     if type(l) == "table" then
4054       return util.rope_last(l)
4055     else
4056       return l
4057     end
4058   end
4059 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  $1 \leq i \leq \#ary$ .

```
4060 function util.intersperse(ary, x)
4061   local new = {}
4062   local l = #ary
4063   for i,v in ipairs(ary) do
4064     local n = #new
4065     new[n + 1] = v
4066     if i ~= l then
4067       new[n + 2] = x
4068     end
4069   end
4070   return new
4071 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  $1 \leq i \leq \#ary$ .

```
4072 function util.map(ary, f)
4073   local new = {}
4074   for i,v in ipairs(ary) do
4075     new[i] = f(v)
4076   end
4077   return new
4078 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.

```
4079 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
4080 local char_escapes_list = ""
4081 for i,_ in pairs(char_escapes) do
4082   char_escapes_list = char_escapes_list .. i
4083 end
```

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

```
4084 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.

```
4085 if string_escapes then
4086   for k,v in pairs(string_escapes) do
4087     escapable = P(k) / v + escapable
4088   end
4089 end
```

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

```
4090 local escape_string = Cs((escapable + any)^0)
```

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

```
4091 return function(s)
4092   return lpeg.match(escape_string, s)
4093 end
4094 end
```

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

```
4095 function util.pathname(dir, file)
4096   if #dir == 0 then
4097     return file
4098   else
4099     return dir .. "/" .. file
4100   end
4101 end
```

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

```
4102 function util.salt(options)
4103   local opt_string = {}
4104   for k, _ in pairs(defaultOptions) do
4105     local v = options[k]
4106     if type(v) == "table" then
4107       for _, i in ipairs(v) do
4108         opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4109       end
4110     end
4111   end
4112   return opt_string
4113 end
```

The `cacheDir` option is disregarded.

```
4110   elseif k ~= "cacheDir" then
4111     opt_string[#opt_string+1] = k .. "=" .. tostring(v)
4112   end
4113 end
4114 table.sort(opt_string)
4115 local salt = table.concat(opt_string, ",")
4116             .. "," .. metadata.version
4117 return salt
4118 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.

```
4119 function util.warning(s)
4120   io.stderr:write("Warning: " .. s .. "\n")
4121 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.

```
4122 local entities = {}
4123
4124 local character_entities = {
4125   ["Tab"] = 9,
4126   ["NewLine"] = 10,
4127   ["excl"] = 33,
4128   ["QUOT"] = 34,
4129   ["quot"] = 34,
4130   ["num"] = 35,
4131   ["dollar"] = 36,
4132   ["percnt"] = 37,
4133   ["AMP"] = 38,
4134   ["amp"] = 38,
4135   ["apos"] = 39,
```

4136 ["lpar"] = 40,  
 4137 ["rpar"] = 41,  
 4138 ["ast"] = 42,  
 4139 ["midast"] = 42,  
 4140 ["plus"] = 43,  
 4141 ["comma"] = 44,  
 4142 ["period"] = 46,  
 4143 ["sol"] = 47,  
 4144 ["colon"] = 58,  
 4145 ["semi"] = 59,  
 4146 ["LT"] = 60,  
 4147 ["lt"] = 60,  
 4148 ["nvlT"] = {60, 8402},  
 4149 ["bne"] = {61, 8421},  
 4150 ["equals"] = 61,  
 4151 ["GT"] = 62,  
 4152 ["gt"] = 62,  
 4153 ["nvgt"] = {62, 8402},  
 4154 ["quest"] = 63,  
 4155 ["commat"] = 64,  
 4156 ["lbrack"] = 91,  
 4157 ["lsqb"] = 91,  
 4158 ["bsol"] = 92,  
 4159 ["rbrack"] = 93,  
 4160 ["rsqb"] = 93,  
 4161 ["Hat"] = 94,  
 4162 ["UnderBar"] = 95,  
 4163 ["lowbar"] = 95,  
 4164 ["DiacriticalGrave"] = 96,  
 4165 ["grave"] = 96,  
 4166 ["fjlig"] = {102, 106},  
 4167 ["lbrace"] = 123,  
 4168 ["lcub"] = 123,  
 4169 ["VerticalLine"] = 124,  
 4170 ["verbar"] = 124,  
 4171 ["vert"] = 124,  
 4172 ["rbrace"] = 125,  
 4173 ["rcub"] = 125,  
 4174 ["NonBreakingSpace"] = 160,  
 4175 ["nbsp"] = 160,  
 4176 ["iexcl"] = 161,  
 4177 ["cent"] = 162,  
 4178 ["pound"] = 163,  
 4179 ["curren"] = 164,  
 4180 ["yen"] = 165,  
 4181 ["brvbar"] = 166,  
 4182 ["sect"] = 167,

4183 ["Dot"] = 168,  
4184 ["DoubleDot"] = 168,  
4185 ["die"] = 168,  
4186 ["uml"] = 168,  
4187 ["COPY"] = 169,  
4188 ["copy"] = 169,  
4189 ["ordf"] = 170,  
4190 ["laquo"] = 171,  
4191 ["not"] = 172,  
4192 ["shy"] = 173,  
4193 ["REG"] = 174,  
4194 ["circledR"] = 174,  
4195 ["reg"] = 174,  
4196 ["macr"] = 175,  
4197 ["strns"] = 175,  
4198 ["deg"] = 176,  
4199 ["PlusMinus"] = 177,  
4200 ["plusmn"] = 177,  
4201 ["pm"] = 177,  
4202 ["sup2"] = 178,  
4203 ["sup3"] = 179,  
4204 ["DiacriticalAcute"] = 180,  
4205 ["acute"] = 180,  
4206 ["micro"] = 181,  
4207 ["para"] = 182,  
4208 ["CenterDot"] = 183,  
4209 ["centerdot"] = 183,  
4210 ["middot"] = 183,  
4211 ["Cedilla"] = 184,  
4212 ["cedil"] = 184,  
4213 ["sup1"] = 185,  
4214 ["ordm"] = 186,  
4215 ["raquo"] = 187,  
4216 ["frac14"] = 188,  
4217 ["frac12"] = 189,  
4218 ["half"] = 189,  
4219 ["frac34"] = 190,  
4220 ["iquest"] = 191,  
4221 ["Agrave"] = 192,  
4222 ["Aacute"] = 193,  
4223 ["Acirc"] = 194,  
4224 ["Atilde"] = 195,  
4225 ["Auml"] = 196,  
4226 ["Aring"] = 197,  
4227 ["angst"] = 197,  
4228 ["AElig"] = 198,  
4229 ["Ccedil"] = 199,

4230 ["Egrave"] = 200,  
4231 ["Eacute"] = 201,  
4232 ["Ecirc"] = 202,  
4233 ["Euml"] = 203,  
4234 ["Igrave"] = 204,  
4235 ["Iacute"] = 205,  
4236 ["Icirc"] = 206,  
4237 ["Iuml"] = 207,  
4238 ["ETH"] = 208,  
4239 ["Ntilde"] = 209,  
4240 ["Ograve"] = 210,  
4241 ["Oacute"] = 211,  
4242 ["Ocirc"] = 212,  
4243 ["Otilde"] = 213,  
4244 ["Ouml"] = 214,  
4245 ["times"] = 215,  
4246 ["Oslash"] = 216,  
4247 ["Ugrave"] = 217,  
4248 ["Uacute"] = 218,  
4249 ["Ucirc"] = 219,  
4250 ["Uuml"] = 220,  
4251 ["Yacute"] = 221,  
4252 ["THORN"] = 222,  
4253 ["szlig"] = 223,  
4254 ["agrave"] = 224,  
4255 ["aacute"] = 225,  
4256 ["acirc"] = 226,  
4257 ["atilde"] = 227,  
4258 ["auml"] = 228,  
4259 ["aring"] = 229,  
4260 ["aelig"] = 230,  
4261 ["ccedil"] = 231,  
4262 ["egrave"] = 232,  
4263 ["eacute"] = 233,  
4264 ["ecirc"] = 234,  
4265 ["euml"] = 235,  
4266 ["igrave"] = 236,  
4267 ["iacute"] = 237,  
4268 ["icirc"] = 238,  
4269 ["iuml"] = 239,  
4270 ["eth"] = 240,  
4271 ["ntilde"] = 241,  
4272 ["ograve"] = 242,  
4273 ["oacute"] = 243,  
4274 ["ocirc"] = 244,  
4275 ["otilde"] = 245,  
4276 ["ouml"] = 246,

4277 ["div"] = 247,  
4278 ["divide"] = 247,  
4279 ["oslash"] = 248,  
4280 ["ugrave"] = 249,  
4281 ["uacute"] = 250,  
4282 ["ucirc"] = 251,  
4283 ["uuml"] = 252,  
4284 ["yacute"] = 253,  
4285 ["thorn"] = 254,  
4286 ["yuml"] = 255,  
4287 ["Amacr"] = 256,  
4288 ["amacr"] = 257,  
4289 ["Abreve"] = 258,  
4290 ["abreve"] = 259,  
4291 ["Aogon"] = 260,  
4292 ["aogon"] = 261,  
4293 ["Cacute"] = 262,  
4294 ["cacute"] = 263,  
4295 ["Ccirc"] = 264,  
4296 ["ccirc"] = 265,  
4297 ["Cdot"] = 266,  
4298 ["cdot"] = 267,  
4299 ["Ccaron"] = 268,  
4300 ["ccaron"] = 269,  
4301 ["Dcaron"] = 270,  
4302 ["dcaron"] = 271,  
4303 ["Dstrok"] = 272,  
4304 ["dstrok"] = 273,  
4305 ["Emacr"] = 274,  
4306 ["emacr"] = 275,  
4307 ["Edot"] = 278,  
4308 ["edot"] = 279,  
4309 ["Eogon"] = 280,  
4310 ["eogon"] = 281,  
4311 ["Ecaron"] = 282,  
4312 ["ecaron"] = 283,  
4313 ["Gcirc"] = 284,  
4314 ["gcirc"] = 285,  
4315 ["Gbreve"] = 286,  
4316 ["gbreve"] = 287,  
4317 ["Gdot"] = 288,  
4318 ["gdot"] = 289,  
4319 ["Gcedil"] = 290,  
4320 ["Hcirc"] = 292,  
4321 ["hcirc"] = 293,  
4322 ["Hstrok"] = 294,  
4323 ["hstrok"] = 295,



4324 ["Itilde"] = 296,  
4325 ["itilde"] = 297,  
4326 ["Imacr"] = 298,  
4327 ["imacr"] = 299,  
4328 ["Iogon"] = 302,  
4329 ["iogon"] = 303,  
4330 ["Idot"] = 304,  
4331 ["imath"] = 305,  
4332 ["inodot"] = 305,  
4333 ["IJlig"] = 306,  
4334 ["ijlig"] = 307,  
4335 ["Jcirc"] = 308,  
4336 ["jcirc"] = 309,  
4337 ["Kcedil"] = 310,  
4338 ["kcedil"] = 311,  
4339 ["kgreen"] = 312,  
4340 ["Lacute"] = 313,  
4341 ["lacute"] = 314,  
4342 ["Lcedil"] = 315,  
4343 ["lcedil"] = 316,  
4344 ["Lcaron"] = 317,  
4345 ["lcaron"] = 318,  
4346 ["Lmidot"] = 319,  
4347 ["lmidot"] = 320,  
4348 ["Lstrok"] = 321,  
4349 ["lstrok"] = 322,  
4350 ["Nacute"] = 323,  
4351 ["nacute"] = 324,  
4352 ["Ncedil"] = 325,  
4353 ["ncedil"] = 326,  
4354 ["Ncaron"] = 327,  
4355 ["ncaron"] = 328,  
4356 ["napos"] = 329,  
4357 ["ENG"] = 330,  
4358 ["eng"] = 331,  
4359 ["Omacr"] = 332,  
4360 ["omacr"] = 333,  
4361 ["Odblac"] = 336,  
4362 ["odblac"] = 337,  
4363 ["OElig"] = 338,  
4364 ["oelig"] = 339,  
4365 ["Racute"] = 340,  
4366 ["racute"] = 341,  
4367 ["Rcedil"] = 342,  
4368 ["rcedil"] = 343,  
4369 ["Rcaron"] = 344,  
4370 ["rcaron"] = 345,

4371 ["Sacute"] = 346,  
4372 ["sacute"] = 347,  
4373 ["Scirc"] = 348,  
4374 ["scirc"] = 349,  
4375 ["Scedil"] = 350,  
4376 ["scedil"] = 351,  
4377 ["Scaron"] = 352,  
4378 ["scaron"] = 353,  
4379 ["Tcedil"] = 354,  
4380 ["tcedil"] = 355,  
4381 ["Tcaron"] = 356,  
4382 ["tcaron"] = 357,  
4383 ["Tstrok"] = 358,  
4384 ["tstrok"] = 359,  
4385 ["Utilde"] = 360,  
4386 ["utilde"] = 361,  
4387 ["Umacr"] = 362,  
4388 ["umacr"] = 363,  
4389 ["Ubreve"] = 364,  
4390 ["ubreve"] = 365,  
4391 ["Uring"] = 366,  
4392 ["uring"] = 367,  
4393 ["Udblac"] = 368,  
4394 ["udblac"] = 369,  
4395 ["Uogon"] = 370,  
4396 ["uogon"] = 371,  
4397 ["Wcirc"] = 372,  
4398 ["wcirc"] = 373,  
4399 ["Ycirc"] = 374,  
4400 ["ycirc"] = 375,  
4401 ["Yuml"] = 376,  
4402 ["Zacute"] = 377,  
4403 ["zacute"] = 378,  
4404 ["Zdot"] = 379,  
4405 ["zdot"] = 380,  
4406 ["Zcaron"] = 381,  
4407 ["zcaron"] = 382,  
4408 ["fnof"] = 402,  
4409 ["imped"] = 437,  
4410 ["gacute"] = 501,  
4411 ["jmath"] = 567,  
4412 ["circ"] = 710,  
4413 ["Hacek"] = 711,  
4414 ["caron"] = 711,  
4415 ["Breve"] = 728,  
4416 ["breve"] = 728,  
4417 ["DiacriticalDot"] = 729,

4418 ["dot"] = 729,  
4419 ["ring"] = 730,  
4420 ["ogon"] = 731,  
4421 ["DiacriticalTilde"] = 732,  
4422 ["tilde"] = 732,  
4423 ["DiacriticalDoubleAcute"] = 733,  
4424 ["dblac"] = 733,  
4425 ["DownBreve"] = 785,  
4426 ["Alpha"] = 913,  
4427 ["Beta"] = 914,  
4428 ["Gamma"] = 915,  
4429 ["Delta"] = 916,  
4430 ["Epsilon"] = 917,  
4431 ["Zeta"] = 918,  
4432 ["Eta"] = 919,  
4433 ["Theta"] = 920,  
4434 ["Iota"] = 921,  
4435 ["Kappa"] = 922,  
4436 ["Lambda"] = 923,  
4437 ["Mu"] = 924,  
4438 ["Nu"] = 925,  
4439 ["Xi"] = 926,  
4440 ["Omicron"] = 927,  
4441 ["Pi"] = 928,  
4442 ["Rho"] = 929,  
4443 ["Sigma"] = 931,  
4444 ["Tau"] = 932,  
4445 ["Upsilon"] = 933,  
4446 ["Phi"] = 934,  
4447 ["Chi"] = 935,  
4448 ["Psi"] = 936,  
4449 ["Omega"] = 937,  
4450 ["ohm"] = 937,  
4451 ["alpha"] = 945,  
4452 ["beta"] = 946,  
4453 ["gamma"] = 947,  
4454 ["delta"] = 948,  
4455 ["epsi"] = 949,  
4456 ["epsilon"] = 949,  
4457 ["zeta"] = 950,  
4458 ["eta"] = 951,  
4459 ["theta"] = 952,  
4460 ["iota"] = 953,  
4461 ["kappa"] = 954,  
4462 ["lambda"] = 955,  
4463 ["mu"] = 956,  
4464 ["nu"] = 957,

4465 ["xi"] = 958,  
4466 ["omicron"] = 959,  
4467 ["pi"] = 960,  
4468 ["rho"] = 961,  
4469 ["sigmaf"] = 962,  
4470 ["sigmav"] = 962,  
4471 ["varsigma"] = 962,  
4472 ["sigma"] = 963,  
4473 ["tau"] = 964,  
4474 ["upsilon"] = 965,  
4475 ["upsilon"] = 965,  
4476 ["phi"] = 966,  
4477 ["chi"] = 967,  
4478 ["psi"] = 968,  
4479 ["omega"] = 969,  
4480 ["thetasym"] = 977,  
4481 ["thetav"] = 977,  
4482 ["vartheta"] = 977,  
4483 ["Upsilon"] = 978,  
4484 ["upsih"] = 978,  
4485 ["phiv"] = 981,  
4486 ["straightphi"] = 981,  
4487 ["varphi"] = 981,  
4488 ["piv"] = 982,  
4489 ["varpi"] = 982,  
4490 ["Gammad"] = 988,  
4491 ["digamma"] = 989,  
4492 ["gammad"] = 989,  
4493 ["kappav"] = 1008,  
4494 ["varkappa"] = 1008,  
4495 ["rhov"] = 1009,  
4496 ["varrho"] = 1009,  
4497 ["epsiv"] = 1013,  
4498 ["straightepsilon"] = 1013,  
4499 ["varepsilon"] = 1013,  
4500 ["backepsilon"] = 1014,  
4501 ["bepsi"] = 1014,  
4502 ["IOcy"] = 1025,  
4503 ["DJcy"] = 1026,  
4504 ["GJcy"] = 1027,  
4505 ["Jukcy"] = 1028,  
4506 ["DScy"] = 1029,  
4507 ["Iukcy"] = 1030,  
4508 ["YIcy"] = 1031,  
4509 ["Jsercy"] = 1032,  
4510 ["LJcy"] = 1033,  
4511 ["NJcy"] = 1034,

4512 ["TSHcy"] = 1035,  
4513 ["KJcy"] = 1036,  
4514 ["Ubrcy"] = 1038,  
4515 ["DZcy"] = 1039,  
4516 ["Acy"] = 1040,  
4517 ["Bcy"] = 1041,  
4518 ["Vcy"] = 1042,  
4519 ["Gcy"] = 1043,  
4520 ["Dcy"] = 1044,  
4521 ["IEcy"] = 1045,  
4522 ["ZHcy"] = 1046,  
4523 ["Zcy"] = 1047,  
4524 ["Icy"] = 1048,  
4525 ["Jcy"] = 1049,  
4526 ["Kcy"] = 1050,  
4527 ["Lcy"] = 1051,  
4528 ["Mcy"] = 1052,  
4529 ["Ncy"] = 1053,  
4530 ["Ocy"] = 1054,  
4531 ["Pcy"] = 1055,  
4532 ["Rcy"] = 1056,  
4533 ["Scy"] = 1057,  
4534 ["Tcy"] = 1058,  
4535 ["Ucy"] = 1059,  
4536 ["Fcy"] = 1060,  
4537 ["KHcy"] = 1061,  
4538 ["TScy"] = 1062,  
4539 ["CHcy"] = 1063,  
4540 ["SHcy"] = 1064,  
4541 ["SHCHcy"] = 1065,  
4542 ["HARDcy"] = 1066,  
4543 ["Ycy"] = 1067,  
4544 ["SOFTcy"] = 1068,  
4545 ["Ecy"] = 1069,  
4546 ["YUcy"] = 1070,  
4547 ["YAcy"] = 1071,  
4548 ["acy"] = 1072,  
4549 ["bcy"] = 1073,  
4550 ["vcy"] = 1074,  
4551 ["gcy"] = 1075,  
4552 ["dcy"] = 1076,  
4553 ["iecy"] = 1077,  
4554 ["zhcy"] = 1078,  
4555 ["zcy"] = 1079,  
4556 ["icy"] = 1080,  
4557 ["jcy"] = 1081,  
4558 ["kcy"] = 1082,

4559 ["lcy"] = 1083,  
4560 ["mcy"] = 1084,  
4561 ["ncy"] = 1085,  
4562 ["ocy"] = 1086,  
4563 ["pcy"] = 1087,  
4564 ["rcy"] = 1088,  
4565 ["scy"] = 1089,  
4566 ["tcy"] = 1090,  
4567 ["ucy"] = 1091,  
4568 ["fcy"] = 1092,  
4569 ["khcy"] = 1093,  
4570 ["tscy"] = 1094,  
4571 ["chcy"] = 1095,  
4572 ["shcy"] = 1096,  
4573 ["shchcy"] = 1097,  
4574 ["hardcy"] = 1098,  
4575 ["ycy"] = 1099,  
4576 ["softcy"] = 1100,  
4577 ["ecy"] = 1101,  
4578 ["yucy"] = 1102,  
4579 ["yacy"] = 1103,  
4580 ["iocy"] = 1105,  
4581 ["djcy"] = 1106,  
4582 ["gjcy"] = 1107,  
4583 ["jukcy"] = 1108,  
4584 ["dscy"] = 1109,  
4585 ["iukcy"] = 1110,  
4586 ["yicy"] = 1111,  
4587 ["jsercy"] = 1112,  
4588 ["ljcy"] = 1113,  
4589 ["njcy"] = 1114,  
4590 ["tshcy"] = 1115,  
4591 ["kjcy"] = 1116,  
4592 ["ubrky"] = 1118,  
4593 ["dzcy"] = 1119,  
4594 ["ensp"] = 8194,  
4595 ["emsp"] = 8195,  
4596 ["emsp13"] = 8196,  
4597 ["emsp14"] = 8197,  
4598 ["numsp"] = 8199,  
4599 ["puncsp"] = 8200,  
4600 ["ThinSpace"] = 8201,  
4601 ["thinsp"] = 8201,  
4602 ["VeryThinSpace"] = 8202,  
4603 ["hairsp"] = 8202,  
4604 ["NegativeMediumSpace"] = 8203,  
4605 ["NegativeThickSpace"] = 8203,

4606 ["NegativeThinSpace"] = 8203,  
4607 ["NegativeVeryThinSpace"] = 8203,  
4608 ["ZeroWidthSpace"] = 8203,  
4609 ["zwnj"] = 8204,  
4610 ["zwj"] = 8205,  
4611 ["lrm"] = 8206,  
4612 ["rlm"] = 8207,  
4613 ["dash"] = 8208,  
4614 ["hyphen"] = 8208,  
4615 ["ndash"] = 8211,  
4616 ["mdash"] = 8212,  
4617 ["horbar"] = 8213,  
4618 ["Verbar"] = 8214,  
4619 ["Vert"] = 8214,  
4620 ["OpenCurlyQuote"] = 8216,  
4621 ["lsquo"] = 8216,  
4622 ["CloseCurlyQuote"] = 8217,  
4623 ["rsquo"] = 8217,  
4624 ["rsquor"] = 8217,  
4625 ["lsquor"] = 8218,  
4626 ["sbquo"] = 8218,  
4627 ["OpenCurlyDoubleQuote"] = 8220,  
4628 ["ldquo"] = 8220,  
4629 ["CloseCurlyDoubleQuote"] = 8221,  
4630 ["rdquo"] = 8221,  
4631 ["rdquor"] = 8221,  
4632 ["bdquo"] = 8222,  
4633 ["ldquor"] = 8222,  
4634 ["dagger"] = 8224,  
4635 ["Dagger"] = 8225,  
4636 ["ddagger"] = 8225,  
4637 ["bull"] = 8226,  
4638 ["bullet"] = 8226,  
4639 ["nldr"] = 8229,  
4640 ["hellip"] = 8230,  
4641 ["mldr"] = 8230,  
4642 ["permil"] = 8240,  
4643 ["pertenk"] = 8241,  
4644 ["prime"] = 8242,  
4645 ["Prime"] = 8243,  
4646 ["tprime"] = 8244,  
4647 ["backprime"] = 8245,  
4648 ["bprime"] = 8245,  
4649 ["lsaquo"] = 8249,  
4650 ["rsaquo"] = 8250,  
4651 ["OverBar"] = 8254,  
4652 ["oline"] = 8254,

4653 ["caret"] = 8257,  
 4654 ["hybull"] = 8259,  
 4655 ["frasl"] = 8260,  
 4656 ["bsemi"] = 8271,  
 4657 ["qprime"] = 8279,  
 4658 ["MediumSpace"] = 8287,  
 4659 ["ThickSpace"] = {8287, 8202},  
 4660 ["NoBreak"] = 8288,  
 4661 ["ApplyFunction"] = 8289,  
 4662 ["af"] = 8289,  
 4663 ["InvisibleTimes"] = 8290,  
 4664 ["it"] = 8290,  
 4665 ["InvisibleComma"] = 8291,  
 4666 ["ic"] = 8291,  
 4667 ["euro"] = 8364,  
 4668 ["TripleDot"] = 8411,  
 4669 ["tdot"] = 8411,  
 4670 ["DotDot"] = 8412,  
 4671 ["Copf"] = 8450,  
 4672 ["complexes"] = 8450,  
 4673 ["incare"] = 8453,  
 4674 ["gscr"] = 8458,  
 4675 ["HilbertSpace"] = 8459,  
 4676 ["Hscr"] = 8459,  
 4677 ["hamilt"] = 8459,  
 4678 ["Hfr"] = 8460,  
 4679 ["Poincareplane"] = 8460,  
 4680 ["Hopf"] = 8461,  
 4681 ["quaternions"] = 8461,  
 4682 ["planckh"] = 8462,  
 4683 ["hbar"] = 8463,  
 4684 ["hslash"] = 8463,  
 4685 ["planck"] = 8463,  
 4686 ["plankv"] = 8463,  
 4687 ["Iscr"] = 8464,  
 4688 ["imagline"] = 8464,  
 4689 ["Ifr"] = 8465,  
 4690 ["Im"] = 8465,  
 4691 ["image"] = 8465,  
 4692 ["imagpart"] = 8465,  
 4693 ["Laplacetrif"] = 8466,  
 4694 ["Lscr"] = 8466,  
 4695 ["lagran"] = 8466,  
 4696 ["ell"] = 8467,  
 4697 ["Nopf"] = 8469,  
 4698 ["naturals"] = 8469,  
 4699 ["numero"] = 8470,



4700 ["copysr"] = 8471,  
4701 ["weierp"] = 8472,  
4702 ["wp"] = 8472,  
4703 ["Popf"] = 8473,  
4704 ["primes"] = 8473,  
4705 ["Qopf"] = 8474,  
4706 ["rationals"] = 8474,  
4707 ["Rscr"] = 8475,  
4708 ["realine"] = 8475,  
4709 ["Re"] = 8476,  
4710 ["Rfr"] = 8476,  
4711 ["real"] = 8476,  
4712 ["realpart"] = 8476,  
4713 ["Ropf"] = 8477,  
4714 ["reals"] = 8477,  
4715 ["rx"] = 8478,  
4716 ["TRADE"] = 8482,  
4717 ["trade"] = 8482,  
4718 ["Zopf"] = 8484,  
4719 ["integers"] = 8484,  
4720 ["mho"] = 8487,  
4721 ["Zfr"] = 8488,  
4722 ["zeetrf"] = 8488,  
4723 ["iiota"] = 8489,  
4724 ["Bernoullis"] = 8492,  
4725 ["Bscr"] = 8492,  
4726 ["bernou"] = 8492,  
4727 ["Cayleys"] = 8493,  
4728 ["Cfr"] = 8493,  
4729 ["escr"] = 8495,  
4730 ["Escr"] = 8496,  
4731 ["expectation"] = 8496,  
4732 ["Fouriertrf"] = 8497,  
4733 ["Fscr"] = 8497,  
4734 ["Mellintrf"] = 8499,  
4735 ["Mscr"] = 8499,  
4736 ["phmmat"] = 8499,  
4737 ["order"] = 8500,  
4738 ["orderof"] = 8500,  
4739 ["oscr"] = 8500,  
4740 ["alefsym"] = 8501,  
4741 ["aleph"] = 8501,  
4742 ["beth"] = 8502,  
4743 ["gimel"] = 8503,  
4744 ["daleth"] = 8504,  
4745 ["CapitalDifferentialD"] = 8517,  
4746 ["DD"] = 8517,

4747 ["DifferentialD"] = 8518,  
 4748 ["dd"] = 8518,  
 4749 ["ExponentialE"] = 8519,  
 4750 ["ee"] = 8519,  
 4751 ["exponentiale"] = 8519,  
 4752 ["ImaginaryI"] = 8520,  
 4753 ["ii"] = 8520,  
 4754 ["frac13"] = 8531,  
 4755 ["frac23"] = 8532,  
 4756 ["frac15"] = 8533,  
 4757 ["frac25"] = 8534,  
 4758 ["frac35"] = 8535,  
 4759 ["frac45"] = 8536,  
 4760 ["frac16"] = 8537,  
 4761 ["frac56"] = 8538,  
 4762 ["frac18"] = 8539,  
 4763 ["frac38"] = 8540,  
 4764 ["frac58"] = 8541,  
 4765 ["frac78"] = 8542,  
 4766 ["LeftArrow"] = 8592,  
 4767 ["ShortLeftArrow"] = 8592,  
 4768 ["larr"] = 8592,  
 4769 ["leftarrow"] = 8592,  
 4770 ["slarr"] = 8592,  
 4771 ["ShortUpArrow"] = 8593,  
 4772 ["UpArrow"] = 8593,  
 4773 ["uarr"] = 8593,  
 4774 ["uparrow"] = 8593,  
 4775 ["RightArrow"] = 8594,  
 4776 ["ShortRightArrow"] = 8594,  
 4777 ["rarr"] = 8594,  
 4778 ["rightarrow"] = 8594,  
 4779 ["srarr"] = 8594,  
 4780 ["DownArrow"] = 8595,  
 4781 ["ShortDownArrow"] = 8595,  
 4782 ["darr"] = 8595,  
 4783 ["downarrow"] = 8595,  
 4784 ["LeftRightArrow"] = 8596,  
 4785 ["harr"] = 8596,  
 4786 ["leftrightarrow"] = 8596,  
 4787 ["UpDownArrow"] = 8597,  
 4788 ["updownarrow"] = 8597,  
 4789 ["varr"] = 8597,  
 4790 ["UpperLeftArrow"] = 8598,  
 4791 ["nwarr"] = 8598,  
 4792 ["nwarrow"] = 8598,  
 4793 ["UpperRightArrow"] = 8599,

4794 ["nearr"] = 8599,  
 4795 ["nearrow"] = 8599,  
 4796 ["LowerRightArrow"] = 8600,  
 4797 ["searr"] = 8600,  
 4798 ["searrow"] = 8600,  
 4799 ["LowerLeftArrow"] = 8601,  
 4800 ["swarr"] = 8601,  
 4801 ["swarrow"] = 8601,  
 4802 ["nlarr"] = 8602,  
 4803 ["nleftarrow"] = 8602,  
 4804 ["nrarr"] = 8603,  
 4805 ["nrightarrow"] = 8603,  
 4806 ["nrarrw"] = {8605, 824},  
 4807 ["rarrw"] = 8605,  
 4808 ["rightsquigarrow"] = 8605,  
 4809 ["Larr"] = 8606,  
 4810 ["twoheadleftarrow"] = 8606,  
 4811 ["Uarr"] = 8607,  
 4812 ["Rarr"] = 8608,  
 4813 ["twoheadrightarrow"] = 8608,  
 4814 ["Darr"] = 8609,  
 4815 ["larrtl"] = 8610,  
 4816 ["leftarrowtail"] = 8610,  
 4817 ["rarrtl"] = 8611,  
 4818 ["rightarrowtail"] = 8611,  
 4819 ["LeftTeeArrow"] = 8612,  
 4820 ["mapstoleft"] = 8612,  
 4821 ["UpTeeArrow"] = 8613,  
 4822 ["mapstoup"] = 8613,  
 4823 ["RightTeeArrow"] = 8614,  
 4824 ["map"] = 8614,  
 4825 ["mapsto"] = 8614,  
 4826 ["DownTeeArrow"] = 8615,  
 4827 ["mapstodown"] = 8615,  
 4828 ["hookleftarrow"] = 8617,  
 4829 ["larrhk"] = 8617,  
 4830 ["hookrightarrow"] = 8618,  
 4831 ["rarrhk"] = 8618,  
 4832 ["larrlp"] = 8619,  
 4833 ["looparrowleft"] = 8619,  
 4834 ["looparrowright"] = 8620,  
 4835 ["rarrlp"] = 8620,  
 4836 ["harrw"] = 8621,  
 4837 ["leftrightsquigarrow"] = 8621,  
 4838 ["nharr"] = 8622,  
 4839 ["nleftrightarrow"] = 8622,  
 4840 ["Lsh"] = 8624,

4841 ["lsh"] = 8624,  
 4842 ["Rsh"] = 8625,  
 4843 ["rsh"] = 8625,  
 4844 ["ldsh"] = 8626,  
 4845 ["rdsh"] = 8627,  
 4846 ["crarr"] = 8629,  
 4847 ["cularr"] = 8630,  
 4848 ["curvearrowleft"] = 8630,  
 4849 ["curarr"] = 8631,  
 4850 ["curvearrowright"] = 8631,  
 4851 ["circlearrowleft"] = 8634,  
 4852 ["olarr"] = 8634,  
 4853 ["circlearrowright"] = 8635,  
 4854 ["orarr"] = 8635,  
 4855 ["LeftVector"] = 8636,  
 4856 ["leftharpoonup"] = 8636,  
 4857 ["lharu"] = 8636,  
 4858 ["DownLeftVector"] = 8637,  
 4859 ["leftharpoondown"] = 8637,  
 4860 ["lhard"] = 8637,  
 4861 ["RightUpVector"] = 8638,  
 4862 ["uharr"] = 8638,  
 4863 ["upharpoonright"] = 8638,  
 4864 ["LeftUpVector"] = 8639,  
 4865 ["uharl"] = 8639,  
 4866 ["upharpoonleft"] = 8639,  
 4867 ["RightVector"] = 8640,  
 4868 ["rharu"] = 8640,  
 4869 ["rightharpoonup"] = 8640,  
 4870 ["DownRightVector"] = 8641,  
 4871 ["rhard"] = 8641,  
 4872 ["rightharpoondown"] = 8641,  
 4873 ["RightDownVector"] = 8642,  
 4874 ["dharr"] = 8642,  
 4875 ["downharpoonright"] = 8642,  
 4876 ["LeftDownVector"] = 8643,  
 4877 ["dharl"] = 8643,  
 4878 ["downharpoonleft"] = 8643,  
 4879 ["RightArrowLeftArrow"] = 8644,  
 4880 ["rightleftarrows"] = 8644,  
 4881 ["rlarr"] = 8644,  
 4882 ["UpArrowDownArrow"] = 8645,  
 4883 ["udarr"] = 8645,  
 4884 ["LeftArrowRightArrow"] = 8646,  
 4885 ["leftrightarrows"] = 8646,  
 4886 ["lrarr"] = 8646,  
 4887 ["leftleftarrows"] = 8647,

4888 ["llarr"] = 8647,  
 4889 ["uparrows"] = 8648,  
 4890 ["uuarr"] = 8648,  
 4891 ["rightrightarrows"] = 8649,  
 4892 ["rrarr"] = 8649,  
 4893 ["ddarr"] = 8650,  
 4894 ["downdownarrows"] = 8650,  
 4895 ["ReverseEquilibrium"] = 8651,  
 4896 ["leftrightharpoons"] = 8651,  
 4897 ["lrhar"] = 8651,  
 4898 ["Equilibrium"] = 8652,  
 4899 ["rightleftharpoons"] = 8652,  
 4900 ["rlhar"] = 8652,  
 4901 ["nLeftarrow"] = 8653,  
 4902 ["nlArr"] = 8653,  
 4903 ["nLeftrightarrow"] = 8654,  
 4904 ["nhArr"] = 8654,  
 4905 ["nRightarrow"] = 8655,  
 4906 ["nrArr"] = 8655,  
 4907 ["DoubleLeftArrow"] = 8656,  
 4908 ["Leftarrow"] = 8656,  
 4909 ["lArr"] = 8656,  
 4910 ["DoubleUpArrow"] = 8657,  
 4911 ["Uparrow"] = 8657,  
 4912 ["uArr"] = 8657,  
 4913 ["DoubleRightArrow"] = 8658,  
 4914 ["Implies"] = 8658,  
 4915 ["Rightarrow"] = 8658,  
 4916 ["rArr"] = 8658,  
 4917 ["DoubleDownArrow"] = 8659,  
 4918 ["Downarrow"] = 8659,  
 4919 ["dArr"] = 8659,  
 4920 ["DoubleLeftRightArrow"] = 8660,  
 4921 ["Leftrightarrow"] = 8660,  
 4922 ["hArr"] = 8660,  
 4923 ["iff"] = 8660,  
 4924 ["DoubleUpDownArrow"] = 8661,  
 4925 ["Updownarrow"] = 8661,  
 4926 ["vArr"] = 8661,  
 4927 ["nwArr"] = 8662,  
 4928 ["neArr"] = 8663,  
 4929 ["seArr"] = 8664,  
 4930 ["swArr"] = 8665,  
 4931 ["Lleftarrow"] = 8666,  
 4932 ["lAarr"] = 8666,  
 4933 ["Rrightarrow"] = 8667,  
 4934 ["rAarr"] = 8667,

4935 ["zigrarr"] = 8669,  
 4936 ["LeftArrowBar"] = 8676,  
 4937 ["larrb"] = 8676,  
 4938 ["RightArrowBar"] = 8677,  
 4939 ["rarrb"] = 8677,  
 4940 ["DownArrowUpArrow"] = 8693,  
 4941 ["duarr"] = 8693,  
 4942 ["loarr"] = 8701,  
 4943 ["roarr"] = 8702,  
 4944 ["hoarr"] = 8703,  
 4945 ["ForAll"] = 8704,  
 4946 ["forall"] = 8704,  
 4947 ["comp"] = 8705,  
 4948 ["complement"] = 8705,  
 4949 ["PartialD"] = 8706,  
 4950 ["npart"] = {8706, 824},  
 4951 ["part"] = 8706,  
 4952 ["Exists"] = 8707,  
 4953 ["exist"] = 8707,  
 4954 ["NotExists"] = 8708,  
 4955 ["nexist"] = 8708,  
 4956 ["nexists"] = 8708,  
 4957 ["empty"] = 8709,  
 4958 ["emptyset"] = 8709,  
 4959 ["emptyv"] = 8709,  
 4960 ["varnothing"] = 8709,  
 4961 ["Del"] = 8711,  
 4962 ["nabla"] = 8711,  
 4963 ["Element"] = 8712,  
 4964 ["in"] = 8712,  
 4965 ["isin"] = 8712,  
 4966 ["isinv"] = 8712,  
 4967 ["NotElement"] = 8713,  
 4968 ["notin"] = 8713,  
 4969 ["notinva"] = 8713,  
 4970 ["ReverseElement"] = 8715,  
 4971 ["SuchThat"] = 8715,  
 4972 ["ni"] = 8715,  
 4973 ["niv"] = 8715,  
 4974 ["NotReverseElement"] = 8716,  
 4975 ["notni"] = 8716,  
 4976 ["notniva"] = 8716,  
 4977 ["Product"] = 8719,  
 4978 ["prod"] = 8719,  
 4979 ["Coproduct"] = 8720,  
 4980 ["coprod"] = 8720,  
 4981 ["Sum"] = 8721,

```

4982 ["sum"] = 8721,
4983 ["minus"] = 8722,
4984 ["MinusPlus"] = 8723,
4985 ["mnplus"] = 8723,
4986 ["mp"] = 8723,
4987 ["dotplus"] = 8724,
4988 ["plusdo"] = 8724,
4989 ["Backslash"] = 8726,
4990 ["setminus"] = 8726,
4991 ["setmn"] = 8726,
4992 ["smallsetminus"] = 8726,
4993 ["ssetmn"] = 8726,
4994 ["lowast"] = 8727,
4995 ["SmallCircle"] = 8728,
4996 ["compfn"] = 8728,
4997 ["Sqrt"] = 8730,
4998 ["radic"] = 8730,
4999 ["Proportional"] = 8733,
5000 ["prop"] = 8733,
5001 ["propto"] = 8733,
5002 ["varpropto"] = 8733,
5003 ["vprop"] = 8733,
5004 ["infin"] = 8734,
5005 ["angrt"] = 8735,
5006 ["ang"] = 8736,
5007 ["angle"] = 8736,
5008 ["nang"] = {8736, 8402},
5009 ["angmsd"] = 8737,
5010 ["measuredangle"] = 8737,
5011 ["angsph"] = 8738,
5012 ["VerticalBar"] = 8739,
5013 ["mid"] = 8739,
5014 ["shortmid"] = 8739,
5015 ["smid"] = 8739,
5016 ["NotVerticalBar"] = 8740,
5017 ["nmid"] = 8740,
5018 ["nshortmid"] = 8740,
5019 ["nsmid"] = 8740,
5020 ["DoubleVerticalBar"] = 8741,
5021 ["par"] = 8741,
5022 ["parallel"] = 8741,
5023 ["shortparallel"] = 8741,
5024 ["spar"] = 8741,
5025 ["NotDoubleVerticalBar"] = 8742,
5026 ["npar"] = 8742,
5027 ["nparallel"] = 8742,
5028 ["nshortparallel"] = 8742,

```

5029 ["nspar"] = 8742,  
5030 ["and"] = 8743,  
5031 ["wedge"] = 8743,  
5032 ["or"] = 8744,  
5033 ["vee"] = 8744,  
5034 ["cap"] = 8745,  
5035 ["caps"] = {8745, 65024},  
5036 ["cup"] = 8746,  
5037 ["cups"] = {8746, 65024},  
5038 ["Integral"] = 8747,  
5039 ["int"] = 8747,  
5040 ["Int"] = 8748,  
5041 ["iiint"] = 8749,  
5042 ["tint"] = 8749,  
5043 ["ContourIntegral"] = 8750,  
5044 ["conint"] = 8750,  
5045 ["oint"] = 8750,  
5046 ["Conint"] = 8751,  
5047 ["DoubleContourIntegral"] = 8751,  
5048 ["Cconint"] = 8752,  
5049 ["cwint"] = 8753,  
5050 ["ClockwiseContourIntegral"] = 8754,  
5051 ["cwconint"] = 8754,  
5052 ["CounterClockwiseContourIntegral"] = 8755,  
5053 ["awconint"] = 8755,  
5054 ["Therefore"] = 8756,  
5055 ["there4"] = 8756,  
5056 ["therefore"] = 8756,  
5057 ["Because"] = 8757,  
5058 ["because"] = 8757,  
5059 ["because"] = 8757,  
5060 ["ratio"] = 8758,  
5061 ["Colon"] = 8759,  
5062 ["Proportion"] = 8759,  
5063 ["dotminus"] = 8760,  
5064 ["minusd"] = 8760,  
5065 ["mDDot"] = 8762,  
5066 ["homtht"] = 8763,  
5067 ["Tilde"] = 8764,  
5068 ["nvsim"] = {8764, 8402},  
5069 ["sim"] = 8764,  
5070 ["thicksim"] = 8764,  
5071 ["thksim"] = 8764,  
5072 ["backsim"] = 8765,  
5073 ["bsim"] = 8765,  
5074 ["race"] = {8765, 817},  
5075 ["ac"] = 8766,



5076 ["acE"] = {8766, 819},  
5077 ["mstpos"] = 8766,  
5078 ["acd"] = 8767,  
5079 ["VerticalTilde"] = 8768,  
5080 ["wr"] = 8768,  
5081 ["wreath"] = 8768,  
5082 ["NotTilde"] = 8769,  
5083 ["nsim"] = 8769,  
5084 ["EqualTilde"] = 8770,  
5085 ["NotEqualTilde"] = {8770, 824},  
5086 ["eqsim"] = 8770,  
5087 ["esim"] = 8770,  
5088 ["nesim"] = {8770, 824},  
5089 ["TildeEqual"] = 8771,  
5090 ["sime"] = 8771,  
5091 ["simeq"] = 8771,  
5092 ["NotTildeEqual"] = 8772,  
5093 ["nsime"] = 8772,  
5094 ["nsimeq"] = 8772,  
5095 ["TildeFullEqual"] = 8773,  
5096 ["cong"] = 8773,  
5097 ["simne"] = 8774,  
5098 ["NotTildeFullEqual"] = 8775,  
5099 ["ncong"] = 8775,  
5100 ["TildeTilde"] = 8776,  
5101 ["ap"] = 8776,  
5102 ["approx"] = 8776,  
5103 ["asymp"] = 8776,  
5104 ["thickapprox"] = 8776,  
5105 ["thkap"] = 8776,  
5106 ["NotTildeTilde"] = 8777,  
5107 ["nap"] = 8777,  
5108 ["napprox"] = 8777,  
5109 ["ape"] = 8778,  
5110 ["approxpeq"] = 8778,  
5111 ["apid"] = 8779,  
5112 ["napid"] = {8779, 824},  
5113 ["backcong"] = 8780,  
5114 ["bcong"] = 8780,  
5115 ["CupCap"] = 8781,  
5116 ["asympeq"] = 8781,  
5117 ["nvap"] = {8781, 8402},  
5118 ["Bumpeq"] = 8782,  
5119 ["HumpDownHump"] = 8782,  
5120 ["NotHumpDownHump"] = {8782, 824},  
5121 ["bump"] = 8782,  
5122 ["nbump"] = {8782, 824},

5123 ["HumpEqual"] = 8783,  
5124 ["NotHumpEqual"] = {8783, 824},  
5125 ["bumpe"] = 8783,  
5126 ["bumpeq"] = 8783,  
5127 ["nbumpe"] = {8783, 824},  
5128 ["DotEqual"] = 8784,  
5129 ["doteq"] = 8784,  
5130 ["esdot"] = 8784,  
5131 ["nedot"] = {8784, 824},  
5132 ["doteqdot"] = 8785,  
5133 ["eDot"] = 8785,  
5134 ["efDot"] = 8786,  
5135 ["fallingdotseq"] = 8786,  
5136 ["erDot"] = 8787,  
5137 ["risingdotseq"] = 8787,  
5138 ["Assign"] = 8788,  
5139 ["colone"] = 8788,  
5140 ["coloneq"] = 8788,  
5141 ["ecolon"] = 8789,  
5142 ["eqcolon"] = 8789,  
5143 ["ecir"] = 8790,  
5144 ["eqcirc"] = 8790,  
5145 ["circeq"] = 8791,  
5146 ["cire"] = 8791,  
5147 ["wedgeq"] = 8793,  
5148 ["veeeq"] = 8794,  
5149 ["triangleq"] = 8796,  
5150 ["trie"] = 8796,  
5151 ["equest"] = 8799,  
5152 ["questeq"] = 8799,  
5153 ["NotEqual"] = 8800,  
5154 ["ne"] = 8800,  
5155 ["Congruent"] = 8801,  
5156 ["bnequiv"] = {8801, 8421},  
5157 ["equiv"] = 8801,  
5158 ["NotCongruent"] = 8802,  
5159 ["nequiv"] = 8802,  
5160 ["le"] = 8804,  
5161 ["leq"] = 8804,  
5162 ["nvle"] = {8804, 8402},  
5163 ["GreaterEqual"] = 8805,  
5164 ["ge"] = 8805,  
5165 ["geq"] = 8805,  
5166 ["nvge"] = {8805, 8402},  
5167 ["LessFullEqual"] = 8806,  
5168 ["lE"] = 8806,  
5169 ["leqq"] = 8806,

5170 ["nlE"] = {8806, 824},  
5171 ["nleqq"] = {8806, 824},  
5172 ["GreaterFullEqual"] = 8807,  
5173 ["NotGreaterFullEqual"] = {8807, 824},  
5174 ["gE"] = 8807,  
5175 ["geqq"] = 8807,  
5176 ["ngE"] = {8807, 824},  
5177 ["ngeqq"] = {8807, 824},  
5178 ["lnE"] = 8808,  
5179 ["lneqq"] = 8808,  
5180 ["lvertneqq"] = {8808, 65024},  
5181 ["lvnE"] = {8808, 65024},  
5182 ["gnE"] = 8809,  
5183 ["gneqq"] = 8809,  
5184 ["gvertneqq"] = {8809, 65024},  
5185 ["gvnE"] = {8809, 65024},  
5186 ["Lt"] = 8810,  
5187 ["NestedLessLess"] = 8810,  
5188 ["NotLessLess"] = {8810, 824},  
5189 ["ll"] = 8810,  
5190 ["nLt"] = {8810, 8402},  
5191 ["nLtv"] = {8810, 824},  
5192 ["Gt"] = 8811,  
5193 ["NestedGreaterGreater"] = 8811,  
5194 ["NotGreaterGreater"] = {8811, 824},  
5195 ["gg"] = 8811,  
5196 ["nGt"] = {8811, 8402},  
5197 ["nGtv"] = {8811, 824},  
5198 ["between"] = 8812,  
5199 ["twixt"] = 8812,  
5200 ["NotCupCap"] = 8813,  
5201 ["NotLess"] = 8814,  
5202 ["nless"] = 8814,  
5203 ["nlt"] = 8814,  
5204 ["NotGreater"] = 8815,  
5205 ["ngt"] = 8815,  
5206 ["ngtr"] = 8815,  
5207 ["NotLessEqual"] = 8816,  
5208 ["nle"] = 8816,  
5209 ["nleq"] = 8816,  
5210 ["NotGreaterEqual"] = 8817,  
5211 ["nge"] = 8817,  
5212 ["ngeq"] = 8817,  
5213 ["LessTilde"] = 8818,  
5214 ["lesssim"] = 8818,  
5215 ["lsim"] = 8818,  
5216 ["GreaterTilde"] = 8819,

5217 ["gsim"] = 8819,  
5218 ["gtrsim"] = 8819,  
5219 ["NotLessTilde"] = 8820,  
5220 ["nlsim"] = 8820,  
5221 ["NotGreaterTilde"] = 8821,  
5222 ["ngsim"] = 8821,  
5223 ["LessGreater"] = 8822,  
5224 ["lessgtr"] = 8822,  
5225 ["lg"] = 8822,  
5226 ["GreaterLess"] = 8823,  
5227 ["gl"] = 8823,  
5228 ["gtrless"] = 8823,  
5229 ["NotLessGreater"] = 8824,  
5230 ["ntlg"] = 8824,  
5231 ["NotGreaterLess"] = 8825,  
5232 ["ntgl"] = 8825,  
5233 ["Precedes"] = 8826,  
5234 ["pr"] = 8826,  
5235 ["prec"] = 8826,  
5236 ["Succeeds"] = 8827,  
5237 ["sc"] = 8827,  
5238 ["succ"] = 8827,  
5239 ["PrecedesSlantEqual"] = 8828,  
5240 ["prcue"] = 8828,  
5241 ["preccurlyeq"] = 8828,  
5242 ["SucceedsSlantEqual"] = 8829,  
5243 ["sccue"] = 8829,  
5244 ["succcurlyeq"] = 8829,  
5245 ["PrecedesTilde"] = 8830,  
5246 ["precsim"] = 8830,  
5247 ["prsim"] = 8830,  
5248 ["NotSucceedsTilde"] = {8831, 824},  
5249 ["SucceedsTilde"] = 8831,  
5250 ["scsim"] = 8831,  
5251 ["succsim"] = 8831,  
5252 ["NotPrecedes"] = 8832,  
5253 ["npr"] = 8832,  
5254 ["nprec"] = 8832,  
5255 ["NotSucceeds"] = 8833,  
5256 ["nsc"] = 8833,  
5257 ["nsucc"] = 8833,  
5258 ["NotSubset"] = {8834, 8402},  
5259 ["nsubset"] = {8834, 8402},  
5260 ["sub"] = 8834,  
5261 ["subset"] = 8834,  
5262 ["vnsup"] = {8834, 8402},  
5263 ["NotSuperset"] = {8835, 8402},

5264 ["Superset"] = 8835,  
5265 ["nsupset"] = {8835, 8402},  
5266 ["sup"] = 8835,  
5267 ["supset"] = 8835,  
5268 ["vnsup"] = {8835, 8402},  
5269 ["nsub"] = 8836,  
5270 ["nsup"] = 8837,  
5271 ["SubsetEqual"] = 8838,  
5272 ["sube"] = 8838,  
5273 ["subseteq"] = 8838,  
5274 ["SupersetEqual"] = 8839,  
5275 ["supe"] = 8839,  
5276 ["supseteq"] = 8839,  
5277 ["NotSubsetEqual"] = 8840,  
5278 ["nsube"] = 8840,  
5279 ["nsubseteq"] = 8840,  
5280 ["NotSupersetEqual"] = 8841,  
5281 ["nsupe"] = 8841,  
5282 ["nsupseteq"] = 8841,  
5283 ["subne"] = 8842,  
5284 ["subsetneq"] = 8842,  
5285 ["varsubsetneq"] = {8842, 65024},  
5286 ["vsubne"] = {8842, 65024},  
5287 ["supne"] = 8843,  
5288 ["supsetneq"] = 8843,  
5289 ["varsupsetneq"] = {8843, 65024},  
5290 ["vsupne"] = {8843, 65024},  
5291 ["cupdot"] = 8845,  
5292 ["UnionPlus"] = 8846,  
5293 ["uplus"] = 8846,  
5294 ["NotSquareSubset"] = {8847, 824},  
5295 ["SquareSubset"] = 8847,  
5296 ["sqsub"] = 8847,  
5297 ["sqsubset"] = 8847,  
5298 ["NotSquareSuperset"] = {8848, 824},  
5299 ["SquareSuperset"] = 8848,  
5300 ["sqsup"] = 8848,  
5301 ["sqsupset"] = 8848,  
5302 ["SquareSubsetEqual"] = 8849,  
5303 ["sqsube"] = 8849,  
5304 ["sqsubseteq"] = 8849,  
5305 ["SquareSupersetEqual"] = 8850,  
5306 ["sqsupe"] = 8850,  
5307 ["sqsupseteq"] = 8850,  
5308 ["SquareIntersection"] = 8851,  
5309 ["sqcap"] = 8851,  
5310 ["sqcaps"] = {8851, 65024},

```

5311 ["SquareUnion"] = 8852,
5312 ["sqcup"] = 8852,
5313 ["sqcups"] = {8852, 65024},
5314 ["CirclePlus"] = 8853,
5315 ["oplus"] = 8853,
5316 ["CircleMinus"] = 8854,
5317 ["ominus"] = 8854,
5318 ["CircleTimes"] = 8855,
5319 ["otimes"] = 8855,
5320 ["osol"] = 8856,
5321 ["CircleDot"] = 8857,
5322 ["odot"] = 8857,
5323 ["circledcirc"] = 8858,
5324 ["ocir"] = 8858,
5325 ["circledast"] = 8859,
5326 ["oast"] = 8859,
5327 ["circleddash"] = 8861,
5328 ["odash"] = 8861,
5329 ["boxplus"] = 8862,
5330 ["plusb"] = 8862,
5331 ["boxminus"] = 8863,
5332 ["minusb"] = 8863,
5333 ["boxtimes"] = 8864,
5334 ["timesb"] = 8864,
5335 ["dotsquare"] = 8865,
5336 ["sdotb"] = 8865,
5337 ["RightTee"] = 8866,
5338 ["vdash"] = 8866,
5339 ["LeftTee"] = 8867,
5340 ["dashv"] = 8867,
5341 ["DownTee"] = 8868,
5342 ["top"] = 8868,
5343 ["UpTee"] = 8869,
5344 ["bot"] = 8869,
5345 ["bottom"] = 8869,
5346 ["perp"] = 8869,
5347 ["models"] = 8871,
5348 ["DoubleRightTee"] = 8872,
5349 ["vDash"] = 8872,
5350 ["VDash"] = 8873,
5351 ["Vvdash"] = 8874,
5352 ["VDash"] = 8875,
5353 ["nvdash"] = 8876,
5354 ["nvDash"] = 8877,
5355 ["nVdash"] = 8878,
5356 ["nVDash"] = 8879,
5357 ["prurel"] = 8880,

```

5358 ["LeftTriangle"] = 8882,  
5359 ["vartriangleleft"] = 8882,  
5360 ["vltri"] = 8882,  
5361 ["RightTriangle"] = 8883,  
5362 ["vartriangleright"] = 8883,  
5363 ["vrtri"] = 8883,  
5364 ["LeftTriangleEqual"] = 8884,  
5365 ["ltrie"] = 8884,  
5366 ["nvltrie"] = {8884, 8402},  
5367 ["trianglelefteq"] = 8884,  
5368 ["RightTriangleEqual"] = 8885,  
5369 ["nvrtrie"] = {8885, 8402},  
5370 ["rtrie"] = 8885,  
5371 ["trianglerighteq"] = 8885,  
5372 ["origof"] = 8886,  
5373 ["imof"] = 8887,  
5374 ["multimap"] = 8888,  
5375 ["mumap"] = 8888,  
5376 ["hercon"] = 8889,  
5377 ["intcal"] = 8890,  
5378 ["intercal"] = 8890,  
5379 ["veebar"] = 8891,  
5380 ["barvee"] = 8893,  
5381 ["angrtvb"] = 8894,  
5382 ["ltri"] = 8895,  
5383 ["Wedge"] = 8896,  
5384 ["bigwedge"] = 8896,  
5385 ["xwedge"] = 8896,  
5386 ["Vee"] = 8897,  
5387 ["bigvee"] = 8897,  
5388 ["xvee"] = 8897,  
5389 ["Intersection"] = 8898,  
5390 ["bigcap"] = 8898,  
5391 ["xcap"] = 8898,  
5392 ["Union"] = 8899,  
5393 ["bigcup"] = 8899,  
5394 ["xcup"] = 8899,  
5395 ["Diamond"] = 8900,  
5396 ["diam"] = 8900,  
5397 ["diamond"] = 8900,  
5398 ["sdot"] = 8901,  
5399 ["Star"] = 8902,  
5400 ["sstarf"] = 8902,  
5401 ["divideontimes"] = 8903,  
5402 ["divonx"] = 8903,  
5403 ["bowtie"] = 8904,  
5404 ["ltimes"] = 8905,

5405 ["rtimes"] = 8906,  
5406 ["leftthreetimes"] = 8907,  
5407 ["lthree"] = 8907,  
5408 ["rightthreetimes"] = 8908,  
5409 ["rthree"] = 8908,  
5410 ["backsimeq"] = 8909,  
5411 ["bsime"] = 8909,  
5412 ["curlyvee"] = 8910,  
5413 ["cuvee"] = 8910,  
5414 ["curlywedge"] = 8911,  
5415 ["cuwed"] = 8911,  
5416 ["Sub"] = 8912,  
5417 ["Subset"] = 8912,  
5418 ["Sup"] = 8913,  
5419 ["Supset"] = 8913,  
5420 ["Cap"] = 8914,  
5421 ["Cup"] = 8915,  
5422 ["fork"] = 8916,  
5423 ["pitchfork"] = 8916,  
5424 ["epar"] = 8917,  
5425 ["lessdot"] = 8918,  
5426 ["ltdot"] = 8918,  
5427 ["gtdot"] = 8919,  
5428 ["gtrdot"] = 8919,  
5429 ["L1"] = 8920,  
5430 ["nL1"] = {8920, 824},  
5431 ["Gg"] = 8921,  
5432 ["ggg"] = 8921,  
5433 ["nGg"] = {8921, 824},  
5434 ["LessEqualGreater"] = 8922,  
5435 ["leg"] = 8922,  
5436 ["lesg"] = {8922, 65024},  
5437 ["lesseqgtr"] = 8922,  
5438 ["GreaterEqualLess"] = 8923,  
5439 ["gel"] = 8923,  
5440 ["gesl"] = {8923, 65024},  
5441 ["gtreqless"] = 8923,  
5442 ["cuepr"] = 8926,  
5443 ["curlyeqprec"] = 8926,  
5444 ["cuesc"] = 8927,  
5445 ["curlyeqsucc"] = 8927,  
5446 ["NotPrecedesSlantEqual"] = 8928,  
5447 ["nprcue"] = 8928,  
5448 ["NotSucceedsSlantEqual"] = 8929,  
5449 ["nsccue"] = 8929,  
5450 ["NotSquareSubsetEqual"] = 8930,  
5451 ["nsqsube"] = 8930,



5452 ["NotSquareSupersetEqual"] = 8931,  
5453 ["nsqsupe"] = 8931,  
5454 ["lnsim"] = 8934,  
5455 ["gnsim"] = 8935,  
5456 ["precnsim"] = 8936,  
5457 ["prnsim"] = 8936,  
5458 ["scnsim"] = 8937,  
5459 ["succnsim"] = 8937,  
5460 ["NotLeftTriangle"] = 8938,  
5461 ["nltri"] = 8938,  
5462 ["ntriangleleft"] = 8938,  
5463 ["NotRightTriangle"] = 8939,  
5464 ["nrtri"] = 8939,  
5465 ["ntriangleright"] = 8939,  
5466 ["NotLeftTriangleEqual"] = 8940,  
5467 ["nltrie"] = 8940,  
5468 ["ntrianglelefteq"] = 8940,  
5469 ["NotRightTriangleEqual"] = 8941,  
5470 ["nrtrie"] = 8941,  
5471 ["ntrianglerighteq"] = 8941,  
5472 ["vellip"] = 8942,  
5473 ["ctdot"] = 8943,  
5474 ["utdot"] = 8944,  
5475 ["dtdot"] = 8945,  
5476 ["disin"] = 8946,  
5477 ["isinsv"] = 8947,  
5478 ["isins"] = 8948,  
5479 ["isindot"] = 8949,  
5480 ["notinvc"] = {8949, 824},  
5481 ["notinvc"] = 8950,  
5482 ["notinvb"] = 8951,  
5483 ["isinE"] = 8953,  
5484 ["notinE"] = {8953, 824},  
5485 ["nisd"] = 8954,  
5486 ["xnis"] = 8955,  
5487 ["nis"] = 8956,  
5488 ["notnivc"] = 8957,  
5489 ["notnivb"] = 8958,  
5490 ["barwed"] = 8965,  
5491 ["barwedge"] = 8965,  
5492 ["Barwed"] = 8966,  
5493 ["doublebarwedge"] = 8966,  
5494 ["LeftCeiling"] = 8968,  
5495 ["lceil"] = 8968,  
5496 ["RightCeiling"] = 8969,  
5497 ["rceil"] = 8969,  
5498 ["LeftFloor"] = 8970,

5499 ["lfloor"] = 8970,  
5500 ["RightFloor"] = 8971,  
5501 ["rfloor"] = 8971,  
5502 ["drcrop"] = 8972,  
5503 ["dlcrop"] = 8973,  
5504 ["urcrop"] = 8974,  
5505 ["ulcrop"] = 8975,  
5506 ["bnot"] = 8976,  
5507 ["proflines"] = 8978,  
5508 ["profsurf"] = 8979,  
5509 ["telrec"] = 8981,  
5510 ["target"] = 8982,  
5511 ["ulcorn"] = 8988,  
5512 ["ulcorner"] = 8988,  
5513 ["urcorn"] = 8989,  
5514 ["urcorner"] = 8989,  
5515 ["dlcorn"] = 8990,  
5516 ["llcorner"] = 8990,  
5517 ["drcorn"] = 8991,  
5518 ["lrcorn"] = 8991,  
5519 ["frown"] = 8994,  
5520 ["sfrown"] = 8994,  
5521 ["smile"] = 8995,  
5522 ["ssmile"] = 8995,  
5523 ["cylcty"] = 9005,  
5524 ["profalar"] = 9006,  
5525 ["topbot"] = 9014,  
5526 ["ovbar"] = 9021,  
5527 ["solbar"] = 9023,  
5528 ["angzarr"] = 9084,  
5529 ["lmoust"] = 9136,  
5530 ["lmoustache"] = 9136,  
5531 ["rmoust"] = 9137,  
5532 ["rmoustache"] = 9137,  
5533 ["OverBracket"] = 9140,  
5534 ["tbrk"] = 9140,  
5535 ["UnderBracket"] = 9141,  
5536 ["bbrk"] = 9141,  
5537 ["bbrktbrk"] = 9142,  
5538 ["OverParenthesis"] = 9180,  
5539 ["UnderParenthesis"] = 9181,  
5540 ["OverBrace"] = 9182,  
5541 ["UnderBrace"] = 9183,  
5542 ["trpezium"] = 9186,  
5543 ["elinters"] = 9191,  
5544 ["blank"] = 9251,  
5545 ["circledS"] = 9416,

5546 ["oS"] = 9416,  
5547 ["HorizontalLine"] = 9472,  
5548 ["boxh"] = 9472,  
5549 ["boxv"] = 9474,  
5550 ["boxdr"] = 9484,  
5551 ["boxdl"] = 9488,  
5552 ["boxur"] = 9492,  
5553 ["boxul"] = 9496,  
5554 ["boxvr"] = 9500,  
5555 ["boxvl"] = 9508,  
5556 ["boxhd"] = 9516,  
5557 ["boxhu"] = 9524,  
5558 ["boxvh"] = 9532,  
5559 ["boxH"] = 9552,  
5560 ["boxV"] = 9553,  
5561 ["boxdR"] = 9554,  
5562 ["boxDr"] = 9555,  
5563 ["boxDR"] = 9556,  
5564 ["boxdL"] = 9557,  
5565 ["boxDL"] = 9558,  
5566 ["boxDL"] = 9559,  
5567 ["boxuR"] = 9560,  
5568 ["boxUr"] = 9561,  
5569 ["boxUR"] = 9562,  
5570 ["boxuL"] = 9563,  
5571 ["boxUL"] = 9564,  
5572 ["boxUL"] = 9565,  
5573 ["boxvR"] = 9566,  
5574 ["boxVr"] = 9567,  
5575 ["boxVR"] = 9568,  
5576 ["boxvL"] = 9569,  
5577 ["boxVL"] = 9570,  
5578 ["boxVL"] = 9571,  
5579 ["boxHd"] = 9572,  
5580 ["boxhD"] = 9573,  
5581 ["boxHD"] = 9574,  
5582 ["boxHu"] = 9575,  
5583 ["boxhU"] = 9576,  
5584 ["boxHU"] = 9577,  
5585 ["boxvH"] = 9578,  
5586 ["boxVh"] = 9579,  
5587 ["boxVH"] = 9580,  
5588 ["uhblk"] = 9600,  
5589 ["lhblk"] = 9604,  
5590 ["block"] = 9608,  
5591 ["blk14"] = 9617,  
5592 ["blk12"] = 9618,

5593 ["blk34"] = 9619,  
5594 ["Square"] = 9633,  
5595 ["squ"] = 9633,  
5596 ["square"] = 9633,  
5597 ["FilledVerySmallSquare"] = 9642,  
5598 ["blacksquare"] = 9642,  
5599 ["squarf"] = 9642,  
5600 ["squf"] = 9642,  
5601 ["EmptyVerySmallSquare"] = 9643,  
5602 ["rect"] = 9645,  
5603 ["marker"] = 9646,  
5604 ["fltns"] = 9649,  
5605 ["bigtriangleup"] = 9651,  
5606 ["xutri"] = 9651,  
5607 ["blacktriangle"] = 9652,  
5608 ["utrif"] = 9652,  
5609 ["triangle"] = 9653,  
5610 ["utri"] = 9653,  
5611 ["blacktriangleright"] = 9656,  
5612 ["rtrif"] = 9656,  
5613 ["rtri"] = 9657,  
5614 ["triangleright"] = 9657,  
5615 ["bigtriangledown"] = 9661,  
5616 ["xdtri"] = 9661,  
5617 ["blacktriangledown"] = 9662,  
5618 ["dtrif"] = 9662,  
5619 ["dtri"] = 9663,  
5620 ["triangledown"] = 9663,  
5621 ["blacktriangleleft"] = 9666,  
5622 ["ltrif"] = 9666,  
5623 ["ltri"] = 9667,  
5624 ["triangleleft"] = 9667,  
5625 ["loz"] = 9674,  
5626 ["lozenge"] = 9674,  
5627 ["cir"] = 9675,  
5628 ["tridot"] = 9708,  
5629 ["bigcirc"] = 9711,  
5630 ["xcirc"] = 9711,  
5631 ["ultri"] = 9720,  
5632 ["urtri"] = 9721,  
5633 ["lltri"] = 9722,  
5634 ["EmptySmallSquare"] = 9723,  
5635 ["FilledSmallSquare"] = 9724,  
5636 ["bigstar"] = 9733,  
5637 ["starf"] = 9733,  
5638 ["star"] = 9734,  
5639 ["phone"] = 9742,

5640 ["female"] = 9792,  
5641 ["male"] = 9794,  
5642 ["spades"] = 9824,  
5643 ["spadesuit"] = 9824,  
5644 ["clubs"] = 9827,  
5645 ["clubsuit"] = 9827,  
5646 ["hearts"] = 9829,  
5647 ["heartsuit"] = 9829,  
5648 ["diamondsuit"] = 9830,  
5649 ["diams"] = 9830,  
5650 ["sung"] = 9834,  
5651 ["flat"] = 9837,  
5652 ["natur"] = 9838,  
5653 ["natural"] = 9838,  
5654 ["sharp"] = 9839,  
5655 ["check"] = 10003,  
5656 ["checkmark"] = 10003,  
5657 ["cross"] = 10007,  
5658 ["malt"] = 10016,  
5659 ["maltese"] = 10016,  
5660 ["sext"] = 10038,  
5661 ["VerticalSeparator"] = 10072,  
5662 ["lbrk"] = 10098,  
5663 ["rbrk"] = 10099,  
5664 ["bsolhsub"] = 10184,  
5665 ["suphsol"] = 10185,  
5666 ["LeftDoubleBracket"] = 10214,  
5667 ["lobrk"] = 10214,  
5668 ["RightDoubleBracket"] = 10215,  
5669 ["robrk"] = 10215,  
5670 ["LeftAngleBracket"] = 10216,  
5671 ["lang"] = 10216,  
5672 ["langle"] = 10216,  
5673 ["RightAngleBracket"] = 10217,  
5674 ["rang"] = 10217,  
5675 ["rangle"] = 10217,  
5676 ["Lang"] = 10218,  
5677 ["Rang"] = 10219,  
5678 ["loang"] = 10220,  
5679 ["roang"] = 10221,  
5680 ["LongLeftArrow"] = 10229,  
5681 ["longleftarrow"] = 10229,  
5682 ["xlarr"] = 10229,  
5683 ["LongRightArrow"] = 10230,  
5684 ["longrightarrow"] = 10230,  
5685 ["xrarr"] = 10230,  
5686 ["LongLeftRightArrow"] = 10231,

5687 ["longleftarrow"] = 10231,  
5688 ["xharr"] = 10231,  
5689 ["DoubleLongLeftArrow"] = 10232,  
5690 ["Longleftarrow"] = 10232,  
5691 ["xlArr"] = 10232,  
5692 ["DoubleLongRightArrow"] = 10233,  
5693 ["Longrightarrow"] = 10233,  
5694 ["xrArr"] = 10233,  
5695 ["DoubleLongLeftRightArrow"] = 10234,  
5696 ["Longleftrightarrow"] = 10234,  
5697 ["xhArr"] = 10234,  
5698 ["longmapsto"] = 10236,  
5699 ["xmap"] = 10236,  
5700 ["dzigrarr"] = 10239,  
5701 ["nvlArr"] = 10498,  
5702 ["nvrArr"] = 10499,  
5703 ["nvHarr"] = 10500,  
5704 ["Map"] = 10501,  
5705 ["lbarr"] = 10508,  
5706 ["bkarow"] = 10509,  
5707 ["rbarr"] = 10509,  
5708 ["lBarr"] = 10510,  
5709 ["dbkarow"] = 10511,  
5710 ["rBarr"] = 10511,  
5711 ["RBarr"] = 10512,  
5712 ["drbkarow"] = 10512,  
5713 ["DDottrahd"] = 10513,  
5714 ["UpArrowBar"] = 10514,  
5715 ["DownArrowBar"] = 10515,  
5716 ["Rarrtl"] = 10518,  
5717 ["latail"] = 10521,  
5718 ["ratail"] = 10522,  
5719 ["lAtail"] = 10523,  
5720 ["rAtail"] = 10524,  
5721 ["larrfs"] = 10525,  
5722 ["rarrfs"] = 10526,  
5723 ["larrbfs"] = 10527,  
5724 ["rarrbfs"] = 10528,  
5725 ["nwarhk"] = 10531,  
5726 ["nearhk"] = 10532,  
5727 ["hksearrow"] = 10533,  
5728 ["searhk"] = 10533,  
5729 ["hkswarrow"] = 10534,  
5730 ["swarhk"] = 10534,  
5731 ["nwnear"] = 10535,  
5732 ["nesear"] = 10536,  
5733 ["toea"] = 10536,

5734 ["seswar"] = 10537,  
5735 ["tosa"] = 10537,  
5736 ["swnwar"] = 10538,  
5737 ["nrarrc"] = {10547, 824},  
5738 ["rarrc"] = 10547,  
5739 ["cudarrrr"] = 10549,  
5740 ["ldca"] = 10550,  
5741 ["rdca"] = 10551,  
5742 ["cudarrl"] = 10552,  
5743 ["larrpl"] = 10553,  
5744 ["curarrm"] = 10556,  
5745 ["cularrp"] = 10557,  
5746 ["rarrpl"] = 10565,  
5747 ["harrcir"] = 10568,  
5748 ["Uarrocir"] = 10569,  
5749 ["lurdshar"] = 10570,  
5750 ["ldrushar"] = 10571,  
5751 ["LeftRightVector"] = 10574,  
5752 ["RightUpDownVector"] = 10575,  
5753 ["DownLeftRightVector"] = 10576,  
5754 ["LeftUpDownVector"] = 10577,  
5755 ["LeftVectorBar"] = 10578,  
5756 ["RightVectorBar"] = 10579,  
5757 ["RightUpVectorBar"] = 10580,  
5758 ["RightDownVectorBar"] = 10581,  
5759 ["DownLeftVectorBar"] = 10582,  
5760 ["DownRightVectorBar"] = 10583,  
5761 ["LeftUpVectorBar"] = 10584,  
5762 ["LeftDownVectorBar"] = 10585,  
5763 ["LeftTeeVector"] = 10586,  
5764 ["RightTeeVector"] = 10587,  
5765 ["RightUpTeeVector"] = 10588,  
5766 ["RightDownTeeVector"] = 10589,  
5767 ["DownLeftTeeVector"] = 10590,  
5768 ["DownRightTeeVector"] = 10591,  
5769 ["LeftUpTeeVector"] = 10592,  
5770 ["LeftDownTeeVector"] = 10593,  
5771 ["lHar"] = 10594,  
5772 ["uHar"] = 10595,  
5773 ["rHar"] = 10596,  
5774 ["dHar"] = 10597,  
5775 ["luruhar"] = 10598,  
5776 ["ldrdhar"] = 10599,  
5777 ["ruluhar"] = 10600,  
5778 ["rdldhar"] = 10601,  
5779 ["lharul"] = 10602,  
5780 ["llhard"] = 10603,

5781 ["rharul"] = 10604,  
5782 ["lrhard"] = 10605,  
5783 ["UpEquilibrium"] = 10606,  
5784 ["udhar"] = 10606,  
5785 ["ReverseUpEquilibrium"] = 10607,  
5786 ["duhar"] = 10607,  
5787 ["RoundImplies"] = 10608,  
5788 ["erarr"] = 10609,  
5789 ["simrarr"] = 10610,  
5790 ["larrsim"] = 10611,  
5791 ["rarrsim"] = 10612,  
5792 ["rarrap"] = 10613,  
5793 ["ltlarr"] = 10614,  
5794 ["gtrarr"] = 10616,  
5795 ["subrarr"] = 10617,  
5796 ["suplarr"] = 10619,  
5797 ["lfisht"] = 10620,  
5798 ["rfisht"] = 10621,  
5799 ["ufisht"] = 10622,  
5800 ["dfisht"] = 10623,  
5801 ["lopar"] = 10629,  
5802 ["ropar"] = 10630,  
5803 ["lbrke"] = 10635,  
5804 ["rbrke"] = 10636,  
5805 ["lbrkslu"] = 10637,  
5806 ["rbrksld"] = 10638,  
5807 ["lbrksld"] = 10639,  
5808 ["rbrkslu"] = 10640,  
5809 ["langd"] = 10641,  
5810 ["rangd"] = 10642,  
5811 ["lparlt"] = 10643,  
5812 ["rpargt"] = 10644,  
5813 ["gtlPar"] = 10645,  
5814 ["ltrPar"] = 10646,  
5815 ["vzigzag"] = 10650,  
5816 ["vangrt"] = 10652,  
5817 ["angrtvbd"] = 10653,  
5818 ["ange"] = 10660,  
5819 ["range"] = 10661,  
5820 ["dwangle"] = 10662,  
5821 ["uwangle"] = 10663,  
5822 ["angmsdaa"] = 10664,  
5823 ["angmsdab"] = 10665,  
5824 ["angmsdac"] = 10666,  
5825 ["angmsdad"] = 10667,  
5826 ["angmsdae"] = 10668,  
5827 ["angmsdaf"] = 10669,



5828 ["angmsdag"] = 10670,  
5829 ["angmsdah"] = 10671,  
5830 ["bemptyv"] = 10672,  
5831 ["demptyv"] = 10673,  
5832 ["cemptyv"] = 10674,  
5833 ["raemptyv"] = 10675,  
5834 ["laemptyv"] = 10676,  
5835 ["ohbar"] = 10677,  
5836 ["omid"] = 10678,  
5837 ["opar"] = 10679,  
5838 ["operp"] = 10681,  
5839 ["olcross"] = 10683,  
5840 ["odsold"] = 10684,  
5841 ["olcir"] = 10686,  
5842 ["ofcir"] = 10687,  
5843 ["olt"] = 10688,  
5844 ["ogt"] = 10689,  
5845 ["cirscir"] = 10690,  
5846 ["cirE"] = 10691,  
5847 ["solb"] = 10692,  
5848 ["bsolb"] = 10693,  
5849 ["boxbox"] = 10697,  
5850 ["trisb"] = 10701,  
5851 ["rtriltri"] = 10702,  
5852 ["LeftTriangleBar"] = 10703,  
5853 ["NotLeftTriangleBar"] = {10703, 824},  
5854 ["NotRightTriangleBar"] = {10704, 824},  
5855 ["RightTriangleBar"] = 10704,  
5856 ["iinfin"] = 10716,  
5857 ["infintie"] = 10717,  
5858 ["nvinfin"] = 10718,  
5859 ["eparsl"] = 10723,  
5860 ["smeparsl"] = 10724,  
5861 ["eqvparsl"] = 10725,  
5862 ["blacklozenge"] = 10731,  
5863 ["lozf"] = 10731,  
5864 ["RuleDelayed"] = 10740,  
5865 ["dsol"] = 10742,  
5866 ["bigodot"] = 10752,  
5867 ["xodot"] = 10752,  
5868 ["bigoplus"] = 10753,  
5869 ["xoplus"] = 10753,  
5870 ["bigotimes"] = 10754,  
5871 ["xotime"] = 10754,  
5872 ["biguplus"] = 10756,  
5873 ["xuplus"] = 10756,  
5874 ["bigscup"] = 10758,

5875 ["xscup"] = 10758,  
5876 ["iiiint"] = 10764,  
5877 ["qint"] = 10764,  
5878 ["fpartint"] = 10765,  
5879 ["cirfnint"] = 10768,  
5880 ["awint"] = 10769,  
5881 ["rppolint"] = 10770,  
5882 ["scpolint"] = 10771,  
5883 ["npolint"] = 10772,  
5884 ["pointint"] = 10773,  
5885 ["quatint"] = 10774,  
5886 ["intlarhk"] = 10775,  
5887 ["pluscir"] = 10786,  
5888 ["plusacir"] = 10787,  
5889 ["simplus"] = 10788,  
5890 ["plusdu"] = 10789,  
5891 ["plussim"] = 10790,  
5892 ["plustwo"] = 10791,  
5893 ["mcomma"] = 10793,  
5894 ["minusdu"] = 10794,  
5895 ["loplus"] = 10797,  
5896 ["roplus"] = 10798,  
5897 ["Cross"] = 10799,  
5898 ["timesd"] = 10800,  
5899 ["timesbar"] = 10801,  
5900 ["smashp"] = 10803,  
5901 ["lotimes"] = 10804,  
5902 ["rotimes"] = 10805,  
5903 ["otimesas"] = 10806,  
5904 ["Otimes"] = 10807,  
5905 ["odiv"] = 10808,  
5906 ["triplus"] = 10809,  
5907 ["triminus"] = 10810,  
5908 ["tritime"] = 10811,  
5909 ["intprod"] = 10812,  
5910 ["iproduct"] = 10812,  
5911 ["amalg"] = 10815,  
5912 ["capdot"] = 10816,  
5913 ["ncup"] = 10818,  
5914 ["ncap"] = 10819,  
5915 ["capand"] = 10820,  
5916 ["cupor"] = 10821,  
5917 ["cupcap"] = 10822,  
5918 ["capcup"] = 10823,  
5919 ["cupbrcap"] = 10824,  
5920 ["capbrcup"] = 10825,  
5921 ["cupcup"] = 10826,

5922 ["capcap"] = 10827,  
5923 ["ccups"] = 10828,  
5924 ["ccaps"] = 10829,  
5925 ["ccupssm"] = 10832,  
5926 ["And"] = 10835,  
5927 ["Or"] = 10836,  
5928 ["andand"] = 10837,  
5929 ["oror"] = 10838,  
5930 ["orslope"] = 10839,  
5931 ["andslope"] = 10840,  
5932 ["andv"] = 10842,  
5933 ["orv"] = 10843,  
5934 ["andd"] = 10844,  
5935 ["ord"] = 10845,  
5936 ["wedbar"] = 10847,  
5937 ["sdote"] = 10854,  
5938 ["simdot"] = 10858,  
5939 ["congdot"] = 10861,  
5940 ["ncongdot"] = {10861, 824},  
5941 ["easter"] = 10862,  
5942 ["apacir"] = 10863,  
5943 ["apE"] = 10864,  
5944 ["napE"] = {10864, 824},  
5945 ["eplus"] = 10865,  
5946 ["pluse"] = 10866,  
5947 ["Esim"] = 10867,  
5948 ["Colone"] = 10868,  
5949 ["Equal"] = 10869,  
5950 ["ddotseq"] = 10871,  
5951 ["eDDot"] = 10871,  
5952 ["equivDD"] = 10872,  
5953 ["ltcir"] = 10873,  
5954 ["gtcir"] = 10874,  
5955 ["ltquest"] = 10875,  
5956 ["gtquest"] = 10876,  
5957 ["LessSlantEqual"] = 10877,  
5958 ["NotLessSlantEqual"] = {10877, 824},  
5959 ["leqslant"] = 10877,  
5960 ["les"] = 10877,  
5961 ["nleqslant"] = {10877, 824},  
5962 ["nles"] = {10877, 824},  
5963 ["GreaterSlantEqual"] = 10878,  
5964 ["NotGreaterSlantEqual"] = {10878, 824},  
5965 ["geqslant"] = 10878,  
5966 ["ges"] = 10878,  
5967 ["ngeqslant"] = {10878, 824},  
5968 ["nges"] = {10878, 824},

5969 ["lesdot"] = 10879,  
5970 ["gesdot"] = 10880,  
5971 ["lesdoto"] = 10881,  
5972 ["gesdoto"] = 10882,  
5973 ["lesdotor"] = 10883,  
5974 ["gesdoto1"] = 10884,  
5975 ["lap"] = 10885,  
5976 ["lessapprox"] = 10885,  
5977 ["gap"] = 10886,  
5978 ["gtrapprox"] = 10886,  
5979 ["lne"] = 10887,  
5980 ["lneq"] = 10887,  
5981 ["gne"] = 10888,  
5982 ["gneq"] = 10888,  
5983 ["lnap"] = 10889,  
5984 ["lnapprox"] = 10889,  
5985 ["gnap"] = 10890,  
5986 ["gnapprox"] = 10890,  
5987 ["lEg"] = 10891,  
5988 ["lesseqqgtr"] = 10891,  
5989 ["gE1"] = 10892,  
5990 ["gtreqqless"] = 10892,  
5991 ["lsime"] = 10893,  
5992 ["gsime"] = 10894,  
5993 ["lsimg"] = 10895,  
5994 ["gsim1"] = 10896,  
5995 ["lgE"] = 10897,  
5996 ["glE"] = 10898,  
5997 ["lesges"] = 10899,  
5998 ["gesles"] = 10900,  
5999 ["els"] = 10901,  
6000 ["eqslantless"] = 10901,  
6001 ["egs"] = 10902,  
6002 ["eqslantgtr"] = 10902,  
6003 ["elsdot"] = 10903,  
6004 ["egsdot"] = 10904,  
6005 ["el"] = 10905,  
6006 ["eg"] = 10906,  
6007 ["sim1"] = 10909,  
6008 ["simg"] = 10910,  
6009 ["simlE"] = 10911,  
6010 ["simgE"] = 10912,  
6011 ["LessLess"] = 10913,  
6012 ["NotNestedLessLess"] = {10913, 824},  
6013 ["GreaterGreater"] = 10914,  
6014 ["NotNestedGreaterGreater"] = {10914, 824},  
6015 ["glj"] = 10916,

6016 ["gla"] = 10917,  
6017 ["ltcc"] = 10918,  
6018 ["gtcc"] = 10919,  
6019 ["lescc"] = 10920,  
6020 ["gescc"] = 10921,  
6021 ["smt"] = 10922,  
6022 ["lat"] = 10923,  
6023 ["smte"] = 10924,  
6024 ["smtes"] = {10924, 65024},  
6025 ["late"] = 10925,  
6026 ["lates"] = {10925, 65024},  
6027 ["bumpE"] = 10926,  
6028 ["NotPrecedesEqual"] = {10927, 824},  
6029 ["PrecedesEqual"] = 10927,  
6030 ["npre"] = {10927, 824},  
6031 ["npreceq"] = {10927, 824},  
6032 ["pre"] = 10927,  
6033 ["preceq"] = 10927,  
6034 ["NotSucceedsEqual"] = {10928, 824},  
6035 ["SucceedsEqual"] = 10928,  
6036 ["nsce"] = {10928, 824},  
6037 ["nsucceq"] = {10928, 824},  
6038 ["sce"] = 10928,  
6039 ["succeq"] = 10928,  
6040 ["prE"] = 10931,  
6041 ["scE"] = 10932,  
6042 ["precneqq"] = 10933,  
6043 ["prnE"] = 10933,  
6044 ["scnE"] = 10934,  
6045 ["succneqq"] = 10934,  
6046 ["prap"] = 10935,  
6047 ["precapprox"] = 10935,  
6048 ["scap"] = 10936,  
6049 ["succapprox"] = 10936,  
6050 ["precnapprox"] = 10937,  
6051 ["prnap"] = 10937,  
6052 ["scnap"] = 10938,  
6053 ["succnapprox"] = 10938,  
6054 ["Pr"] = 10939,  
6055 ["Sc"] = 10940,  
6056 ["subdot"] = 10941,  
6057 ["supdot"] = 10942,  
6058 ["subplus"] = 10943,  
6059 ["supplus"] = 10944,  
6060 ["submult"] = 10945,  
6061 ["supmult"] = 10946,  
6062 ["subedot"] = 10947,

6063 ["supedot"] = 10948,  
6064 ["nsubE"] = {10949, 824},  
6065 ["nsubseteqq"] = {10949, 824},  
6066 ["subE"] = 10949,  
6067 ["subseteqq"] = 10949,  
6068 ["nsupE"] = {10950, 824},  
6069 ["nsupseteqq"] = {10950, 824},  
6070 ["supE"] = 10950,  
6071 ["supseteqq"] = 10950,  
6072 ["subsim"] = 10951,  
6073 ["supsim"] = 10952,  
6074 ["subnE"] = 10955,  
6075 ["subsetneqq"] = 10955,  
6076 ["varsubsetneqq"] = {10955, 65024},  
6077 ["vsubnE"] = {10955, 65024},  
6078 ["supnE"] = 10956,  
6079 ["supsetneqq"] = 10956,  
6080 ["varsupsetneqq"] = {10956, 65024},  
6081 ["vsupnE"] = {10956, 65024},  
6082 ["csub"] = 10959,  
6083 ["csup"] = 10960,  
6084 ["csube"] = 10961,  
6085 ["csupe"] = 10962,  
6086 ["subsup"] = 10963,  
6087 ["supsub"] = 10964,  
6088 ["subsub"] = 10965,  
6089 ["supsup"] = 10966,  
6090 ["suphsub"] = 10967,  
6091 ["supdsub"] = 10968,  
6092 ["forkv"] = 10969,  
6093 ["topfork"] = 10970,  
6094 ["mlcp"] = 10971,  
6095 ["Dashv"] = 10980,  
6096 ["DoubleLeftTee"] = 10980,  
6097 ["Vdashl"] = 10982,  
6098 ["Barv"] = 10983,  
6099 ["vBar"] = 10984,  
6100 ["vBarv"] = 10985,  
6101 ["Vbar"] = 10987,  
6102 ["Not"] = 10988,  
6103 ["bNot"] = 10989,  
6104 ["rnmid"] = 10990,  
6105 ["cirmid"] = 10991,  
6106 ["midcir"] = 10992,  
6107 ["topcir"] = 10993,  
6108 ["nhpar"] = 10994,  
6109 ["parsim"] = 10995,

```
6110 ["npars1"] = {11005, 8421},
6111 ["pars1"] = 11005,
6112 ["fflig"] = 64256,
6113 ["filig"] = 64257,
6114 ["fllig"] = 64258,
6115 ["ffilig"] = 64259,
6116 ["ffllig"] = 64260,
6117 ["Ascr"] = 119964,
6118 ["Cscr"] = 119966,
6119 ["Dscr"] = 119967,
6120 ["Gscr"] = 119970,
6121 ["Jscr"] = 119973,
6122 ["Kscr"] = 119974,
6123 ["Nscr"] = 119977,
6124 ["Oscr"] = 119978,
6125 ["Pscr"] = 119979,
6126 ["Qscr"] = 119980,
6127 ["Sscr"] = 119982,
6128 ["Tscr"] = 119983,
6129 ["Uscr"] = 119984,
6130 ["Vscr"] = 119985,
6131 ["Wscr"] = 119986,
6132 ["Xscr"] = 119987,
6133 ["Yscr"] = 119988,
6134 ["Zscr"] = 119989,
6135 ["ascr"] = 119990,
6136 ["bscr"] = 119991,
6137 ["cscr"] = 119992,
6138 ["dscr"] = 119993,
6139 ["fscr"] = 119995,
6140 ["hscr"] = 119997,
6141 ["iscr"] = 119998,
6142 ["jscr"] = 119999,
6143 ["kscr"] = 120000,
6144 ["lscr"] = 120001,
6145 ["mscr"] = 120002,
6146 ["nscr"] = 120003,
6147 ["pscr"] = 120005,
6148 ["qscr"] = 120006,
6149 ["rscr"] = 120007,
6150 ["sscr"] = 120008,
6151 ["tscr"] = 120009,
6152 ["uscr"] = 120010,
6153 ["vscr"] = 120011,
6154 ["wscr"] = 120012,
6155 ["xscr"] = 120013,
6156 ["yscr"] = 120014,
```

6157 ["zscr"] = 120015,  
6158 ["Afr"] = 120068,  
6159 ["Bfr"] = 120069,  
6160 ["Dfr"] = 120071,  
6161 ["Efr"] = 120072,  
6162 ["Ffr"] = 120073,  
6163 ["Gfr"] = 120074,  
6164 ["Jfr"] = 120077,  
6165 ["Kfr"] = 120078,  
6166 ["Lfr"] = 120079,  
6167 ["Mfr"] = 120080,  
6168 ["Nfr"] = 120081,  
6169 ["Ofr"] = 120082,  
6170 ["Pfr"] = 120083,  
6171 ["Qfr"] = 120084,  
6172 ["Sfr"] = 120086,  
6173 ["Tfr"] = 120087,  
6174 ["Ufr"] = 120088,  
6175 ["Vfr"] = 120089,  
6176 ["Wfr"] = 120090,  
6177 ["Xfr"] = 120091,  
6178 ["Yfr"] = 120092,  
6179 ["afr"] = 120094,  
6180 ["bfr"] = 120095,  
6181 ["cfr"] = 120096,  
6182 ["dfr"] = 120097,  
6183 ["efr"] = 120098,  
6184 ["ffr"] = 120099,  
6185 ["gfr"] = 120100,  
6186 ["hfr"] = 120101,  
6187 ["ifr"] = 120102,  
6188 ["jfr"] = 120103,  
6189 ["kfr"] = 120104,  
6190 ["lfr"] = 120105,  
6191 ["mfr"] = 120106,  
6192 ["nfr"] = 120107,  
6193 ["ofr"] = 120108,  
6194 ["pfr"] = 120109,  
6195 ["qfr"] = 120110,  
6196 ["rfr"] = 120111,  
6197 ["sfr"] = 120112,  
6198 ["tfr"] = 120113,  
6199 ["ufr"] = 120114,  
6200 ["vfr"] = 120115,  
6201 ["wfr"] = 120116,  
6202 ["xfr"] = 120117,  
6203 ["yfr"] = 120118,



```
6204 ["zfr"] = 120119,  
6205 ["Aopf"] = 120120,  
6206 ["Bopf"] = 120121,  
6207 ["Dopf"] = 120123,  
6208 ["Eopf"] = 120124,  
6209 ["Fopf"] = 120125,  
6210 ["Gopf"] = 120126,  
6211 ["Iopf"] = 120128,  
6212 ["Jopf"] = 120129,  
6213 ["Kopf"] = 120130,  
6214 ["Lopf"] = 120131,  
6215 ["Mopf"] = 120132,  
6216 ["Oopf"] = 120134,  
6217 ["Sopf"] = 120138,  
6218 ["Topf"] = 120139,  
6219 ["Uopf"] = 120140,  
6220 ["Vopf"] = 120141,  
6221 ["Wopf"] = 120142,  
6222 ["Xopf"] = 120143,  
6223 ["Yopf"] = 120144,  
6224 ["aopf"] = 120146,  
6225 ["bopf"] = 120147,  
6226 ["copf"] = 120148,  
6227 ["dopf"] = 120149,  
6228 ["eopf"] = 120150,  
6229 ["fopf"] = 120151,  
6230 ["gopf"] = 120152,  
6231 ["hopf"] = 120153,  
6232 ["iopf"] = 120154,  
6233 ["jopf"] = 120155,  
6234 ["kopf"] = 120156,  
6235 ["lopf"] = 120157,  
6236 ["mopf"] = 120158,  
6237 ["nopf"] = 120159,  
6238 ["oopf"] = 120160,  
6239 ["popf"] = 120161,  
6240 ["qopf"] = 120162,  
6241 ["ropf"] = 120163,  
6242 ["sopf"] = 120164,  
6243 ["topf"] = 120165,  
6244 ["uopf"] = 120166,  
6245 ["vopf"] = 120167,  
6246 ["wopf"] = 120168,  
6247 ["xopf"] = 120169,  
6248 ["yopf"] = 120170,  
6249 ["zopf"] = 120171,  
6250 }
```

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

```
6251 function entities.dec_entity(s)
6252   local n = tonumber(s)
6253   if n == nil then
6254     return "&#" .. s .. ";" -- fallback for unknown entities
6255   end
6256   return unicode.utf8.char(n)
6257 end
```

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

```
6258 function entities.hex_entity(s)
6259   local n = tonumber("0x"..s)
6260   if n == nil then
6261     return "&#x" .. s .. ";" -- fallback for unknown entities
6262   end
6263   return unicode.utf8.char(n)
6264 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.

```
6265 function entities.hex_entity_with_x_char(x, s)
6266   local n = tonumber("0x"..s)
6267   if n == nil then
6268     return "&#" .. x .. s .. ";" -- fallback for unknown entities
6269   end
6270   return unicode.utf8.char(n)
6271 end
```

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

```
6272 function entities.char_entity(s)
6273   local code_points = character_entities[s]
6274   if code_points == nil then
6275     return "&" .. s .. ";"
6276   end
6277   if type(code_points) ~= 'table' then
6278     code_points = {code_points}
6279   end
6280   local char_table = {}
6281   for _, code_point in ipairs(code_points) do
6282     table.insert(char_table, unicode.utf8.char(code_point))
6283   end
6284   return table.concat(char_table)
6285 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.

```
6286 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.

```
6287 function M.writer.new(options)
6288   local self = {}
```

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

```
6289   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.

```
6290   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.

```
6291   local slice_specifiers = {}
6292   for specifier in options.slice:gmatch("[^%s]+") do
6293     table.insert(slice_specifiers, specifier)
6294   end
6295
6296   if #slice_specifiers == 2 then
6297     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6298     local slice_begin_type = self.slice_begin:sub(1, 1)
6299     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
6300       self.slice_begin = "^" .. self.slice_begin
6301     end
6302     local slice_end_type = self.slice_end:sub(1, 1)
6303     if slice_end_type ~= "^" and slice_end_type ~= "$" then
```

```

6304     self.slice_end = "$" .. self.slice_end
6305   end
6306   elseif #slice_specifiers == 1 then
6307     self.slice_begin = "^" .. slice_specifiers[1]
6308     self.slice_end = "$" .. slice_specifiers[1]
6309   end
6310
6311   self.slice_begin_type = self.slice_begin:sub(1, 1)
6312   self.slice_begin_identifer = self.slice_begin:sub(2) or ""
6313   self.slice_end_type = self.slice_end:sub(1, 1)
6314   self.slice_end_identifer = self.slice_end:sub(2) or ""
6315
6316   if self.slice_begin == "^" and self.slice_end ~= "^" then
6317     self.is_writing = true
6318   else
6319     self.is_writing = false
6320   end

```

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

```
6321   self.space = " "
```

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

```
6322   self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
6323   function self.plain(s)
6324     return s
6325   end

```

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

```
6326   function self.paragraph(s)
6327     if not self.is_writing then return "" end
6328     return s
6329   end

```

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

```
6330   self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
6331   function self.interblocksep()
6332     if not self.is_writing then return "" end
6333     return self.interblocksep_text
6334   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.

```
6335   self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
```

```

6336 function self.paragraphsep()
6337     if not self.is_writing then return "" end
6338     return self.paragraphsep_text
6339 end

```

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

```

6340 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
6341 function self.undosep()
6342     if not self.is_writing then return "" end
6343     return self.undosep_text
6344 end

```

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

```

6345 self.soft_line_break = function()
6346     if self.flatten_inlines then return "\n" end
6347     return "\\markdownRendererSoftLineBreak\n{"
6348 end

```

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

```

6349 self.hard_line_break = function()
6350     if self.flatten_inlines then return "\n" end
6351     return "\\markdownRendererHardLineBreak\n{"
6352 end

```

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

```

6353 self.ellipsis = "\\markdownRendererEllipsis{"

```

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

```

6354 function self.thematic_break()
6355     if not self.is_writing then return "" end
6356     return "\\markdownRendererThematicBreak{"
6357 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.

```

6358 self.escaped_uri_chars = {
6359     [{""] = "\\markdownRendererLeftBrace{"},
6360     ["}"] = "\\markdownRendererRightBrace{"},
6361     [{"\\"} = "\\markdownRendererBackslash{"},
6362     [{"r"} = " ",
6363     [{"n"} = " ",
6364 }
6365 self.escaped_minimal_strings = {
6366     [{"^"} = "\\markdownRendererCircumflex",
6367     .. "\\markdownRendererCircumflex ",
6368     [{"☒"} = "\\markdownRendererTickedBox{"},
6369     [{"☐"} = "\\markdownRendererHalfTickedBox{"},

```

```

6370     ["□"] = "\\markdownRendererUntickedBox{}",
6371     [entities.hex_entity('FFFD')]
6372     = "\\markdownRendererReplacementCharacter{}",
6373 }

```

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

```

6374 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6375 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.

```

6376 self.escaped_chars = {
6377     [{""] = "\\markdownRendererLeftBrace{}",
6378     ["}"] = "\\markdownRendererRightBrace{}",
6379     ["%"] = "\\markdownRendererPercentSign{}",
6380     ["\\"] = "\\markdownRendererBackslash{}",
6381     ["#"] = "\\markdownRendererHash{}",
6382     ["$"] = "\\markdownRendererDollarSign{}",
6383     ["&"] = "\\markdownRendererAmpersand{}",
6384     ["_"] = "\\markdownRendererUnderscore{}",
6385     ["^"] = "\\markdownRendererCircumflex{}",
6386     ["~"] = "\\markdownRendererTilde{}",
6387     ["|"] = "\\markdownRendererPipe{}",
6388     [entities.hex_entity('0000')]
6389     = "\\markdownRendererReplacementCharacter{}",
6390 }

```

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.

```

6391 local function create_escaper(char_escapes, string_escapes)
6392     local escape = util.escaper(char_escapes, string_escapes)
6393     return function(s)
6394         if self.flatten_inlines then return s end
6395         return escape(s)
6396     end
6397 end
6398 local escape_typographic_text = create_escaper(
6399     self.escaped_chars, self.escaped_strings)
6400 local escape_programmatic_text = create_escaper(
6401     self.escaped_uri_chars, self.escaped_minimal_strings)
6402 local escape_minimal = create_escaper(
6403     {}, 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.

```

6404 self.escape = escape_typographic_text
6405 self.math = escape_minimal
6406 if options.hybrid then
6407     self.identifier = escape_minimal
6408     self.string = escape_minimal
6409     self.uri = escape_minimal
6410     self.infostring = escape_minimal
6411 else
6412     self.identifier = escape_programmatic_text
6413     self.string = escape_typographic_text
6414     self.uri = escape_programmatic_text
6415     self.infostring = escape_programmatic_text
6416 end

```

Define `writer->warning` as a function that will transform an input warning `t` with optional more warning text `m` to the output format.

```

6417 function self.warning(t, m)
6418     return {"\markdownRendererWarning{" , self.escape(t), "}{" ,
6419           escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6420           escape_minimal(m or ""), "}"}
6421 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.

```

6422 function self.error(t, m)
6423     return {"\markdownRendererError{" , self.escape(t), "}{" ,
6424           escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6425           escape_minimal(m or ""), "}"}
6426 end

```

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

```

6427 function self.code(s, attributes)
6428     if self.flatten_inlines then return s end
6429     local buf = {}
6430     if attributes ~= nil then
6431         table.insert(buf,
6432             "\markdownRendererCodeSpanAttributeContextBegin\n")
6433         table.insert(buf, self.attributes(attributes))

```

```

6434     end
6435     table.insert(buf,
6436         {"\\markdownRendererCodeSpan{" , self.escape(s), "}"}
6437     if attributes ~= nil then
6438         table.insert(buf,
6439             "\\markdownRendererCodeSpanAttributeContextEnd{")
6440     end
6441     return buf
6442 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.

```

6443 function self.link(lab, src, tit, attributes)
6444     if self.flatten_inlines then return lab end
6445     local buf = {}
6446     if attributes ~= nil then
6447         table.insert(buf,
6448             "\\markdownRendererLinkAttributeContextBegin\n")
6449         table.insert(buf, self.attributes(attributes))
6450     end
6451     table.insert(buf, {"\\markdownRendererLink{" ,lab,"}"} ,
6452         {"",self.escape(src),""} ,
6453         {"",self.uri(src),""} ,
6454         {"",self.string(tit or ""),""}})
6455     if attributes ~= nil then
6456         table.insert(buf,
6457             "\\markdownRendererLinkAttributeContextEnd{")
6458     end
6459     return buf
6460 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.

```

6461 function self.image(lab, src, tit, attributes)
6462     if self.flatten_inlines then return lab end
6463     local buf = {}
6464     if attributes ~= nil then
6465         table.insert(buf,
6466             "\\markdownRendererImageAttributeContextBegin\n")
6467         table.insert(buf, self.attributes(attributes))
6468     end
6469     table.insert(buf, {"\\markdownRendererImage{" ,lab,"}"} ,
6470         {"",self.string(src),""} ,
6471         {"",self.uri(src),""} ,
6472         {"",self.string(tit or ""),""}})

```



```

6473     if attributes ~= nil then
6474         table.insert(buf,
6475             "\\markdownRendererImageAttributeContextEnd{ }")
6476     end
6477     return buf
6478 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.

```

6479     function self.bulletlist(items,tight)
6480         if not self.is_writing then return "" end
6481         local buffer = {}
6482         for _,item in ipairs(items) do
6483             if item ~= "" then
6484                 buffer[#buffer + 1] = self.bulletitem(item)
6485             end
6486         end
6487         local contents = util.intersperse(buffer,"\n")
6488         if tight and options.tightLists then
6489             return {"\\markdownRendererUlBeginTight\n",contents,
6490                 "\n\\markdownRendererUlEndTight "}
6491         else
6492             return {"\\markdownRendererUlBegin\n",contents,
6493                 "\n\\markdownRendererUlEnd "}
6494         end
6495     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.

```

6496     function self.bulletitem(s)
6497         return {"\\markdownRendererUlItem ",s,
6498             "\\markdownRendererUlItemEnd "}
6499     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.

```

6500     function self.orderedlist(items,tight,startnum)
6501         if not self.is_writing then return "" end
6502         local buffer = {}
6503         local num = startnum
6504         for _,item in ipairs(items) do
6505             if item ~= "" then
6506                 buffer[#buffer + 1] = self.ordereditem(item,num)
6507             end
6508             if num ~= nil and item ~= "" then

```

```

6509         num = num + 1
6510     end
6511 end
6512 local contents = util.intersperse(buffer,"\n")
6513 if tight and options.tightLists then
6514     return {"\\markdownRenderer01BeginTight\n",contents,
6515           "\\n\\markdownRenderer01EndTight "}
6516 else
6517     return {"\\markdownRenderer01Begin\n",contents,
6518           "\\n\\markdownRenderer01End "}
6519 end
6520 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.

```

6521 function self.ordereditem(s,num)
6522     if num ~= nil then
6523         return {"\\markdownRenderer01ItemWithNumber{" ,num,"}",s,
6524               "\\markdownRenderer01ItemEnd "}
6525     else
6526         return {"\\markdownRenderer01Item ",s,
6527               "\\markdownRenderer01ItemEnd "}
6528     end
6529 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.

```

6530 function self.inline_html_comment(contents)
6531     if self.flatten_inlines then return contents end
6532     return {"\\markdownRendererInlineHtmlComment{" ,contents,""}
6533 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.

```

6534 function self.inline_html_tag(contents)
6535     if self.flatten_inlines then return contents end
6536     return {"\\markdownRendererInlineHtmlTag{" ,
6537           self.string(contents),""}
6538 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.

```

6539 function self.block_html_element(s)
6540     if not self.is_writing then return "" end

```

```

6541     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6542     return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
6543 end

```

Define `writer->emphasis` as a function that will transform an emphasized span `s` of input text to the output format.

```

6544     function self.emphasis(s)
6545         if self.flatten_inlines then return s end
6546         return {"\\markdownRendererEmphasis{" ,s,"}"}
6547     end

```

Define `writer->checkbox` as a function that will transform a number `f` to the output format.

```

6548     function self.checkbox(f)
6549         if f == 1.0 then
6550             return "☒ "
6551         elseif f == 0.0 then
6552             return "☐ "
6553         else
6554             return "◻ "
6555         end
6556     end

```

Define `writer->strong` as a function that will transform a strongly emphasized span `s` of input text to the output format.

```

6557     function self.strong(s)
6558         if self.flatten_inlines then return s end
6559         return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
6560     end

```

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

```

6561     function self.blockquote(s)
6562         if not self.is_writing then return "" end
6563         return {"\\markdownRendererBlockQuoteBegin\n",s,
6564             "\\markdownRendererBlockQuoteEnd "}
6565     end

```

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

```

6566     function self.verbatim(s)
6567         if not self.is_writing then return "" end
6568         s = s:gsub("\n$", "")
6569         local name = util.cache_verbatim(options.cacheDir, s)
6570         return {"\\markdownRendererInputVerbatim{" ,name,"}"}
6571     end

```

Define `writer->document` as a function that will transform a document `d` to the output format.

```

6572 function self.document(d)
6573     local buf = {"\\markdownRendererDocumentBegin\n"}
6574
6575     -- warn against the `hybrid` option
6576     if options.hybrid then
6577         local text = "The `hybrid` option has been soft-deprecated."
6578         local more = "Consider using one of the following better options "
6579             .. "for mixing TeX and markdown: `contentBlocks`, "
6580             .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6581             .. "`texMathSingleBackslash`, and "
6582             .. "`texMathDoubleBackslash`. "
6583             .. "For more information, see the user manual at "
6584             .. "<https://witiko.github.io/markdown/>."
6585         table.insert(buf, self.warning(text, more))
6586     end
6587
6588     -- insert the text of the document
6589     table.insert(buf, d)
6590
6591     -- pop all attributes
6592     table.insert(buf, self.pop_attributes())
6593
6594     table.insert(buf, "\\markdownRendererDocumentEnd")
6595
6596     return buf
6597 end

```

Define `writer->attributes` as a function that will transform input attributes `attrs` to the output format.

```

6598 local seen_identifiers = {}
6599 local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
6600 local function normalize_attributes(attributes, auto_identifiers)
6601     -- normalize attributes
6602     local normalized_attributes = {}
6603     local has_explicit_identifiers = false
6604     local key, value
6605     for _, attribute in ipairs(attributes or {}) do
6606         if attribute:sub(1, 1) == "#" then
6607             table.insert(normalized_attributes, attribute)
6608             has_explicit_identifiers = true
6609             seen_identifiers[attribute:sub(2)] = true
6610         elseif attribute:sub(1, 1) == "." then
6611             table.insert(normalized_attributes, attribute)
6612         else
6613             key, value = attribute:match(key_value_regex)
6614             if key:lower() == "id" then
6615                 table.insert(normalized_attributes, "#" .. value)

```

```

6616     elseif key:lower() == "class" then
6617         local classes = {}
6618         for class in value:gmatch("%S+") do
6619             table.insert(classes, class)
6620         end
6621         table.sort(classes)
6622         for _, class in ipairs(classes) do
6623             table.insert(normalized_attributes, "." .. class)
6624         end
6625     else
6626         table.insert(normalized_attributes, attribute)
6627     end
6628 end
6629 end
6630
6631 -- if no explicit identifiers exist, add auto identifiers
6632 if not has_explicit_identifiers and auto_identifiers ~= nil then
6633     local seen_auto_identifiers = {}
6634     for _, auto_identifier in ipairs(auto_identifiers) do
6635         if seen_auto_identifiers[auto_identifier] == nil then
6636             seen_auto_identifiers[auto_identifier] = true
6637             if seen_identifiers[auto_identifier] == nil then
6638                 seen_identifiers[auto_identifier] = true
6639                 table.insert(normalized_attributes,
6640                     "#" .. auto_identifier)
6641             else
6642                 local auto_identifier_number = 1
6643                 while true do
6644                     local numbered_auto_identifier = auto_identifier .. "-"
6645   .. auto_identifier_number
6646                     if seen_identifiers[numbered_auto_identifier] == nil then
6647                         seen_identifiers[numbered_auto_identifier] = true
6648                         table.insert(normalized_attributes,
6649                             "#" .. numbered_auto_identifier)
6650                     break
6651                 end
6652                 auto_identifier_number = auto_identifier_number + 1
6653             end
6654         end
6655     end
6656 end
6657 end
6658
6659 -- sort and deduplicate normalized attributes
6660 table.sort(normalized_attributes)
6661 local seen_normalized_attributes = {}
6662 local deduplicated_normalized_attributes = {}

```

```

6663     for _, attribute in ipairs(normalized_attributes) do
6664         if seen_normalized_attributes[attribute] == nil then
6665             seen_normalized_attributes[attribute] = true
6666             table.insert(deduplicated_normalized_attributes, attribute)
6667         end
6668     end
6669
6670     return deduplicated_normalized_attributes
6671 end
6672
6673 function self.attributes(attributes, should_normalize_attributes)
6674     local normalized_attributes
6675     if should_normalize_attributes == false then
6676         normalized_attributes = attributes
6677     else
6678         normalized_attributes = normalize_attributes(attributes)
6679     end
6680
6681     local buf = {}
6682     local key, value
6683     for _, attribute in ipairs(normalized_attributes) do
6684         if attribute:sub(1, 1) == "#" then
6685             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6686                 attribute:sub(2), "}"})
6687         elseif attribute:sub(1, 1) == "." then
6688             table.insert(buf, {"\\markdownRendererAttributeName{" ,
6689                 attribute:sub(2), "}"})
6690         else
6691             key, value = attribute:match(key_value_regex)
6692             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6693                 key, "{" , value, "}"})
6694         end
6695     end
6696
6697     return buf
6698 end

```

Define `writer->active_attributes` as a stack of block-level attributes that are currently active. The `writer->active_attributes` member variable is mutable.

```

6699     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`.

```

6700     self.attribute_type_levels = {}
6701     setmetatable(self.attribute_type_levels,
6702         { __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`.

```

6703 local function apply_attributes()
6704   local buf = {}
6705   for i = 1, #self.active_attributes do
6706     local start_output = self.active_attributes[i][3]
6707     if start_output ~= nil then
6708       table.insert(buf, start_output)
6709     end
6710   end
6711   return buf
6712 end
6713
6714 local function tear_down_attributes()
6715   local buf = {}
6716   for i = #self.active_attributes, 1, -1 do
6717     local end_output = self.active_attributes[i][4]
6718     if end_output ~= nil then
6719       table.insert(buf, end_output)
6720     end
6721   end
6722   return buf
6723 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.

```

6724 function self.push_attributes(attribute_type, attributes,
6725                               start_output, end_output)
6726   local attribute_type_level
6727     = self.attribute_type_levels[attribute_type]
6728   self.attribute_type_levels[attribute_type]
6729     = attribute_type_level + 1
6730
6731   -- index attributes in a hash table for easy lookup
6732   attributes = attributes or {}
6733   for i = 1, #attributes do
6734     attributes[attributes[i]] = true
6735   end
6736
6737   local buf = {}
6738   -- handle slicing
6739   if attributes["#" .. self.slice_end_identifrier] ~= nil and
6740     self.slice_end_type == "^" then

```

```

6741     if self.is_writing then
6742         table.insert(buf, self.undosep())
6743         table.insert(buf, tear_down_attributes())
6744     end
6745     self.is_writing = false
6746 end
6747 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6748     self.slice_begin_type == "^" then
6749     table.insert(buf, apply_attributes())
6750     self.is_writing = true
6751 end
6752 if self.is_writing and start_output ~= nil then
6753     table.insert(buf, start_output)
6754 end
6755 table.insert(self.active_attributes,
6756             {attribute_type, attributes,
6757              start_output, end_output})
6758 return buf
6759 end
6760

```

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`).

```

6761 function self.pop_attributes(attribute_type)
6762     local buf = {}
6763     -- pop attributes until we find attributes of correct type
6764     -- or until no attributes remain
6765     local current_attribute_type = false
6766     while current_attribute_type ~= attribute_type and
6767         #self.active_attributes > 0 do
6768         local attributes, _, end_output
6769         current_attribute_type, attributes, _, end_output = table.unpack(
6770             self.active_attributes[#self.active_attributes])
6771         local attribute_type_level
6772         = self.attribute_type_levels[current_attribute_type]
6773         self.attribute_type_levels[current_attribute_type]
6774         = attribute_type_level - 1
6775         if self.is_writing and end_output ~= nil then
6776             table.insert(buf, end_output)
6777         end
6778         table.remove(self.active_attributes, #self.active_attributes)
6779         -- handle slicing
6780         if attributes["#" .. self.slice_end_identifier] ~= nil

```



```

6781         and self.slice_end_type == "$" then
6782     if self.is_writing then
6783         table.insert(buf, self.undosep())
6784         table.insert(buf, tear_down_attributes())
6785     end
6786     self.is_writing = false
6787 end
6788 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6789 self.slice_begin_type == "$" then
6790     self.is_writing = true
6791     table.insert(buf, apply_attributes())
6792 end
6793 end
6794 return buf
6795 end

```

Create an auto identifier string by stripping and converting characters from string `s`.

```

6796 local function create_auto_identifier(s)
6797     local buffer = {}
6798     local prev_space = false
6799     local letter_found = false
6800     local normalized_s = s
6801     if not options.unicodeNormalization
6802         or options.unicodeNormalizationForm ~= "nfc" then
6803         normalized_s = uni_algos.normalize.NFC(normalized_s)
6804     end
6805
6806     for _, code in utf8.codes(normalized_s) do
6807         local char = utf8.char(code)
6808
6809         -- Remove everything up to the first letter.
6810         if not letter_found then
6811             local is_letter = unicode.utf8.match(char, "%a")
6812             if is_letter then
6813                 letter_found = true
6814             else
6815                 goto continue
6816             end
6817         end
6818
6819         -- Remove all non-alphanumeric characters, except underscores,
6820         -- hyphens, and periods.
6821         if not unicode.utf8.match(char, "[%w_-%.%s]") then
6822             goto continue
6823         end
6824
6825         -- Replace all spaces and newlines with hyphens.
6826         if unicode.utf8.match(char, "[%s\n]") then

```

```

6827     char = "-"
6828     if prev_space then
6829         goto continue
6830     else
6831         prev_space = true
6832     end
6833 else
6834     -- Convert all alphabetic characters to lowercase.
6835     char = unicode.utf8.lower(char)
6836     prev_space = false
6837 end
6838
6839 table.insert(buffer, char)
6840
6841 ::continue::
6842 end
6843
6844 if prev_space then
6845     table.remove(buffer)
6846 end
6847
6848 local identifier = #buffer == 0 and "section"
6849                 or table.concat(buffer, "")
6850 return identifier
6851 end

```

Create an GitHub-flavored auto identifier string by stripping and converting characters from string `s`.

```

6852 local function create_gfm_auto_identifier(s)
6853     local buffer = {}
6854     local prev_space = false
6855     local letter_found = false
6856     local normalized_s = s
6857     if not options.unicodeNormalization
6858         or options.unicodeNormalizationForm ~= "nfc" then
6859         normalized_s = uni_algos.normalize.NFC(normalized_s)
6860     end
6861
6862     for _, code in utf8.codes(normalized_s) do
6863         local char = utf8.char(code)
6864
6865         -- Remove everything up to the first non-space.
6866         if not letter_found then
6867             local is_letter = unicode.utf8.match(char, "%S")
6868             if is_letter then
6869                 letter_found = true
6870             else

```

```

6871         goto continue
6872     end
6873 end
6874
6875     -- Remove all non-alphanumeric characters, except underscores
6876     -- and hyphens.
6877     if not unicode.utf8.match(char, "[%w_-%s]") then
6878         prev_space = false
6879         goto continue
6880     end
6881
6882     -- Replace all spaces and newlines with hyphens.
6883     if unicode.utf8.match(char, "[%s\n]") then
6884         char = "-"
6885         if prev_space then
6886             goto continue
6887         else
6888             prev_space = true
6889         end
6890     else
6891         -- Convert all alphabetic characters to lowercase.
6892         char = unicode.utf8.lower(char)
6893         prev_space = false
6894     end
6895
6896     table.insert(buffer, char)
6897
6898     ::continue::
6899 end
6900
6901 if prev_space then
6902     table.remove(buffer)
6903 end
6904
6905 local identifier = #buffer == 0 and "section"
6906                 or table.concat(buffer, "")
6907 return identifier
6908 end

```

Define `writer->heading` as a function that will transform an input heading `s` at level `level` with attributes `attributes` to the output format.

```

6909 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6910 self.secend_text = "\n\\markdownRendererSectionEnd "
6911 function self.heading(s, level, attributes)
6912     local buf = {}
6913     local flat_text, inlines = table.unpack(s)
6914

```

```

6915 -- push empty attributes for implied sections
6916 while self.attribute_type_levels["heading"] < level - 1 do
6917     table.insert(buf,
6918         self.push_attributes("heading",
6919             nil,
6920             self.secbegin_text,
6921             self.secend_text))
6922 end
6923
6924 -- pop attributes for sections that have ended
6925 while self.attribute_type_levels["heading"] >= level do
6926     table.insert(buf, self.pop_attributes("heading"))
6927 end
6928
6929 -- construct attributes for the new section
6930 local auto_identifiers = {}
6931 if self.options.autoIdentifiers then
6932     table.insert(auto_identifiers, create_auto_identifier(flat_text))
6933 end
6934 if self.options.gfmAutoIdentifiers then
6935     table.insert(auto_identifiers,
6936         create_gfm_auto_identifier(flat_text))
6937 end
6938 local normalized_attributes = normalize_attributes(attributes,
6939     auto_identifiers)
6940
6941 -- push attributes for the new section
6942 local start_output = {}
6943 local end_output = {}
6944 table.insert(start_output, self.secbegin_text)
6945 table.insert(end_output, self.secend_text)
6946
6947 table.insert(buf, self.push_attributes("heading",
6948     normalized_attributes,
6949     start_output,
6950     end_output))
6951 assert(self.attribute_type_levels["heading"] == level)
6952
6953 -- render the heading and its attributes
6954 if self.is_writing and #normalized_attributes > 0 then
6955     table.insert(buf,
6956         "\\markdownRendererHeaderAttributeContextBegin\n")
6957     table.insert(buf, self.attributes(normalized_attributes, false))
6958 end
6959
6960 local cmd
6961 level = level + options.shiftHeadings

```

```

6962     if level <= 1 then
6963         cmd = "\\markdownRendererHeadingOne"
6964     elseif level == 2 then
6965         cmd = "\\markdownRendererHeadingTwo"
6966     elseif level == 3 then
6967         cmd = "\\markdownRendererHeadingThree"
6968     elseif level == 4 then
6969         cmd = "\\markdownRendererHeadingFour"
6970     elseif level == 5 then
6971         cmd = "\\markdownRendererHeadingFive"
6972     elseif level >= 6 then
6973         cmd = "\\markdownRendererHeadingSix"
6974     else
6975         cmd = ""
6976     end
6977     if self.is_writing then
6978         table.insert(buf, {cmd, "{", inlines, "}"})
6979     end
6980
6981     if self.is_writing and #normalized_attributes > 0 then
6982         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6983     end
6984
6985     return buf
6986 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.

```

6987     function self.get_state()
6988         return {
6989             is_writing=self.is_writing,
6990             flatten_inlines=self.flatten_inlines,
6991             active_attributes={table.unpack(self.active_attributes)},
6992         }
6993     end

```

Define `writer->set_state` as a function that restores the input state `s` and returns the previous state of the writer.

```

6994     function self.set_state(s)
6995         local previous_state = self.get_state()
6996         for key, value in pairs(s) do
6997             self[key] = value
6998         end
6999         return previous_state
7000     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`.

```
7001 function self.defer_call(f)
7002     local previous_state = self.get_state()
7003     return function(...)
7004         local state = self.set_state(previous_state)
7005         local return_value = f(...)
7006         self.set_state(state)
7007         return return_value
7008     end
7009 end
7010
7011 return self
7012 end
```

### 3.1.4 Parsers

The `parsers` hash table stores PEG patterns that are static and can be reused between different `reader` objects.

```
7013 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
7014 parsers.percent = P("%")
7015 parsers.at = P("@")
7016 parsers.comma = P(",")
7017 parsers.asterisk = P("*")
7018 parsers.dash = P("-")
7019 parsers.plus = P("+")
7020 parsers.underscore = P("_")
7021 parsers.period = P(".")
7022 parsers.hash = P("#")
7023 parsers.dollar = P("$")
7024 parsers.ampersand = P("&")
7025 parsers.backtick = P("`")
7026 parsers.less = P("<")
7027 parsers.more = P(">")
7028 parsers.space = P(" ")
7029 parsers.squote = P("'")
7030 parsers.dquote = P('"')
7031 parsers.lparent = P("(")
7032 parsers.rparent = P(")")
7033 parsers.lbracket = P("[")
7034 parsers.rbracket = P("]")
7035 parsers.lbrace = P("{")
```

```

7036 parsers.rbrace           = P("}")
7037 parsers.circumflex       = P("^")
7038 parsers.slash             = P("/")
7039 parsers.equal             = P("=")
7040 parsers.colon              = P(":")
7041 parsers.semicolon         = P(";")
7042 parsers.exclamation       = P("!")
7043 parsers.pipe               = P("|")
7044 parsers.tilde              = P("~")
7045 parsers.backslash         = P("\\")
7046 parsers.tab                = P("\t")
7047 parsers.newline           = P("\n")
7048
7049 parsers.digit              = R("09")
7050 parsers.hexdigit           = R("09","af","AF")
7051 parsers.letter            = R("AZ","az")
7052 parsers.alphanumeric      = R("AZ","az","09")
7053 parsers.keyword           = parsers.letter
7054                           * (parsers.alphanumeric + parsers.dash)^0
7055
7056 parsers.doubleasterisks    = P("**")
7057 parsers.doubleunderscores  = P("__")
7058 parsers.doubletildes      = P("~~")
7059 parsers.fourspace         = P("  ")
7060
7061 parsers.any                 = P(1)
7062 parsers.succeed            = P(true)
7063 parsers.fail                = P(false)
7064
7065 parsers.internal_punctuation = S(":,.?")
7066 parsers.ascii_punctuation  = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
7067

```

### 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>.

```

7068 ;(function()
7069   local pathname = assert(kpse.find_file("UnicodeData.txt"),
7070     [[Could not locate file "UnicodeData.txt"]])
7071   local file = assert(io.open(pathname, "r"),
7072     [[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.

```

7073   local prefix_trees = {}
7074   for line in file:lines() do
7075     local codepoint, major_category = line:match("^(%x+);[~;]*;(%)a")
7076     if major_category == "P" or major_category == "S" then
7077       local code = unicode.utf8.char(tonumber(codepoint, 16))
7078       if prefix_trees[#code] == nil then
7079         prefix_trees[#code] = {}
7080       end
7081       local node = prefix_trees[#code]
7082       for i = 1, #code do
7083         local byte = code:sub(i, i)
7084         if i < #code then
7085           if node[byte] == nil then
7086             node[byte] = {}
7087           end
7088           node = node[byte]
7089         else
7090           table.insert(node, byte)
7091         end
7092       end
7093     end
7094   end
7095   assert(file:close())
7096

```

Next, we will construct a parser out of the prefix tree.

```

7097   local function depth_first_search(node, path, visit, leave)
7098     visit(node, path)
7099     for label, child in pairs(node) do
7100       if type(child) == "table" then
7101         depth_first_search(child, path .. label, visit, leave)
7102       else
7103         visit(child, path)
7104       end
7105     end
7106     leave(node, path)
7107   end
7108
7109   print("M.punctuation = {}")
7110   print("local P = lpeg.P")

```



```

7111 print("-- luacheck: push no max line length")
7112 for length, prefix_tree in pairs(prefix_trees) do
7113     local subparsers = {}
7114     depth_first_search(prefix_tree, "", function(node, path)
7115         if type(node) == "string" then
7116             local suffix
7117             if node == "]" then
7118                 suffix = "P('" .. node .. "')"
7119             else
7120                 suffix = "P([[ " .. node .. "]])"
7121             end
7122             if subparsers[path] ~= nil then
7123                 subparsers[path] = subparsers[path] .. " + " .. suffix
7124             else
7125                 subparsers[path] = suffix
7126             end
7127         end
7128     end, function(_, path)
7129         if #path > 0 then
7130             local byte = path:sub(#path, #path)
7131             local parent_path = path:sub(1, #path-1)
7132             local prefix
7133             if byte == "]" then
7134                 prefix = "P('" .. byte .. "')"
7135             else
7136                 prefix = "P([[ " .. byte .. "]])"
7137             end
7138             local suffix
7139             if subparsers[path]:find(" %+ ") then
7140                 suffix = prefix .. " * (" .. subparsers[path] .. ")"
7141             else
7142                 suffix = prefix .. " * " .. subparsers[path]
7143             end
7144             if subparsers[parent_path] ~= nil then
7145                 subparsers[parent_path] = subparsers[parent_path]
7146                     .. " + " .. suffix
7147             else
7148                 subparsers[parent_path] = suffix
7149             end
7150         else
7151             print("M.punctuation[" .. length .. "] = " .. subparsers[path])
7152         end
7153     end)
7154 end
7155 print("-- luacheck: pop")
7156 end)()
7157 print("return M")

```

Back in the Markdown package, we will load the precompiled parser of Unicode punctuation.

```
7158 local unicode_data = require("markdown-unicode-data")
7159 if metadata.version ~= unicode_data.metadata.version then
7160   util.warning(
7161     "markdown.lua " .. metadata.version .. " used with " ..
7162     "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
7163   )
7164 end
7165 parsers.punctuation = unicode_data.punctuation
7166
7167 parsers.escapable           = parsers.ascii_punctuation
7168 parsers.anyescaped         = parsers.backslash / ""
7169                             * parsers.escapable
7170                             + parsers.any
7171
7172 parsers.spacechar          = S("\t ")
7173 parsers.spacing            = S(" \n\r\t")
7174 parsers.nonpacechar        = parsers.any - parsers.spacing
7175 parsers.optionalspace     = parsers.spacechar^0
7176
7177 parsers.normalchar         = parsers.any - (V("SpecialChar")
7178   + parsers.spacing)
7179 parsers.eof                 = -parsers.any
7180 parsers.nonindentspace     = parsers.space^-3 * - parsers.spacechar
7181 parsers.indent             = parsers.space^-3 * parsers.tab
7182                             + parsers.fourspace / ""
7183 parsers.linechar           = P(1 - parsers.newline)
7184
7185 parsers.blankline          = parsers.optionalspace
7186                             * parsers.newline / "\n"
7187 parsers.blanklines         = parsers.blankline^0
7188 parsers.skipblanklines     = ( parsers.optionalspace
7189                               * parsers.newline)^0
7190 parsers.indentedline       = parsers.indent / ""
7191                             * C( parsers.linechar^1
7192                               * parsers.newline^-1)
7193 parsers.optionallyindentedline = parsers.indent^-1 / ""
7194                             * C( parsers.linechar^1
7195                               * parsers.newline^-1)
7196 parsers.sp                 = parsers.spacing^0
7197 parsers.spnl               = parsers.optionalspace
7198                             * ( parsers.newline
7199                               * parsers.optionalspace)^-1
7200 parsers.line               = parsers.linechar^0 * parsers.newline
7201 parsers.nonemptyline       = parsers.line - parsers.blankline
```

### 3.1.5.1 Parsers Used for Indentation

```
7202
7203 parsers.leader      = parsers.space^-3
7204
```

Check if a trail exists and is non-empty in the indent table `indent_table`.

```
7205 local function has_trail(indent_table)
7206   return indent_table ~= nil and
7207     indent_table.trail ~= nil and
7208     next(indent_table.trail) ~= nil
7209 end
7210
```

Check if indent table `indent_table` has any indents.

```
7211 local function has_indents(indent_table)
7212   return indent_table ~= nil and
7213     indent_table.indents ~= nil and
7214     next(indent_table.indents) ~= nil
7215 end
7216
```

Add a trail `trail_info` to the indent table `indent_table`.

```
7217 local function add_trail(indent_table, trail_info)
7218   indent_table.trail = trail_info
7219   return indent_table
7220 end
7221
```

Remove a trail `trail_info` from the indent table `indent_table`.

```
7222 local function remove_trail(indent_table)
7223   indent_table.trail = nil
7224   return indent_table
7225 end
7226
```

Update the indent table `indent_table` by adding or removing a new indent `add`.

```
7227 local function update_indent_table(indent_table, new_indent, add)
7228   indent_table = remove_trail(indent_table)
7229
7230   if not has_indents(indent_table) then
7231     indent_table.indents = {}
7232   end
7233
7234
7235   if add then
7236     indent_table.indents[#indent_table.indents + 1] = new_indent
7237   else
7238     if indent_table.indents[#indent_table.indents].name
7239       == new_indent.name then
```

```

7240     indent_table.indents[#indent_table.indents] = nil
7241   end
7242 end
7243
7244 return indent_table
7245 end
7246

```

Remove an indent by its name `name`.

```

7247 local function remove_indent(name)
7248   local remove_indent_level =
7249     function(s, i, indent_table) -- luacheck: ignore s i
7250       indent_table = update_indent_table(indent_table, {name=name},
7251   false)
7252       return true, indent_table
7253     end
7254
7255   return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
7256 end
7257

```

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`.

```

7258 local function process_starter_spacing(indent, spacing,
7259                                     minimum, left_strip_length)
7260   left_strip_length = left_strip_length or 0
7261
7262   local count = 0
7263   local tab_value = 4 - (indent) % 4
7264
7265   local code_started, minimum_found = false, false
7266   local code_start, minimum_remainder = "", ""
7267
7268   local left_total_stripped = 0
7269   local full_remainder = ""
7270
7271   if spacing ~= nil then
7272     for i = 1, #spacing do
7273       local character = spacing:sub(i, i)
7274
7275       if character == "\t" then
7276         count = count + tab_value
7277         tab_value = 4
7278       elseif character == " " then

```

```

7279         count = count + 1
7280         tab_value = 4 - (1 - tab_value) % 4
7281     end
7282
7283     if (left_strip_length ~= 0) then
7284         local possible_to_strip = math.min(count, left_strip_length)
7285         count = count - possible_to_strip
7286         left_strip_length = left_strip_length - possible_to_strip
7287         left_total_stripped = left_total_stripped + possible_to_strip
7288     else
7289         full_remainder = full_remainder .. character
7290     end
7291
7292     if (minimum_found) then
7293         minimum_remainder = minimum_remainder .. character
7294     elseif (count >= minimum) then
7295         minimum_found = true
7296         minimum_remainder = minimum_remainder
7297             .. string.rep(" ", count - minimum)
7298     end
7299
7300     if (code_started) then
7301         code_start = code_start .. character
7302     elseif (count >= minimum + 4) then
7303         code_started = true
7304         code_start = code_start
7305             .. string.rep(" ", count - (minimum + 4))
7306     end
7307 end
7308 end
7309
7310 local remainder
7311 if (code_started) then
7312     remainder = code_start
7313 else
7314     remainder = string.rep(" ", count - minimum)
7315 end
7316
7317 local is_minimum = count >= minimum
7318 return {
7319     is_code = code_started,
7320     remainder = remainder,
7321     left_total_stripped = left_total_stripped,
7322     is_minimum = is_minimum,
7323     minimum_remainder = minimum_remainder,
7324     total_length = count,
7325     full_remainder = full_remainder

```

```

7326 }
7327 end
7328

```

Count the total width of all indents in the indent table `indent_table`.

```

7329 local function count_indent_tab_level(indent_table)
7330   local count = 0
7331   if not has_indents(indent_table) then
7332     return count
7333   end
7334
7335   for i=1, #indent_table.indents do
7336     count = count + indent_table.indents[i].length
7337   end
7338   return count
7339 end
7340

```

Count the total width of a delimiter `delimiter`.

```

7341 local function total_delimiter_length(delimiter)
7342   local count = 0
7343   if type(delimiter) == "string" then return #delimiter end
7344   for _, value in pairs(delimiter) do
7345     count = count + total_delimiter_length(value)
7346   end
7347   return count
7348 end
7349

```

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`.

```

7350 local function process_starter_indent(_, _, indent_table, starter,
7351                                     is_blank, indent_type, breakable)
7352   local last_trail = starter[1]
7353   local delimiter = starter[2]
7354   local raw_new_trail = starter[3]
7355
7356   if indent_type == "bq" and not breakable then
7357     indent_table.ignore_blockquote_blank = true
7358   end
7359
7360   if has_trail(indent_table) then
7361     local trail = indent_table.trail
7362     if trail.is_code then
7363       return false
7364     end
7365     last_trail = trail.remainder
7366   else

```

```

7367     local sp = process_starter_spacing(0, last_trail, 0, 0)
7368
7369     if sp.is_code then
7370         return false
7371     end
7372     last_trail = sp.remainder
7373 end
7374
7375 local preceding_indentation = count_indent_tab_level(indent_table) % 4
7376 local last_trail_length = #last_trail
7377 local delimiter_length = total_delimiter_length(delimiter)
7378
7379 local total_indent_level = preceding_indentation + last_trail_length
7380                        + delimiter_length
7381
7382 local sp = {}
7383 if not is_blank then
7384     sp = process_starter_spacing(total_indent_level, raw_new_trail,
7385                                0, 1)
7386 end
7387
7388 local del_trail_length = sp.left_total_stripped
7389 if is_blank then
7390     del_trail_length = 1
7391 elseif not sp.is_code then
7392     del_trail_length = del_trail_length + #sp.remainder
7393 end
7394
7395 local indent_length = last_trail_length + delimiter_length
7396                        + del_trail_length
7397 local new_indent_info = {name=indent_type, length=indent_length}
7398
7399 indent_table = update_indent_table(indent_table, new_indent_info,
7400                                    true)
7401 indent_table = add_trail(indent_table,
7402                          {is_code=sp.is_code,
7403                           remainder=sp.remainder,
7404                           total_length=sp.total_length,
7405                           full_remainder=sp.full_remainder})
7406
7407 return true, indent_table
7408 end
7409

```

Return the pattern corresponding with the indent name [name](#).

```

7410 local function decode_pattern(name)
7411     local delimiter = parsers.succeed
7412     if name == "bq" then

```

```

7413     delimiter = parsers.more
7414 end
7415
7416 return C(parsers.optionalspace) * C(delimiter)
7417     * C(parsers.optionalspace) * Cp()
7418 end
7419

```

Find the first blank-only indent of the indent table `indent_table` followed by blank-only indents.

```

7420 local function left_blank_starter(indent_table)
7421     local blank_starter_index
7422
7423     if not has_indents(indent_table) then
7424         return
7425     end
7426
7427     for i = #indent_table.indents,1,-1 do
7428         local value = indent_table.indents[i]
7429         if value.name == "li" then
7430             blank_starter_index = i
7431         else
7432             break
7433         end
7434     end
7435
7436     return blank_starter_index
7437 end
7438

```

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.

```

7439 local function traverse_indent(s, i, indent_table, is_optional,
7440                               is_blank, current_line_indents)
7441     local new_index = i
7442
7443     local preceding_indentation = 0
7444     local current_trail = {}
7445
7446     local blank_starter = left_blank_starter(indent_table)
7447
7448     if current_line_indents == nil then
7449         current_line_indents = {}
7450     end

```



```

7451
7452 for index = 1,#indent_table.indents do
7453     local value = indent_table.indents[index]
7454     local pattern = decode_pattern(value.name)
7455
7456     -- match decoded pattern
7457     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7458     if new_indent_info == nil then
7459         local blankline_end = lpeg.match(
7460             Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7461         if is_optional or not indent_table.ignore_blockquote_blank
7462             or not blankline_end then
7463             return is_optional, new_index, current_trail,
7464                 current_line_indents
7465         end
7466
7467         return traverse_indent(s, tonumber(blankline_end.pos),
7468             indent_table, is_optional, is_blank,
7469             current_line_indents)
7470     end
7471
7472     local raw_last_trail = new_indent_info[1]
7473     local delimiter = new_indent_info[2]
7474     local raw_new_trail = new_indent_info[3]
7475     local next_index = new_indent_info[4]
7476
7477     local space_only = delimiter == ""
7478
7479     -- check previous trail
7480     if not space_only and next(current_trail) == nil then
7481         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7482         current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7483             total_length=sp.total_length,
7484             full_remainder=sp.full_remainder}
7485     end
7486
7487     if next(current_trail) ~= nil then
7488         if not space_only and current_trail.is_code then
7489             return is_optional, new_index, current_trail,
7490                 current_line_indents
7491         end
7492         if current_trail.internal_remainder ~= nil then
7493             raw_last_trail = current_trail.internal_remainder
7494         end
7495     end
7496
7497     local raw_last_trail_length = 0

```

```

7498     local delimiter_length = 0
7499
7500     if not space_only then
7501         delimiter_length = #delimiter
7502         raw_last_trail_length = #raw_last_trail
7503     end
7504
7505     local total_indent_level = preceding_indentation
7506                             + raw_last_trail_length + delimiter_length
7507
7508     local spacing_to_process
7509     local minimum = 0
7510     local left_strip_length = 0
7511
7512     if not space_only then
7513         spacing_to_process = raw_new_trail
7514         left_strip_length = 1
7515     else
7516         spacing_to_process = raw_last_trail
7517         minimum = value.length
7518     end
7519
7520     local sp = process_starter_spacing(total_indent_level,
7521                                     spacing_to_process, minimum,
7522                                     left_strip_length)
7523
7524     if space_only and not sp.is_minimum then
7525         return is_optional or (is_blank and blank_starter <= index),
7526             new_index, current_trail, current_line_indents
7527     end
7528
7529     local indent_length = raw_last_trail_length + delimiter_length
7530                       + sp.left_total_stripped
7531
7532     -- update info for the next pattern
7533     if not space_only then
7534         preceding_indentation = preceding_indentation + indent_length
7535     else
7536         preceding_indentation = preceding_indentation + value.length
7537     end
7538
7539     current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7540                   internal_remainder=sp.minimum_remainder,
7541                   total_length=sp.total_length,
7542                   full_remainder=sp.full_remainder}
7543
7544     current_line_indents[#current_line_indents + 1] = new_indent_info

```

```

7545     new_index = next_index
7546 end
7547
7548 return true, new_index, current_trail, current_line_indents
7549 end
7550

```

Check if a code trail is expected.

```

7551 local function check_trail(expect_code, is_code)
7552     return (expect_code and is_code) or (not expect_code and not is_code)
7553 end
7554

```

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`.

```

7555 local check_trail_joined =
7556     function(s, i, indent_table, -- luacheck: ignore s i
7557             spacing, expect_code, omit_remainder)
7558         local is_code
7559         local remainder
7560
7561         if has_trail(indent_table) then
7562             local trail = indent_table.trail
7563             is_code = trail.is_code
7564             if is_code then
7565                 remainder = trail.remainder
7566             else
7567                 remainder = trail.full_remainder
7568             end
7569         else
7570             local sp = process_starter_spacing(0, spacing, 0, 0)
7571             is_code = sp.is_code
7572             if is_code then
7573                 remainder = sp.remainder
7574             else
7575                 remainder = sp.full_remainder
7576             end
7577         end
7578
7579         local result = check_trail(expect_code, is_code)
7580         if omit_remainder then
7581             return result
7582         end
7583         return result, remainder
7584     end
7585

```

Check if the current trail of the `indent_table` is of length between `min` and `max`.

```
7586 local check_trail_length =
7587   function(s, i, indent_table, -- luacheck: ignore s i
7588     spacing, min, max)
7589     local trail
7590
7591     if has_trail(indent_table) then
7592       trail = indent_table.trail
7593     else
7594       trail = process_starter_spacing(0, spacing, 0, 0)
7595     end
7596
7597     local total_length = trail.total_length
7598     if total_length == nil then
7599       return false
7600     end
7601
7602     return min <= total_length and total_length <= max
7603   end
7604
```

Check the indentation of the continuation line, optionally with the mode `is_optional` selected. Check blank line exclusively with `is_blank`.

```
7605 local function check_continuation_indentation(s, i, indent_table,
7606   is_optional, is_blank)
7607   if not has_indents(indent_table) then
7608     return true
7609   end
7610
7611   local passes, new_index, current_trail, current_line_indents =
7612     traverse_indent(s, i, indent_table, is_optional, is_blank)
7613
7614   if passes then
7615     indent_table.current_line_indents = current_line_indents
7616     indent_table = add_trail(indent_table, current_trail)
7617     return new_index, indent_table
7618   end
7619   return false
7620 end
7621
```

Get name of the last indent from the `indent_table`.

```
7622 local function get_last_indent_name(indent_table)
7623   if has_indents(indent_table) then
7624     return indent_table.indents[#indent_table.indents].name
7625   end
7626 end
7627
```

Remove the remainder altogether if the last indent from the `indent_table` is blank-only.

```
7628 local function remove_remainder_if_blank(indent_table, remainder)
7629   if get_last_indent_name(indent_table) == "li" then
7630     return ""
7631   end
7632   return remainder
7633 end
7634
```

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.

```
7635 local check_trail_type =
7636   function(s, i, -- luacheck: ignore s i
7637     trail, spacing, trail_type)
7638     if trail == nil then
7639       trail = process_starter_spacing(0, spacing, 0, 0)
7640     end
7641
7642     if trail_type == "non-code" then
7643       return check_trail(false, trail.is_code)
7644     end
7645     if trail_type == "code" then
7646       return check_trail(true, trail.is_code)
7647     end
7648     if trail_type == "full-code" then
7649       if (trail.is_code) then
7650         return i, trail.remainder
7651       end
7652       return i, ""
7653     end
7654     if trail_type == "full-any" then
7655       return i, trail.internal_remainder
7656     end
7657   end
7658
```

Stores or restores an `is_freezing` trail from indent table `indent_table`.

```
7659 local trail_freezing =
7660   function(s, i, -- luacheck: ignore s i
7661     indent_table, is_freezing)
7662     if is_freezing then
7663       if indent_table.is_trail_frozen then
7664         indent_table.trail = indent_table.frozen_trail
7665       else
7666         indent_table.frozen_trail = indent_table.trail
7667         indent_table.is_trail_frozen = true
7668       end
7669     end
7670
```

```

7668     end
7669   else
7670     indent_table.frozen_trail = nil
7671     indent_table.is_trail_frozen = false
7672   end
7673   return true, indent_table
7674 end
7675

```

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`.

```

7676 local check_continuation_indentation_and_trail =
7677 function (s, i, indent_table, is_optional, is_blank, trail_type,
7678         reset_rem, omit_remainder)
7679   if not has_indents(indent_table) then
7680     local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
7681   * Cp(), s, i)
7682     local result, remainder = check_trail_type(s, i,
7683       indent_table.trail, spacing, trail_type)
7684     if remainder == nil then
7685       if result then
7686         return new_index
7687       end
7688       return false
7689     end
7690     if result then
7691       return new_index, remainder
7692     end
7693     return false
7694   end
7695
7696   local passes, new_index, current_trail = traverse_indent(s, i,
7697     indent_table, is_optional, is_blank)
7698
7699   if passes then
7700     local spacing
7701     if current_trail == nil then
7702       local newer_spacing, newer_index = lpeg.match(
7703         C(parsers.spacechar^0) * Cp(), s, i)
7704       current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7705       new_index = newer_index
7706       spacing = newer_spacing
7707     else
7708       spacing = current_trail.remainder
7709     end
7710     local result, remainder = check_trail_type(s, new_index,

```

```

7711     current_trail, spacing, trail_type)
7712   if remainder == nil or omit_remainder then
7713     if result then
7714       return new_index
7715     end
7716     return false
7717   end
7718
7719   if is_blank and reset_rem then
7720     remainder = remove_remainder_if_blank(indent_table, remainder)
7721   end
7722   if result then
7723     return new_index, remainder
7724   end
7725   return false
7726 end
7727 return false
7728 end
7729

```

The following patterns check whitespace indentation at the start of a block.

```

7730 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)
7731     * Cc(false), check_trail_joined)
7732
7733 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7734     * C(parsers.spacechar^0) * Cc(false)
7735     * Cc(true), check_trail_joined)
7736
7737 parsers.check_code_trail = Cmt( Cb("indent_info")
7738     * C(parsers.spacechar^0)
7739     * Cc(true), check_trail_joined)
7740
7741 parsers.check_trail_length_range = function(min, max)
7742   return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7743     * Cc(max), check_trail_length)
7744 end
7745
7746 parsers.check_trail_length = function(n)
7747   return parsers.check_trail_length_range(n, n)
7748 end
7749

```

The following patterns handle trail backup, to prevent a failing pattern to modify it before passing it to the next.

```

7750 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
7751     * Cc(true), trail_freezing), "indent_info")
7752
7753 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),

```

```
7754         trail_freezing), "indent_info")
7755
```

The following patterns check indentation in continuation lines as defined by the container start.

```
7756 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7757                                     check_continuation_indentation)
7758
7759 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7760                                     check_continuation_indentation)
7761
7762 parsers.check_minimal_blank_indent
7763 = Cmt( Cb("indent_info") * Cc(false)
7764       * Cc(true)
7765       , check_continuation_indentation)
7766
```

The following patterns check indentation in continuation lines as defined by the container start. Additionally the subsequent trail is also directly checked.

```
7767
7768 parsers.check_minimal_indent_and_trail =
7769   Cmt( Cb("indent_info")
7770       * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7771       , check_continuation_indentation_and_trail)
7772
7773 parsers.check_minimal_indent_and_code_trail =
7774   Cmt( Cb("indent_info")
7775       * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7776       , check_continuation_indentation_and_trail)
7777
7778 parsers.check_minimal_blank_indent_and_full_code_trail =
7779   Cmt( Cb("indent_info")
7780       * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7781       , check_continuation_indentation_and_trail)
7782
7783 parsers.check_minimal_indent_and_any_trail =
7784   Cmt( Cb("indent_info")
7785       * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7786       , check_continuation_indentation_and_trail)
7787
7788 parsers.check_minimal_blank_indent_and_any_trail =
7789   Cmt( Cb("indent_info")
7790       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7791       , check_continuation_indentation_and_trail)
7792
7793 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7794   Cmt( Cb("indent_info")
7795       * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
```



```

7796     , check_continuation_indentation_and_trail)
7797
7798 parsers.check_optional_indent_and_any_trail =
7799   Cmt( Cb("indent_info")
7800     * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7801     , check_continuation_indentation_and_trail)
7802
7803 parsers.check_optional_blank_indent_and_any_trail =
7804   Cmt( Cb("indent_info")
7805     * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7806     , check_continuation_indentation_and_trail)
7807

```

The following patterns specify behaviour around newlines.

```

7808
7809 parsers.spnlc_noexc = parsers.optionalspace
7810                       * ( parsers.newline
7811                       * parsers.check_minimal_indent_and_any_trail)^-1
7812
7813 parsers.spnlc = parsers.optionalspace
7814               * (V("EndlineNoSub"))^-1
7815
7816 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7817                  + parsers.spacechar^1
7818
7819 parsers.only_blank = parsers.spacechar^0
7820                   * (parsers.newline + parsers.eof)
7821

```

The `parsers.commented\_line^1` parser recognizes the regular language of T<sub>E</sub>X comments, see an equivalent finite automaton in Figure 8.

```

7822 parsers.commented_line_letter = parsers.linechar
7823                               + parsers.newline
7824                               - parsers.backslash
7825                               - parsers.percent
7826 parsers.commented_line = Cg(Cc(""), "backslashes")
7827                       * ((#(parsers.commented_line_letter
7828                           - parsers.newline)
7829                          * Cb("backslashes")
7830                          * Cs(parsers.commented_line_letter
7831                              - parsers.newline)^1 -- initial
7832                          * Cg(Cc(""), "backslashes"))
7833                       + #( parsers.backslash
7834                           * (parsers.backslash + parsers.newline))
7835                       * Cg((parsers.backslash -- even backslash
7836                           * ( parsers.backslash
7837                               + #parsers.newline))^1, "backslashes")
7838                       + (parsers.backslash

```



```

7839         * (#parsers.percent
7840         * Cb("backslashes")
7841         / function(backslashes)
7842         return string.rep("\\", #backslashes / 2)
7843         end
7844         * C(parsers.percent)
7845         + #parsers.commented_line_letter
7846         * Cb("backslashes")
7847         * Cc("\\")
7848         * C(parsers.commented_line_letter))
7849         * Cg(Cc(""), "backslashes"))^0
7850     * (#parsers.percent
7851     * Cb("backslashes")
7852     / function(backslashes)
7853     return string.rep("\\", #backslashes / 2)
7854     end
7855     * ((parsers.percent -- comment
7856     * parsers.line
7857     * #parsers.blankline) -- blank line
7858     / "\n"
7859     + parsers.percent -- comment
7860     * parsers.line
7861     * parsers.optionalspace) -- leading spaces
7862     + #parsers.newline)
7863     * Cb("backslashes")
7864     * C(parsers.newline))
7865
7866     parsers.chunk = parsers.line * (parsers.optionallyindentedline
7867     - parsers.blankline)^0
7868
7869     parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7870     parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7871     parsers.attribute_key = (parsers.attribute_key_char
7872     - parsers.dash - parsers.digit)
7873     * parsers.attribute_key_char^0
7874     parsers.attribute_value = ( (parsers.dquote / "")
7875     * (parsers.anyescaped - parsers.dquote)^0
7876     * (parsers.dquote / ""))
7877     + ( (parsers.squote / "")
7878     * (parsers.anyescaped - parsers.squote)^0
7879     * (parsers.squote / ""))
7880     + ( parsers.anyescaped
7881     - parsers.dquote
7882     - parsers.rbrace
7883     - parsers.space)^0
7884     parsers.attribute_identifier = parsers.attribute_key_char^1
7885     parsers.attribute_classname = parsers.letter

```

```

7886             * parsers.attribute_key_char^0
7887 parsers.attribute_raw = parsers.attribute_raw_char^1
7888
7889 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7890             + C( parsers.hash
7891                 * parsers.attribute_identifier)
7892             + C( parsers.period
7893                 * parsers.attribute_classname)
7894             + Cs( parsers.attribute_key
7895                 * parsers.optionalspace
7896                 * parsers.equal
7897                 * parsers.optionalspace
7898                 * parsers.attribute_value)
7899 parsers.attributes = parsers.lbrace
7900             * parsers.optionalspace
7901             * parsers.attribute
7902             * (parsers.spacechar^1
7903                 * parsers.attribute)^0
7904             * parsers.optionalspace
7905             * parsers.rbrace
7906
7907 parsers.raw_attribute = parsers.lbrace
7908             * parsers.optionalspace
7909             * parsers.equal
7910             * C(parsers.attribute_raw)
7911             * parsers.optionalspace
7912             * parsers.rbrace
7913
7914 -- block followed by 0 or more optionally
7915 -- indented blocks with first line indented.
7916 parsers.indented_blocks = function(bl)
7917   return Cs( bl
7918             * ( parsers.blankline^1
7919                 * parsers.indent
7920                 * -parsers.blankline
7921                 * bl)^0
7922             * (parsers.blankline^1 + parsers.eof) )
7923 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

7924 local function repeat_between(pattern, min, max)
7925   return -pattern^(max + 1) * pattern^min
7926 end
7927
7928 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7929             * C(repeat_between(parsers.hexdigit, 1, 6))

```

```

7930             * parsers.semicolon
7931 parsers.decentity = parsers.ampersand * parsers.hash
7932             * C(repeat_between(parsers.digit, 1, 7))
7933             * parsers.semicolon
7934 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7935             * parsers.semicolon
7936
7937 parsers.html_entities
7938 = parsers.hexentity / entities.hex_entity_with_x_char
7939 + parsers.decentity / entities.dec_entity
7940 + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

7941 parsers.bullet = function(bullet_char, interrupting)
7942   local allowed_end
7943   if interrupting then
7944     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7945   else
7946     allowed_end = C(parsers.spacechar^1)
7947                 + #(parsers.newline + parsers.eof)
7948   end
7949   return parsers.check_trail
7950         * Ct(C(bullet_char) * Cc(""))
7951         * allowed_end
7952 end
7953
7954 local function tickbox(interior)
7955   return parsers.optionalspace * parsers.lbracket
7956         * interior * parsers.rbracket * parsers.spacechar^1
7957 end
7958
7959 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7960 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7961 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7962

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```

7963 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7964
7965 local function captures_equal_length(_,i,a,b)
7966   return #a == #b and i
7967 end
7968
7969 parsers.closeticks = Cmt(C(parsers.backtick^1)
7970                          * Cb("ticks"), captures_equal_length)
7971

```

```

7972 parsers.intickschar = (parsers.any - S("\n\r`"))
7973                       + V("NoSoftLineBreakEndline")
7974                       + (parsers.backtick^1 - parsers.closeticks)
7975
7976 local function process_inticks(s)
7977   s = s:gsub("\n", " ")
7978   s = s:gsub("^ (.*) $", "%1")
7979   return s
7980 end
7981
7982 parsers.inticks = parsers.openticks
7983                 * C(parsers.space^0)
7984                 * parsers.closeticks
7985                 + parsers.openticks
7986                 * Cs(Cs(parsers.intickschar^0) / process_inticks)
7987                 * parsers.closeticks
7988

```

### 3.1.5.5 Parsers Used for HTML

```

7989 -- case-insensitive match (we assume s is lowercase)
7990 -- must be single byte encoding
7991 parsers.keyword_exact = function(s)
7992   local parser = P(0)
7993   for i=1,#s do
7994     local c = s:sub(i,i)
7995     local m = c .. upper(c)
7996     parser = parser * S(m)
7997   end
7998   return parser
7999 end
8000
8001 parsers.special_block_keyword =
8002   parsers.keyword_exact("pre") +
8003   parsers.keyword_exact("script") +
8004   parsers.keyword_exact("style") +
8005   parsers.keyword_exact("textarea")
8006
8007 parsers.block_keyword =
8008   parsers.keyword_exact("address") +
8009   parsers.keyword_exact("article") +
8010   parsers.keyword_exact("aside") +
8011   parsers.keyword_exact("base") +
8012   parsers.keyword_exact("basefont") +
8013   parsers.keyword_exact("blockquote") +
8014   parsers.keyword_exact("body") +
8015   parsers.keyword_exact("caption") +

```

```
8016 parsers.keyword_exact("center") +
8017 parsers.keyword_exact("col") +
8018 parsers.keyword_exact("colgroup") +
8019 parsers.keyword_exact("dd") +
8020 parsers.keyword_exact("details") +
8021 parsers.keyword_exact("dialog") +
8022 parsers.keyword_exact("dir") +
8023 parsers.keyword_exact("div") +
8024 parsers.keyword_exact("dl") +
8025 parsers.keyword_exact("dt") +
8026 parsers.keyword_exact("fieldset") +
8027 parsers.keyword_exact("figcaption") +
8028 parsers.keyword_exact("figure") +
8029 parsers.keyword_exact("footer") +
8030 parsers.keyword_exact("form") +
8031 parsers.keyword_exact("frame") +
8032 parsers.keyword_exact("frameset") +
8033 parsers.keyword_exact("h1") +
8034 parsers.keyword_exact("h2") +
8035 parsers.keyword_exact("h3") +
8036 parsers.keyword_exact("h4") +
8037 parsers.keyword_exact("h5") +
8038 parsers.keyword_exact("h6") +
8039 parsers.keyword_exact("head") +
8040 parsers.keyword_exact("header") +
8041 parsers.keyword_exact("hr") +
8042 parsers.keyword_exact("html") +
8043 parsers.keyword_exact("iframe") +
8044 parsers.keyword_exact("legend") +
8045 parsers.keyword_exact("li") +
8046 parsers.keyword_exact("link") +
8047 parsers.keyword_exact("main") +
8048 parsers.keyword_exact("menu") +
8049 parsers.keyword_exact("menuitem") +
8050 parsers.keyword_exact("nav") +
8051 parsers.keyword_exact("noframes") +
8052 parsers.keyword_exact("ol") +
8053 parsers.keyword_exact("optgroup") +
8054 parsers.keyword_exact("option") +
8055 parsers.keyword_exact("p") +
8056 parsers.keyword_exact("param") +
8057 parsers.keyword_exact("section") +
8058 parsers.keyword_exact("source") +
8059 parsers.keyword_exact("summary") +
8060 parsers.keyword_exact("table") +
8061 parsers.keyword_exact("tbody") +
8062 parsers.keyword_exact("td") +
```

```

8063     parsers.keyword_exact("tfoot") +
8064     parsers.keyword_exact("th") +
8065     parsers.keyword_exact("thead") +
8066     parsers.keyword_exact("title") +
8067     parsers.keyword_exact("tr") +
8068     parsers.keyword_exact("track") +
8069     parsers.keyword_exact("ul")
8070
8071 -- end conditions
8072 parsers.html_blankline_end_condition
8073 = parsers.linechar^0
8074 * ( parsers.newline
8075     * (parsers.check_minimal_blank_indent_and_any_trail
8076         * #parsers.blankline
8077         + parsers.check_minimal_indent_and_any_trail)
8078     * parsers.linechar^1)^0
8079 * (parsers.newline^-1 / "")
8080
8081 local function remove_trailing_blank_lines(s)
8082 return s:gsub("[\n\r]+%s*$", "")
8083 end
8084
8085 parsers.html_until_end = function(end_marker)
8086 return Cs(Cs((parsers.newline
8087     * (parsers.check_minimal_blank_indent_and_any_trail
8088     * #parsers.blankline
8089     + parsers.check_minimal_indent_and_any_trail)
8090     + parsers.linechar - end_marker)^0
8091     * parsers.linechar^0 * parsers.newline^-1)
8092     / remove_trailing_blank_lines)
8093 end
8094
8095 -- attributes
8096 parsers.html_attribute_spacing = parsers.optionalspace
8097     * V("NoSoftLineBreakEndline")
8098     * parsers.optionalspace
8099     + parsers.spacechar^1
8100
8101 parsers.html_attribute_name = ( parsers.letter
8102     + parsers.colon
8103     + parsers.underscore)
8104     * ( parsers.alphanumeric
8105     + parsers.colon
8106     + parsers.underscore
8107     + parsers.period
8108     + parsers.dash)^0
8109

```



```

8110 parsers.html_attribute_value = parsers.squote
8111     * (parsers.linechar - parsers.squote)^0
8112     * parsers.squote
8113     + parsers.dquote
8114     * (parsers.linechar - parsers.dquote)^0
8115     * parsers.dquote
8116     + ( parsers.any
8117         - parsers.spacechar
8118         - parsers.newline
8119         - parsers.dquote
8120         - parsers.squote
8121         - parsers.backtick
8122         - parsers.equal
8123         - parsers.less
8124         - parsers.more)^1
8125
8126 parsers.html_inline_attribute_value = parsers.squote
8127     * (V("NoSoftLineBreakEndline")
8128         + parsers.any
8129         - parsers.blankline^2
8130         - parsers.squote)^0
8131     * parsers.squote
8132     + parsers.dquote
8133     * (V("NoSoftLineBreakEndline")
8134         + parsers.any
8135         - parsers.blankline^2
8136         - parsers.dquote)^0
8137     * parsers.dquote
8138     + (parsers.any
8139         - parsers.spacechar
8140         - parsers.newline
8141         - parsers.dquote
8142         - parsers.squote
8143         - parsers.backtick
8144         - parsers.equal
8145         - parsers.less
8146         - parsers.more)^1
8147
8148 parsers.html_attribute_value_specification
8149     = parsers.optionalspace
8150     * parsers.equal
8151     * parsers.optionalspace
8152     * parsers.html_attribute_value
8153
8154 parsers.html_spnl = parsers.optionalspace
8155     * (V("NoSoftLineBreakEndline")
8156     * parsers.optionalspace)^-1

```

```

8157
8158 parsers.html_inline_attribute_value_specification
8159     = parsers.html_spn1
8160     * parsers.equal
8161     * parsers.html_spn1
8162     * parsers.html_inline_attribute_value
8163
8164 parsers.html_attribute
8165     = parsers.html_attribute_spacing
8166     * parsers.html_attribute_name
8167     * parsers.html_inline_attribute_value_specification^-1
8168
8169 parsers.html_non_newline_attribute
8170     = parsers.spacechar^1
8171     * parsers.html_attribute_name
8172     * parsers.html_attribute_value_specification^-1
8173
8174 parsers.nested_breaking_blank = parsers.newline
8175                               * parsers.check_minimal_blank_indent
8176                               * parsers.blankline
8177
8178 parsers.html_comment_start = P("<!--")
8179
8180 parsers.html_comment_end = P("-->")
8181
8182 parsers.html_comment
8183     = Cs( parsers.html_comment_start
8184           * parsers.html_until_end(parsers.html_comment_end))
8185
8186 parsers.html_inline_comment = (parsers.html_comment_start / "")
8187                               * -P(">") * -P("->")
8188                               * Cs(( V("NoSoftLineBreakEndline")
8189                                     + parsers.any
8190                                     - parsers.nested_breaking_blank
8191                                     - parsers.html_comment_end)^0)
8192                               * (parsers.html_comment_end / "")
8193
8194 parsers.html_cdatasection_start = P("<![CDATA[")
8195
8196 parsers.html_cdatasection_end = P("]]>")
8197
8198 parsers.html_cdatasection
8199     = Cs( parsers.html_cdatasection_start
8200           * parsers.html_until_end(parsers.html_cdatasection_end))
8201
8202 parsers.html_inline_cdatasection
8203     = parsers.html_cdatasection_start

```

```

8204 * Cs(V("NoSoftLineBreakEndline") + parsers.any
8205     - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
8206 * parsers.html_cdatasection_end
8207
8208 parsers.html_declaration_start = P("<!") * parsers.letter
8209
8210 parsers.html_declaration_end = P(">")
8211
8212 parsers.html_declaration
8213   = Cs( parsers.html_declaration_start
8214       * parsers.html_until_end(parsers.html_declaration_end))
8215
8216 parsers.html_inline_declaration
8217   = parsers.html_declaration_start
8218     * Cs(V("NoSoftLineBreakEndline") + parsers.any
8219         - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
8220     * parsers.html_declaration_end
8221
8222 parsers.html_instruction_start = P("<?")
8223
8224 parsers.html_instruction_end = P("?>")
8225
8226 parsers.html_instruction
8227   = Cs( parsers.html_instruction_start
8228       * parsers.html_until_end(parsers.html_instruction_end))
8229
8230 parsers.html_inline_instruction = parsers.html_instruction_start
8231                               * Cs( V("NoSoftLineBreakEndline")
8232                                   + parsers.any
8233                                   - parsers.nested_breaking_blank
8234                                   - parsers.html_instruction_end)^0
8235                               * parsers.html_instruction_end
8236
8237 parsers.html_blankline = parsers.newline
8238                       * parsers.optionalspace
8239                       * parsers.newline
8240
8241 parsers.html_tag_start = parsers.less
8242
8243 parsers.html_tag_closing_start = parsers.less
8244                               * parsers.slash
8245
8246 parsers.html_tag_end = parsers.html_spnl
8247                     * parsers.more
8248
8249 parsers.html_empty_tag_end = parsers.html_spnl
8250                           * parsers.slash

```

```

8251             * parsers.more
8252
8253 -- opening tags
8254 parsers.html_any_open_inline_tag = parsers.html_tag_start
8255             * parsers.keyword
8256             * parsers.html_attribute^0
8257             * parsers.html_tag_end
8258
8259 parsers.html_any_open_tag = parsers.html_tag_start
8260             * parsers.keyword
8261             * parsers.html_non_newline_attribute^0
8262             * parsers.html_tag_end
8263
8264 parsers.html_open_tag = parsers.html_tag_start
8265             * parsers.block_keyword
8266             * parsers.html_attribute^0
8267             * parsers.html_tag_end
8268
8269 parsers.html_open_special_tag = parsers.html_tag_start
8270             * parsers.special_block_keyword
8271             * parsers.html_attribute^0
8272             * parsers.html_tag_end
8273
8274 -- incomplete tags
8275 parsers.incomplete_tag_following = parsers.spacechar
8276             + parsers.more
8277             + parsers.slash * parsers.more
8278             + #(parsers.newline + parsers.eof)
8279
8280 parsers.incomplete_special_tag_following = parsers.spacechar
8281             + parsers.more
8282             + #( parsers.newline
8283                + parsers.eof)
8284
8285 parsers.html_incomplete_open_tag = parsers.html_tag_start
8286             * parsers.block_keyword
8287             * parsers.incomplete_tag_following
8288
8289 parsers.html_incomplete_open_special_tag
8290 = parsers.html_tag_start
8291 * parsers.special_block_keyword
8292 * parsers.incomplete_special_tag_following
8293
8294 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
8295             * parsers.block_keyword
8296             * parsers.incomplete_tag_following
8297

```

```

8298 parsers.html_incomplete_close_special_tag
8299     = parsers.html_tag_closing_start
8300     * parsers.special_block_keyword
8301     * parsers.incomplete_tag_following
8302
8303 -- closing tags
8304 parsers.html_close_tag = parsers.html_tag_closing_start
8305                       * parsers.block_keyword
8306                       * parsers.html_tag_end
8307
8308 parsers.html_any_close_tag = parsers.html_tag_closing_start
8309                           * parsers.keyword
8310                           * parsers.html_tag_end
8311
8312 parsers.html_close_special_tag = parsers.html_tag_closing_start
8313                               * parsers.special_block_keyword
8314                               * parsers.html_tag_end
8315
8316 -- empty tags
8317 parsers.html_any_empty_inline_tag = parsers.html_tag_start
8318                                  * parsers.keyword
8319                                  * parsers.html_attribute^0
8320                                  * parsers.html_empty_tag_end
8321
8322 parsers.html_any_empty_tag = parsers.html_tag_start
8323                             * parsers.keyword
8324                             * parsers.html_non_newline_attribute^0
8325                             * parsers.optionalspace
8326                             * parsers.slash
8327                             * parsers.more
8328
8329 parsers.html_empty_tag = parsers.html_tag_start
8330                       * parsers.block_keyword
8331                       * parsers.html_attribute^0
8332                       * parsers.html_empty_tag_end
8333
8334 parsers.html_empty_special_tag = parsers.html_tag_start
8335                                * parsers.special_block_keyword
8336                                * parsers.html_attribute^0
8337                                * parsers.html_empty_tag_end
8338
8339 parsers.html_incomplete_blocks
8340     = parsers.html_incomplete_open_tag
8341     + parsers.html_incomplete_open_special_tag
8342     + parsers.html_incomplete_close_tag
8343
8344 -- parse special html blocks

```

```

8345 parsers.html_blankline_ending_special_block_opening
8346 = ( parsers.html_close_special_tag
8347   + parsers.html_empty_special_tag)
8348 * #( parsers.optionalspace
8349   * (parsers.newline + parsers.eof))
8350
8351 parsers.html_blankline_ending_special_block
8352 = parsers.html_blankline_ending_special_block_opening
8353 * parsers.html_blankline_end_condition
8354
8355 parsers.html_special_block_opening
8356 = parsers.html_incomplete_open_special_tag
8357 - parsers.html_empty_special_tag
8358
8359 parsers.html_closing_special_block
8360 = parsers.html_special_block_opening
8361 * parsers.html_until_end(parsers.html_close_special_tag)
8362
8363 parsers.html_special_block
8364 = parsers.html_blankline_ending_special_block
8365 + parsers.html_closing_special_block
8366
8367 -- parse html blocks
8368 parsers.html_block_opening = parsers.html_incomplete_open_tag
8369   + parsers.html_incomplete_close_tag
8370
8371 parsers.html_block = parsers.html_block_opening
8372   * parsers.html_blankline_end_condition
8373
8374 -- parse any html blocks
8375 parsers.html_any_block_opening
8376 = ( parsers.html_any_open_tag
8377   + parsers.html_any_close_tag
8378   + parsers.html_any_empty_tag)
8379 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8380
8381 parsers.html_any_block = parsers.html_any_block_opening
8382   * parsers.html_blankline_end_condition
8383
8384 parsers.html_inline_comment_full = parsers.html_comment_start
8385   * -P(">") * -P("->")
8386   * Cs(( V("NoSoftLineBreakEndline")
8387     + parsers.any - P("--")
8388     - parsers.nested_breaking_blank
8389     - parsers.html_comment_end)^0)
8390   * parsers.html_comment_end
8391

```

```

8392 parsers.html_inline_tags = parsers.html_inline_comment_full
8393 + parsers.html_any_empty_inline_tag
8394 + parsers.html_inline_instruction
8395 + parsers.html_inline_cdatasection
8396 + parsers.html_inline_declaration
8397 + parsers.html_any_open_inline_tag
8398 + parsers.html_any_close_tag
8399

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

8400 parsers.urlchar = parsers.anyescaped
8401 - parsers.newline
8402 - parsers.more
8403
8404 parsers.auto_link_scheme_part = parsers.alphanumeric
8405 + parsers.plus
8406 + parsers.period
8407 + parsers.dash
8408
8409 parsers.auto_link_scheme = parsers.letter
8410 * parsers.auto_link_scheme_part
8411 * parsers.auto_link_scheme_part^-30
8412
8413 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
8414 * ( parsers.any - parsers.spacing
8415 - parsers.less - parsers.more)^0
8416
8417 parsers.printable_characters = S(" !#$%&'*/=?^_`{|}~-")
8418
8419 parsers.email_address_local_part_char = parsers.alphanumeric
8420 + parsers.printable_characters
8421
8422 parsers.email_address_local_part
8423 = parsers.email_address_local_part_char^1
8424
8425 parsers.email_address_dns_label = parsers.alphanumeric
8426 * ( parsers.alphanumeric
8427 + parsers.dash)^-62
8428 * B(parsers.alphanumeric)
8429
8430 parsers.email_address_domain = parsers.email_address_dns_label
8431 * ( parsers.period
8432 * parsers.email_address_dns_label)^0
8433
8434 parsers.email_address = parsers.email_address_local_part
8435 * parsers.at

```

```

8436             * parsers.email_address_domain
8437
8438 parsers.auto_link_url = parsers.less
8439             * C(parsers.absolute_uri)
8440             * parsers.more
8441
8442 parsers.auto_link_email = parsers.less
8443             * C(parsers.email_address)
8444             * parsers.more
8445
8446 parsers.auto_link_relative_reference = parsers.less
8447             * C(parsers.urlchar^1)
8448             * parsers.more
8449
8450 parsers.autolink = parsers.auto_link_url
8451             + parsers.auto_link_email
8452
8453 -- content in balanced brackets, parentheses, or quotes:
8454 parsers.bracketed = P{ parsers.lbracket
8455             * (( parsers.backslash / "\"" * parsers.rbracket
8456             + parsers.any - (parsers.lbracket
8457             + parsers.rbracket
8458             + parsers.blankline^2)
8459             ) + V(1))^0
8460             * parsers.rbracket }
8461
8462 parsers.inparens = P{ parsers.lparent
8463             * ((parsers.anyescaped - (parsers.lparent
8464             + parsers.rparent
8465             + parsers.blankline^2)
8466             ) + V(1))^0
8467             * parsers.rparent }
8468
8469 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8470             * ((parsers.anyescaped - (parsers.squote
8471             + parsers.blankline^2)
8472             ) + V(1))^0
8473             * parsers.squote }
8474
8475 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8476             * ((parsers.anyescaped - (parsers.dquote
8477             + parsers.blankline^2)
8478             ) + V(1))^0
8479             * parsers.dquote }
8480
8481 parsers.link_text = parsers.lbracket
8482             * Cs((parsers.alphanumeric^1

```



```

8483         + parsers.bracketed
8484         + parsers.inticks
8485         + parsers.autolink
8486         + V("InlineHtml")
8487         + ( parsers.backslash * parsers.backslash)
8488         + ( parsers.backslash
8489           * ( parsers.lbracket
8490             + parsers.rbracket)
8491           + V("NoSoftLineBreakSpace")
8492           + V("NoSoftLineBreakEndline")
8493           + (parsers.any
8494             - ( parsers.newline
8495               + parsers.lbracket
8496               + parsers.rbracket
8497               + parsers.blankline^2))))^0)
8498     * parsers.rbracket
8499
8500 parsers.link_label_body = -#(parsers.sp * parsers.rbracket)
8501     * #( ( parsers.any
8502         - parsers.rbracket)^-999
8503         * parsers.rbracket)
8504     * Cs((parsers.alphanumeric^1
8505         + parsers.inticks
8506         + parsers.autolink
8507         + V("InlineHtml")
8508         + ( parsers.backslash * parsers.backslash)
8509         + ( parsers.backslash
8510           * ( parsers.lbracket
8511             + parsers.rbracket)
8512           + V("NoSoftLineBreakSpace")
8513           + V("NoSoftLineBreakEndline")
8514           + (parsers.any
8515             - ( parsers.newline
8516               + parsers.lbracket
8517               + parsers.rbracket
8518               + parsers.blankline^2))))^1)
8519
8520 parsers.link_label = parsers.lbracket
8521     * parsers.link_label_body
8522     * parsers.rbracket
8523
8524 parsers.inparens_url = P{ parsers.lparent
8525     * ((parsers.anyescaped - (parsers.lparent
8526         + parsers.rparent
8527         + parsers.spacing)
8528     ) + V(1))^0
8529     * parsers.rparent }

```

```

8530
8531 -- url for markdown links, allowing nested brackets:
8532 parsers.url      = parsers.less * Cs((parsers.anyescaped
8533                  - parsers.newline
8534                  - parsers.less
8535                  - parsers.more)^0)
8536                  * parsers.more
8537                  + -parsers.less
8538                  * Cs((parsers.inparens_url + (parsers.anyescaped
8539                  - parsers.spacing
8540                  - parsers.lparent
8541                  - parsers.rparent))^1)
8542
8543 -- quoted text:
8544 parsers.title_s   = parsers.squote
8545                  * Cs((parsers.html_entities
8546                  + V("NoSoftLineBreakSpace")
8547                  + V("NoSoftLineBreakEndline")
8548                  + ( parsers.anyescaped
8549                  - parsers.newline
8550                  - parsers.squote
8551                  - parsers.blankline^2))^0)
8552                  * parsers.squote
8553
8554 parsers.title_d   = parsers.dquote
8555                  * Cs((parsers.html_entities
8556                  + V("NoSoftLineBreakSpace")
8557                  + V("NoSoftLineBreakEndline")
8558                  + ( parsers.anyescaped
8559                  - parsers.newline
8560                  - parsers.dquote
8561                  - parsers.blankline^2))^0)
8562                  * parsers.dquote
8563
8564 parsers.title_p   = parsers.lparent
8565                  * Cs((parsers.html_entities
8566                  + V("NoSoftLineBreakSpace")
8567                  + V("NoSoftLineBreakEndline")
8568                  + ( parsers.anyescaped
8569                  - parsers.newline
8570                  - parsers.lparent
8571                  - parsers.rparent
8572                  - parsers.blankline^2))^0)
8573                  * parsers.rparent
8574
8575 parsers.title     = parsers.title_d + parsers.title_s + parsers.title_p
8576

```

```

8577
8578 parsers.optionaltitle
8579   = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8580

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```

8581 -- parse a reference definition: [foo]: /bar "title"
8582 parsers.define_reference_parser = (parsers.check_trail / "")
8583                                 * parsers.link_label * parsers.colon
8584                                 * parsers.spnlc * parsers.url
8585                                 * ( parsers.spnlc_sep * parsers.title
8586                                   * parsers.only_blank
8587                                   + Cc("") * parsers.only_blank)

```

### 3.1.5.8 Inline Elements

```

8588 parsers.Inline           = V("Inline")
8589
8590 -- parse many p between starter and ender
8591 parsers.between = function(p, starter, ender)
8592   local ender2 = B(parsers.nonspacechar) * ender
8593   return ( starter
8594           * #parsers.nonspacechar
8595           * Ct(p * (p - ender2)^0)
8596           * ender2)
8597 end
8598

```

### 3.1.5.9 Block Elements

```

8599 parsers.lineof = function(c)
8600   return ( parsers.check_trail_no_rem
8601           * (P(c) * parsers.optionalspace)^3
8602           * (parsers.newline + parsers.eof))
8603 end
8604
8605 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8606                               + parsers.lineof(parsers.dash)
8607                               + parsers.lineof(parsers.underscore)

```

### 3.1.5.10 Headings

```

8608 -- parse Atx heading start and return level
8609 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8610                       * -parsers.hash / length
8611
8612 -- parse setext header ending and return level
8613 parsers.heading_level

```

```

8614 = parsers.nonindentspace * parsers.equal^1
8615 * parsers.optionalspace * #parsers.newline * Cc(1)
8616 + parsers.nonindentspace * parsers.dash^1
8617 * parsers.optionalspace * #parsers.newline * Cc(2)
8618
8619 local function strip_atx_end(s)
8620   return s:gsub("%s+#+%s*\n$", "")
8621 end
8622
8623 parsers.atx_heading = parsers.check_trail_no_rem
8624                       * Cg(parsers.heading_start, "level")
8625                       * (C( parsers.optionalspace
8626                           * parsers.hash^0
8627                           * parsers.optionalspace
8628                           * parsers.newline)
8629                       + parsers.spacechar^1
8630                       * 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`.

```

8631 M.reader = {}
8632 function M.reader.new(writer, options)
8633   local self = {}

```

Make the `writer` and `options` parameters available as `reader->writer` and `reader->options`, respectively, so that they are accessible from extensions.

```

8634   self.writer = writer
8635   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.

```

8636   self.parsers = {}
8637   (function(parsers)
8638     setmetatable(self.parsers, {
8639       __index = function (_, key)
8640         return parsers[key]

```

```

8641     end
8642   })
8643 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.

```

8644   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.

```

8645   function self.normalize_tag(tag)
8646     tag = util.rope_to_string(tag)
8647     tag = tag:gsub("[ \n\r\t]+", " ")
8648     tag = tag:gsub("^ ", ""):gsub(" $", "")
8649     tag = uni_algos.case.casefold(tag, true, false)
8650     return tag
8651   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.

```

8652   local function iterlines(s, f)
8653     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8654     return util.rope_to_string(rope)
8655   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.

```

8656   if options.preserveTabs then
8657     self.expandtabs = function(s) return s end
8658   else
8659     self.expandtabs = function(s)
8660       if s:find("\t") then
8661         return iterlines(s, util.expand_tabs_in_line)
8662       else
8663         return s
8664       end
8665     end
8666   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.

```
8667 self.parser_functions = {}
8668 self.create_parser = function(name, grammar, toplevel)
8669     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.

```
8670     if toplevel and options.stripIndent then
8671         local min_prefix_length, min_prefix = nil, ''
8672         str = iterlines(str, function(line)
8673             if lpeg.match(parsers.nonemptyline, line) == nil then
8674                 return line
8675             end
8676             line = util.expand_tabs_in_line(line)
8677             local prefix = lpeg.match(C(parsers.optionalspace), line)
8678             local prefix_length = #prefix
8679             local is_shorter = min_prefix_length == nil
8680             if not is_shorter then
8681                 is_shorter = prefix_length < min_prefix_length
8682             end
8683             if is_shorter then
8684                 min_prefix_length, min_prefix = prefix_length, prefix
8685             end
8686             return line
8687         end)
8688         str = str:gsub('^' .. min_prefix, '')
8689     end
```

If the parser is top-level and the `texComments` or `hybrid` Lua options are enabled, we will strip all plain  $\TeX$  comments from the input string `str` together with the trailing newline characters.

```
8690     if toplevel and (options.texComments or options.hybrid) then
8691         str = lpeg.match(Ct(parsers.commented_line^1), str)
8692         str = util.rope_to_string(str)
8693     end
8694     local res = lpeg.match(grammar(), str)
8695     if res == nil then
8696         return writer.error(
8697             format("Parser `%s` failed to process the input text.", name),
8698             format("Here are the first 20 characters of the remaining "
8699                 .. "unprocessed text: `%s`.", str:sub(1,20))
8700         )
8701     else
8702         return res
```

```

8703     end
8704     end
8705 end
8706
8707 self.create_parser("parse_blocks",
8708                   function()
8709                     return parsers.blocks
8710                   end, true)
8711
8712 self.create_parser("parse_blocks_nested",
8713                   function()
8714                     return parsers.blocks_nested
8715                   end, false)
8716
8717 self.create_parser("parse_inlines",
8718                   function()
8719                     return parsers.inlines
8720                   end, false)
8721
8722 self.create_parser("parse_inlines_no_inline_note",
8723                   function()
8724                     return parsers.inlines_no_inline_note
8725                   end, false)
8726
8727 self.create_parser("parse_inlines_no_html",
8728                   function()
8729                     return parsers.inlines_no_html
8730                   end, false)
8731
8732 self.create_parser("parse_inlines_nbsp",
8733                   function()
8734                     return parsers.inlines_nbsp
8735                   end, false)
8736 self.create_parser("parse_inlines_no_link_or_emphasis",
8737                   function()
8738                     return parsers.inlines_no_link_or_emphasis
8739                   end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8740 parsers.minimally_indented_blankline
8741     = parsers.check_minimal_indent * (parsers.blankline / "")
8742
8743 parsers.minimally_indented_block
8744     = parsers.check_minimal_indent * V("Block")
8745

```

```

8746 parsers.minimally_indented_block_or_paragraph
8747     = parsers.check_minimal_indent * V("BlockOrParagraph")
8748
8749 parsers.minimally_indented_paragraph
8750     = parsers.check_minimal_indent * V("Paragraph")
8751
8752 parsers.minimally_indented_plain
8753     = parsers.check_minimal_indent * V("Plain")
8754
8755 parsers.minimally_indented_par_or_plain
8756     = parsers.minimally_indented_paragraph
8757     + parsers.minimally_indented_plain
8758
8759 parsers.minimally_indented_par_or_plain_no_blank
8760     = parsers.minimally_indented_par_or_plain
8761     - parsers.minimally_indented_blankline
8762
8763 parsers.minimally_indented_ref
8764     = parsers.check_minimal_indent * V("Reference")
8765
8766 parsers.minimally_indented_blank
8767     = parsers.check_minimal_indent * V("Blank")
8768
8769 parsers.conditionally_indented_blankline
8770     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8771
8772 parsers.minimally_indented_ref_or_block
8773     = parsers.minimally_indented_ref
8774     + parsers.minimally_indented_block
8775     - parsers.minimally_indented_blankline
8776
8777 parsers.minimally_indented_ref_or_block_or_par
8778     = parsers.minimally_indented_ref
8779     + parsers.minimally_indented_block_or_paragraph
8780     - parsers.minimally_indented_blankline
8781

```

The following pattern parses the properly indented content that follows the initial container start.

```

8782
8783 function parsers.separator_loop(separated_block, paragraph,
8784                               block_separator, paragraph_separator)
8785     return separated_block
8786         + block_separator
8787         * paragraph
8788         * separated_block
8789         + paragraph_separator

```



```

8790         * paragraph
8791     end
8792
8793     function parsers.create_loop_body_pair(separated_block, paragraph,
8794   block_separator,
8795   paragraph_separator)
8796         return {
8797             block = parsers.separator_loop(separated_block, paragraph,
8798   block_separator, block_separator),
8799             par = parsers.separator_loop(separated_block, paragraph,
8800   block_separator, paragraph_separator)
8801         }
8802     end
8803
8804     parsers.block_sep_group = function(blank)
8805         return blank^0 * parsers.eof
8806             + ( blank^2 / writer.paragraphsep
8807               + blank^0 / writer.interblocksep
8808             )
8809     end
8810
8811     parsers.par_sep_group = function(blank)
8812         return blank^0 * parsers.eof
8813             + blank^0 / writer.paragraphsep
8814     end
8815
8816     parsers.sep_group_no_output = function(blank)
8817         return blank^0 * parsers.eof
8818             + blank^0
8819     end
8820
8821     parsers.content_blank = parsers.minimally_indented_blankline
8822
8823     parsers.ref_or_block_separated
8824     = parsers.sep_group_no_output(parsers.content_blank)
8825     * ( parsers.minimally_indented_ref
8826       - parsers.content_blank)
8827     + parsers.block_sep_group(parsers.content_blank)
8828     * ( parsers.minimally_indented_block
8829       - parsers.content_blank)
8830
8831     parsers.loop_body_pair =
8832     parsers.create_loop_body_pair(
8833         parsers.ref_or_block_separated,
8834         parsers.minimally_indented_par_or_plain_no_blank,
8835         parsers.block_sep_group(parsers.content_blank),
8836         parsers.par_sep_group(parsers.content_blank))

```

```

8837
8838 parsers.content_loop = ( V("Block")
8839     * parsers.loop_body_pair.block^0
8840     + (V("Paragraph") + V("Plain"))
8841     * parsers.ref_or_block_separated
8842     * parsers.loop_body_pair.block^0
8843     + (V("Paragraph") + V("Plain"))
8844     * parsers.loop_body_pair.par^0
8845     * parsers.content_blank^0
8846
8847 parsers.indented_content = function()
8848     return Ct( (V("Reference") + (parsers.blankline / ""))
8849         * parsers.content_blank^0
8850         * parsers.check_minimal_indent
8851         * parsers.content_loop
8852         + (V("Reference") + (parsers.blankline / ""))
8853         * parsers.content_blank^0
8854         + parsers.content_loop)
8855 end
8856
8857 parsers.add_indent = function(pattern, name, breakable)
8858     return Cg(Cmt( Cb("indent_info")
8859         * Ct(pattern)
8860         * ( #parsers.linechar -- check if starter is blank
8861         * Cc(false) + Cc(true))
8862         * Cc(name)
8863         * Cc(breakable),
8864         process_starter_indent), "indent_info")
8865 end
8866

```

### 3.1.6.4 Parsers Used for Markdown Lists (local)

```

8867 if options.hashEnumerators then
8868     parsers.dig = parsers.digit + parsers.hash
8869 else
8870     parsers.dig = parsers.digit
8871 end
8872
8873 parsers.enumerator = function(delimiter_type, interrupting)
8874     local delimiter_range
8875     local allowed_end
8876     if interrupting then
8877         delimiter_range = P("1")
8878         allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8879     else
8880         delimiter_range = parsers.dig * parsers.dig^-8

```

```

8881     allowed_end = C(parsers.spacechar^1)
8882                 + #(parsers.newline + parsers.eof)
8883   end
8884
8885   return parsers.check_trail
8886         * Ct(C(delimiter_range) * C(delimiter_type))
8887         * allowed_end
8888   end
8889
8890   parsers.starter = parsers.bullet(parsers.dash)
8891                   + parsers.bullet(parsers.asterisk)
8892                   + parsers.bullet(parsers.plus)
8893                   + parsers.enumerator(parsers.period)
8894                   + parsers.enumerator(parsers.rparent)
8895

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

8896   parsers.blockquote_start
8897     = parsers.check_trail
8898     * C(parsers.more)
8899     * C(parsers.spacechar^0)
8900
8901   parsers.blockquote_body
8902     = parsers.add_indent(parsers.blockquote_start, "bq", true)
8903     * parsers.indented_content()
8904     * remove_indent("bq")
8905
8906   if not options.breakableBlockquotes then
8907     parsers.blockquote_body
8908       = parsers.add_indent(parsers.blockquote_start, "bq", false)
8909       * parsers.indented_content()
8910       * remove_indent("bq")
8911   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.

```

8912   local function parse_content_part(content_part)
8913     local rope = util.rope_to_string(content_part)
8914     local parsed
8915       = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8916     parsed.indent_info = nil
8917     return parsed
8918   end
8919

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```
8920 local collect_emphasis_content =
8921   function(t, opening_index, closing_index)
8922     local content = {}
8923
8924     local content_part = {}
8925     for i = opening_index, closing_index do
8926       local value = t[i]
8927
8928       if value.rendered ~= nil then
8929         content[#content + 1] = parse_content_part(content_part)
8930         content_part = {}
8931         content[#content + 1] = value.rendered
8932         value.rendered = nil
8933       else
8934         if value.type == "delimiter"
8935           and value.element == "emphasis" then
8936           if value.is_active then
8937             content_part[#content_part + 1]
8938               = string.rep(value.character, value.current_count)
8939           end
8940         else
8941             content_part[#content_part + 1] = value.content
8942         end
8943         value.content = ''
8944         value.is_active = false
8945       end
8946     end
8947
8948     if next(content_part) ~= nil then
8949       content[#content + 1] = parse_content_part(content_part)
8950     end
8951
8952     return content
8953   end
8954
```

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as emphasis.

```
8955 local function fill_emph(t, opening_index, closing_index)
8956   local content
8957     = collect_emphasis_content(t, opening_index + 1,
8958                               closing_index - 1)
8959   t[opening_index + 1].is_active = true
8960   t[opening_index + 1].rendered = writer.emphasis(content)
8961 end
```

8962

Render content between the `opening_index` and `closing_index` in the delimiter table `t` as strong emphasis.

```
8963 local function fill_strong(t, opening_index, closing_index)
8964     local content
8965         = collect_emphasis_content(t, opening_index + 1,
8966                                   closing_index - 1)
8967     t[opening_index + 1].is_active = true
8968     t[opening_index + 1].rendered = writer.strong(content)
8969 end
8970
```

Check whether the opening delimiter `opening_delimiter` and closing delimiter `closing_delimiter` break rule three together.

```
8971 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8972     return ( opening_delimiter.is_closing
8973             or closing_delimiter.is_opening)
8974     and (( opening_delimiter.original_count
8975            + closing_delimiter.original_count) % 3 == 0)
8976     and ( opening_delimiter.original_count % 3 ~= 0
8977           or closing_delimiter.original_count % 3 ~= 0)
8978 end
8979
```

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`.

```
8980 local find_emphasis_opener = function(t, bottom_index, latest_index,
8981                                     character, closing_delimiter)
8982     for i = latest_index, bottom_index, -1 do
8983         local value = t[i]
8984         if value.is_active and
8985            value.is_opening and
8986            value.type == "delimiter" and
8987            value.element == "emphasis" and
8988            (value.character == character) and
8989            (value.current_count > 0) then
8990             if not breaks_three_rule(value, closing_delimiter) then
8991                 return i
8992             end
8993         end
8994     end
8995 end
8996
```

Iterate over the delimiters in the delimiter table `t`, producing emphasis or strong emphasis macros.

```

8997 local function process_emphasis(t, opening_index, closing_index)
8998     for i = opening_index, closing_index do
8999         local value = t[i]
9000         if value.type == "delimiter" and value.element == "emphasis" then
9001             local delimiter_length = string.len(value.content)
9002             value.character = string.sub(value.content, 1, 1)
9003             value.current_count = delimiter_length
9004             value.original_count = delimiter_length
9005         end
9006     end
9007
9008     local openers_bottom = {
9009         ['*'] = {
9010             [true] = {opening_index, opening_index, opening_index},
9011             [false] = {opening_index, opening_index, opening_index}
9012         },
9013         ['_'] = {
9014             [true] = {opening_index, opening_index, opening_index},
9015             [false] = {opening_index, opening_index, opening_index}
9016         }
9017     }
9018
9019     local current_position = opening_index
9020     local max_position = closing_index
9021
9022     while current_position <= max_position do
9023         local value = t[current_position]
9024
9025         if value.type ~= "delimiter" or
9026            value.element ~= "emphasis" or
9027            not value.is_active or
9028            not value.is_closing or
9029            (value.current_count <= 0) then
9030             current_position = current_position + 1
9031             goto continue
9032         end
9033
9034         local character = value.character
9035         local is_opening = value.is_opening
9036         local closing_length_modulo_three = value.original_count % 3
9037
9038         local current_openers_bottom
9039             = openers_bottom[character][is_opening]
9040               [closing_length_modulo_three + 1]
9041
9042         local opener_position
9043             = find_emphasis_opener(t, current_openers_bottom,

```

```

9044             current_position - 1, character, value)
9045
9046     if (opener_position == nil) then
9047         openers_bottom[character][is_opening]
9048             [closing_length_modulo_three + 1]
9049         = current_position
9050         current_position = current_position + 1
9051         goto continue
9052     end
9053
9054     local opening_delimiter = t[opener_position]
9055
9056     local current_opening_count = opening_delimiter.current_count
9057     local current_closing_count = t[current_position].current_count
9058
9059     if (current_opening_count >= 2)
9060         and (current_closing_count >= 2) then
9061         opening_delimiter.current_count = current_opening_count - 2
9062         t[current_position].current_count = current_closing_count - 2
9063         fill_strong(t, opener_position, current_position)
9064     else
9065         opening_delimiter.current_count = current_opening_count - 1
9066         t[current_position].current_count = current_closing_count - 1
9067         fill_emph(t, opener_position, current_position)
9068     end
9069
9070     ::continue::
9071 end
9072 end
9073
9074 local cont = lpeg.R("\128\191") -- continuation byte
9075

```

Match a UTF-8 character of byte length *n*.

```

9076 local function utf8_by_byte_count(n)
9077     if (n == 1) then
9078         return lpeg.R("\0\127")
9079     end
9080     if (n == 2) then
9081         return lpeg.R("\194\223") * cont
9082     end
9083     if (n == 3) then
9084         return lpeg.R("\224\239") * cont * cont
9085     end
9086     if (n == 4) then
9087         return lpeg.R("\240\244") * cont * cont * cont
9088     end
9089 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`.

```
9090 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
9091     local c
9092     local char_length
9093     for pos = start_pos, end_pos, 1 do
9094         if (start_pos < 0) then
9095             char_length = -pos
9096         else
9097             char_length = pos + 1
9098         end
9099
9100         if (chartype == "punctuation") then
9101             if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
9102                 return i
9103             end
9104         else
9105             c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
9106             if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
9107                 return i
9108             end
9109         end
9110     end
9111 end
9112
9113 local function check_preceding_unicode_punctuation(s, i)
9114     return check_unicode_type(s, i, -4, -1, "punctuation")
9115 end
9116
9117 local function check_preceding_unicode_whitespace(s, i)
9118     return check_unicode_type(s, i, -4, -1, "%s")
9119 end
9120
9121 local function check_following_unicode_punctuation(s, i)
9122     return check_unicode_type(s, i, 0, 3, "punctuation")
9123 end
9124
9125 local function check_following_unicode_whitespace(s, i)
9126     return check_unicode_type(s, i, 0, 3, "%s")
9127 end
9128
9129 parsers.unicode_preceding_punctuation
9130     = B(parsers.escapable)
9131     + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
9132
9133 parsers.unicode_preceding_whitespace
9134     = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
```



```

9135
9136 parsers.unicode_following_punctuation
9137     = #parsers.escapable
9138     + Cmt(parsers.succeed, check_following_unicode_punctuation)
9139
9140 parsers.unicode_following_whitespace
9141     = Cmt(parsers.succeed, check_following_unicode_whitespace)
9142
9143 parsers.delimiter_run = function(character)
9144     return (B(parsers.backslash * character) + -B(character))
9145             * character^1
9146             * -#character
9147 end
9148
9149 parsers.left_flanking_delimiter_run = function(character)
9150     return (B( parsers.any)
9151             * ( parsers.unicode_preceding_punctuation
9152                 + parsers.unicode_preceding_whitespace)
9153             + -B(parsers.any))
9154             * parsers.delimiter_run(character)
9155             * parsers.unicode_following_punctuation
9156             + parsers.delimiter_run(character)
9157             * -( parsers.unicode_following_punctuation
9158                 + parsers.unicode_following_whitespace
9159                 + parsers.eof)
9160 end
9161
9162 parsers.right_flanking_delimiter_run = function(character)
9163     return parsers.unicode_preceding_punctuation
9164             * parsers.delimiter_run(character)
9165             * ( parsers.unicode_following_punctuation
9166                 + parsers.unicode_following_whitespace
9167                 + parsers.eof)
9168             + (B(parsers.any)
9169                 * -( parsers.unicode_preceding_punctuation
9170                     + parsers.unicode_preceding_whitespace))
9171             * parsers.delimiter_run(character)
9172 end
9173
9174 if options.underscores then
9175     parsers.emph_start
9176         = parsers.left_flanking_delimiter_run(parsers.asterisk)
9177         + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
9178             + ( parsers.unicode_preceding_punctuation
9179                 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
9180         * parsers.left_flanking_delimiter_run(parsers.underscore)
9181

```

```

9182     parsers.emph_end
9183         = parsers.right_flanking_delimiter_run(parsers.asterisk)
9184         + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
9185             + #( parsers.left_flanking_delimiter_run(parsers.underscore)
9186                 * parsers.unicode_following_punctuation))
9187             * parsers.right_flanking_delimiter_run(parsers.underscore)
9188     else
9189         parsers.emph_start
9190             = parsers.left_flanking_delimiter_run(parsers.asterisk)
9191
9192         parsers.emph_end
9193             = parsers.right_flanking_delimiter_run(parsers.asterisk)
9194     end
9195
9196     parsers.emph_capturing_open_and_close
9197         = #parsers.emph_start * #parsers.emph_end
9198         * Ct( Cg(Cc("delimiter"), "type")
9199             * Cg(Cc("emphasis"), "element")
9200             * Cg(C(parsers.emph_start), "content")
9201             * Cg(Cc(true), "is_opening")
9202             * Cg(Cc(true), "is_closing"))
9203
9204     parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
9205                                     * Cg(Cc("emphasis"), "element")
9206                                     * Cg(C(parsers.emph_start), "content")
9207                                     * Cg(Cc(true), "is_opening")
9208                                     * Cg(Cc(false), "is_closing"))
9209
9210     parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
9211                                       * Cg(Cc("emphasis"), "element")
9212                                       * Cg(C(parsers.emph_end), "content")
9213                                       * Cg(Cc(false), "is_opening")
9214                                       * Cg(Cc(true), "is_closing"))
9215
9216     parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
9217                                 + parsers.emph_capturing_open
9218                                 + parsers.emph_capturing_close
9219
9220     parsers.emph_open = parsers.emph_capturing_open_and_close
9221                       + parsers.emph_capturing_open
9222
9223     parsers.emph_close = parsers.emph_capturing_open_and_close
9224                       + parsers.emph_capturing_close
9225

```

### 3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

9226 -- List of references defined in the document
9227 local references
9228
9229 -- List of note references defined in the document
9230 parsers.rawnotes = {}
9231

```

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.

```

9232 function self.register_link(_, tag, url, title,
9233                             attributes)
9234     local normalized_tag = self.normalize_tag(tag)
9235     if references[normalized_tag] == nil then
9236         references[normalized_tag] = {
9237             url = url,
9238             title = title,
9239             attributes = attributes
9240         }
9241     end
9242     return ""
9243 end
9244

```

The `reader->lookup_reference` method looks up a reference with link label `tag`.

```

9245 function self.lookup_reference(tag)
9246     return references[self.normalize_tag(tag)]
9247 end
9248

```

The `reader->lookup_note_reference` method looks up a note reference with label `tag`.

```

9249 function self.lookup_note_reference(tag)
9250     return parsers.rawnotes[self.normalize_tag(tag)]
9251 end
9252
9253 parsers.title_s_direct_ref = parsers.squote
9254                             * Cs((parsers.html_entities
9255                                 + ( parsers.anyescaped
9256                                   - parsers.squote
9257                                   - parsers.blankline^2))^0)
9258                             * parsers.squote
9259
9260 parsers.title_d_direct_ref = parsers.dquote
9261                             * Cs((parsers.html_entities
9262                                 + ( parsers.anyescaped
9263                                   - parsers.dquote
9264                                   - parsers.blankline^2))^0)

```

```

9265             * parsers.dquote
9266
9267 parsers.title_p_direct_ref = parsers.lparent
9268             * Cs((parsers.html_entities
9269                 + ( parsers.anyescaped
9270                   - parsers.lparent
9271                   - parsers.rparent
9272                   - parsers.blankline^2))^0)
9273             * parsers.rparent
9274
9275 parsers.title_direct_ref = parsers.title_s_direct_ref
9276             + parsers.title_d_direct_ref
9277             + parsers.title_p_direct_ref
9278
9279 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
9280             * Cg(parsers.url + Cc(""), "url")
9281             * parsers.spnl
9282             * Cg( parsers.title_direct_ref
9283                 + Cc(""), "title")
9284             * parsers.spnl * parsers.rparent
9285
9286 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
9287             * Cg(parsers.url + Cc(""), "url")
9288             * parsers.spnlc
9289             * Cg(parsers.title + Cc(""), "title")
9290             * parsers.spnlc * parsers.rparent
9291
9292 parsers.empty_link = parsers.lbracket
9293             * parsers.rbracket
9294
9295 parsers.inline_link = parsers.link_text
9296             * parsers.inline_direct_ref
9297
9298 parsers.full_link = parsers.link_text
9299             * parsers.link_label
9300
9301 parsers.shortcut_link = parsers.link_label
9302             * -(parsers.empty_link + parsers.link_label)
9303
9304 parsers.collapsed_link = parsers.link_label
9305             * parsers.empty_link
9306
9307 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
9308             * Cg(Cc("inline"), "link_type")
9309             + #(parsers.exclamation * parsers.full_link)
9310             * Cg(Cc("full"), "link_type")
9311             + #( parsers.exclamation

```

```

9312         * parsers.collapsed_link)
9313     * Cg(Cc("collapsed"), "link_type")
9314     + #(parsers.exclamation * parsers.shortcut_link)
9315     * Cg(Cc("shortcut"), "link_type")
9316     + #(parsers.exclamation * parsers.empty_link)
9317     * Cg(Cc("empty"), "link_type")
9318
9319     parsers.link_opening = #parsers.inline_link
9320     * Cg(Cc("inline"), "link_type")
9321     + #parsers.full_link
9322     * Cg(Cc("full"), "link_type")
9323     + #parsers.collapsed_link
9324     * Cg(Cc("collapsed"), "link_type")
9325     + #parsers.shortcut_link
9326     * Cg(Cc("shortcut"), "link_type")
9327     + #parsers.empty_link
9328     * Cg(Cc("empty_link"), "link_type")
9329     + #parsers.link_text
9330     * Cg(Cc("link_text"), "link_type")
9331
9332     parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9333     * Cg(Cc("note_inline"), "link_type")
9334
9335     parsers.raw_note_opening = #( parsers.lbracket
9336         * parsers.circumflex
9337         * parsers.link_label_body
9338         * parsers.rbracket)
9339     * Cg(Cc("raw_note"), "link_type")
9340
9341     local inline_note_element = Cg(Cc("note"), "element")
9342     * parsers.note_opening
9343     * Cg( parsers.circumflex
9344         * parsers.lbracket, "content")
9345
9346     local image_element = Cg(Cc("image"), "element")
9347     * parsers.image_opening
9348     * Cg( parsers.exclamation
9349         * parsers.lbracket, "content")
9350
9351     local note_element = Cg(Cc("note"), "element")
9352     * parsers.raw_note_opening
9353     * Cg( parsers.lbracket
9354         * parsers.circumflex, "content")
9355
9356     local link_element = Cg(Cc("link"), "element")
9357     * parsers.link_opening
9358     * Cg(parsers.lbracket, "content")

```

```

9359
9360 local opening_elements = parsers.fail
9361
9362 if options.inlineNotes then
9363   opening_elements = opening_elements + inline_note_element
9364 end
9365
9366 opening_elements = opening_elements + image_element
9367
9368 if options.notes then
9369   opening_elements = opening_elements + note_element
9370 end
9371
9372 opening_elements = opening_elements + link_element
9373
9374 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
9375                                * Cg(Cc(true), "is_opening")
9376                                * Cg(Cc(false), "is_closing")
9377                                * opening_elements)
9378
9379 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
9380                                * Cg(Cc("link"), "element")
9381                                * Cg(Cc(false), "is_opening")
9382                                * Cg(Cc(true), "is_closing")
9383                                * ( Cg(Cc(true), "is_direct")
9384                                * Cg( parsers.rbracket
9385                                    * #parsers.inline_direct_ref,
9386                                      "content")
9387                                + Cg(Cc(false), "is_direct")
9388                                * Cg(parsers.rbracket, "content")))
9389
9390 parsers.link_image_open_or_close = parsers.link_image_opening
9391                                + parsers.link_image_closing
9392
9393 if options.html then
9394   parsers.link_emph_precedence = parsers.inticks
9395                                + parsers.autolink
9396                                + parsers.html_inline_tags
9397 else
9398   parsers.link_emph_precedence = parsers.inticks
9399                                + parsers.autolink
9400 end
9401
9402 parsers.link_and_emph_endline = parsers.newline
9403                                * ((parsers.check_minimal_indent
9404                                * -V("EndlineExceptions")
9405                                + parsers.check_optional_indent

```

```

9406             * -V("EndlineExceptions")
9407             * -V("ListStarter")) / "")
9408             * parsers.spacechar^0 / "\n"
9409
9410 parsers.link_and_emph_content
9411   = Ct( Cg(Cc("content"), "type")
9412     * Cg(Cs(( parsers.link_emph_precedence
9413       + parsers.backslash * parsers.linechar
9414       + parsers.link_and_emph_endline
9415       + (parsers.linechar
9416         - parsers.blankline^2
9417         - parsers.link_image_open_or_close
9418         - parsers.emph_open_or_close))^0), "content"))
9419
9420 parsers.link_and_emph_table
9421   = (parsers.link_image_opening + parsers.emph_open)
9422     * parsers.link_and_emph_content
9423     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9424       * parsers.link_and_emph_content)^1
9425

```

Collect the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9426 local function collect_link_content(t, opening_index, closing_index)
9427   local content = {}
9428   for i = opening_index, closing_index do
9429     content[#content + 1] = t[i].content
9430   end
9431   return util.rope_to_string(content)
9432 end
9433

```

Look for the closest potential link opener in the delimiter table `t` in the range from `bottom_index` to `latest_index`.

```

9434 local function find_link_opener(t, bottom_index, latest_index)
9435   for i = latest_index, bottom_index, -1 do
9436     local value = t[i]
9437     if value.type == "delimiter" and
9438       value.is_opening and
9439       ( value.element == "link"
9440       or value.element == "image"
9441       or value.element == "note")
9442       and not value.removed then
9443       if value.is_active then
9444         return i
9445       end
9446       value.removed = true
9447     end
9448   end
9449   return nil

```

```

9448     end
9449     end
9450     end
9451

```

Find the position of a delimiter that closes a full link after an an index `latest_index` in the delimiter table `t`.

```

9452     local function find_next_link_closing_index(t, latest_index)
9453         for i = latest_index, #t do
9454             local value = t[i]
9455             if value.is_closing and
9456                 value.element == "link" and
9457                 not value.removed then
9458                 return i
9459             end
9460         end
9461     end
9462

```

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.

```

9463     local function disable_previous_link_openers(t, opening_index)
9464         if t[opening_index].element == "image" then
9465             return
9466         end
9467
9468         for i = opening_index, 1, -1 do
9469             local value = t[i]
9470             if value.is_active and
9471                 value.type == "delimiter" and
9472                 value.is_opening and
9473                 value.element == "link" then
9474                 value.is_active = false
9475             end
9476         end
9477     end
9478

```

Disable the delimiters between the `opening_index` and `closing_index` in the delimiter table `t` by marking them inactive with the `is_active` property.

```

9479     local function disable_range(t, opening_index, closing_index)
9480         for i = opening_index, closing_index do
9481             local value = t[i]
9482             if value.is_active then
9483                 value.is_active = false
9484                 if value.type == "delimiter" then
9485                     value.removed = true
9486                 end

```



```

9487     end
9488     end
9489 end
9490

```

Clear the parsed content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9491 local delete_parsed_content_in_range =
9492   function(t, opening_index, closing_index)
9493     for i = opening_index, closing_index do
9494       t[i].rendered = nil
9495     end
9496   end
9497

```

Clear the content between the `opening_index` and `closing_index` in the delimiter table `t`.

```

9498 local function empty_content_in_range(t, opening_index, closing_index)
9499   for i = opening_index, closing_index do
9500     t[i].content = ''
9501   end
9502 end
9503

```

Join the attributes from the link reference definition `reference_attributes` with the link's own attributes `own_attributes`.

```

9504 local function join_attributes(reference_attributes, own_attributes)
9505   local merged_attributes = {}
9506   for _, attribute in ipairs(reference_attributes or {}) do
9507     table.insert(merged_attributes, attribute)
9508   end
9509   for _, attribute in ipairs(own_attributes or {}) do
9510     table.insert(merged_attributes, attribute)
9511   end
9512   if next(merged_attributes) == nil then
9513     merged_attributes = nil
9514   end
9515   return merged_attributes
9516 end
9517

```

Parse content between two delimiters in the delimiter table `t`. Produce the respective link and image macros.

```

9518 local render_link_or_image =
9519   function(t, opening_index, closing_index, content_end_index,
9520           reference)
9521     process_emphasis(t, opening_index, content_end_index)
9522     local mapped = collect_emphasis_content(t, opening_index + 1,

```

```

9523                                     content_end_index - 1)
9524
9525     local rendered = {}
9526     if (t[opening_index].element == "link") then
9527         rendered = writer.link(mapped, reference.url,
9528                               reference.title, reference.attributes)
9529     end
9530
9531     if (t[opening_index].element == "image") then
9532         rendered = writer.image(mapped, reference.url, reference.title,
9533                                reference.attributes)
9534     end
9535
9536     if (t[opening_index].element == "note") then
9537         if (t[opening_index].link_type == "note_inline") then
9538             rendered = writer.note(mapped)
9539         end
9540         if (t[opening_index].link_type == "raw_note") then
9541             rendered = writer.note(reference)
9542         end
9543     end
9544
9545     t[opening_index].rendered = rendered
9546     delete_parsed_content_in_range(t, opening_index + 1,
9547                                   closing_index)
9548     empty_content_in_range(t, opening_index, closing_index)
9549     disable_previous_link_openers(t, opening_index)
9550     disable_range(t, opening_index, closing_index)
9551 end
9552

```

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.

```

9553     local resolve_inline_following_content =
9554         function(t, closing_index, match_reference, match_link_attributes)
9555             local content = ""
9556             for i = closing_index + 1, #t do
9557                 content = content .. t[i].content
9558             end
9559
9560             local matching_content = parsers.succeed
9561
9562             if match_reference then
9563                 matching_content = matching_content
9564                     * parsers.inline_direct_ref_inside
9565             end

```

```

9566
9567     if match_link_attributes then
9568         matching_content = matching_content
9569             * Cg(Ct(parsers.attributes^-1), "attributes")
9570     end
9571
9572     local matched = lpeg.match(Ct( matching_content
9573         * Cg(Cp(), "end_position")), content)
9574
9575     local matched_count = matched.end_position - 1
9576     for i = closing_index + 1, #t do
9577         local value = t[i]
9578
9579         local chars_left = matched_count
9580         matched_count = matched_count - #value.content
9581
9582         if matched_count <= 0 then
9583             value.content = value.content:sub(chars_left + 1)
9584             break
9585         end
9586
9587         value.content = ''
9588         value.is_active = false
9589     end
9590
9591     local attributes = matched.attributes
9592     if attributes == nil or next(attributes) == nil then
9593         attributes = nil
9594     end
9595
9596     return {
9597         url = matched.url or "",
9598         title = matched.title or "",
9599         attributes = attributes
9600     }
9601 end
9602

```

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.

```

9603 local function resolve_inline_link(t, opening_index, closing_index)
9604     local inline_content
9605         = resolve_inline_following_content(t, closing_index, true,
9606             t.match_link_attributes)
9607     render_link_or_image(t, opening_index, closing_index,
9608         closing_index, inline_content)

```

```
9609 end
9610
```

Resolve an inline note `^[a]` from the delimiters at `opening_index` and `closing_index` within a delimiter table `t`.

```
9611 local function resolve_note_inline_link =
9612     function(t, opening_index, closing_index)
9613         local inline_content
9614             = resolve_inline_following_content(t, closing_index,
9615   false, false)
9616         render_link_or_image(t, opening_index, closing_index,
9617                             closing_index, inline_content)
9618     end
9619
```

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.

```
9620 local function resolve_shortcut_link(t, opening_index, closing_index)
9621     local content
9622         = collect_link_content(t, opening_index + 1, closing_index - 1)
9623     local r = self.lookup_reference(content)
9624
9625     if r then
9626         local inline_content
9627             = resolve_inline_following_content(t, closing_index, false,
9628   t.match_link_attributes)
9629         r.attributes
9630             = join_attributes(r.attributes, inline_content.attributes)
9631         render_link_or_image(t, opening_index, closing_index,
9632                             closing_index, r)
9633     end
9634 end
9635
```

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.

```
9636 local function resolve_raw_note_link(t, opening_index, closing_index)
9637     local content
9638         = collect_link_content(t, opening_index + 1, closing_index - 1)
9639     local r = self.lookup_note_reference(content)
9640
9641     if r then
9642         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9643         render_link_or_image(t, opening_index, closing_index,
9644                             closing_index, parsed_ref)
9645     end
9646 end
```

9647

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.

```
9648 local function resolve_full_link(t, opening_index, closing_index)
9649   local next_link_closing_index
9650     = find_next_link_closing_index(t, closing_index + 4)
9651   local next_link_content
9652     = collect_link_content(t, closing_index + 3,
9653                           next_link_closing_index - 1)
9654   local r = self.lookup_reference(next_link_content)
9655
9656   if r then
9657     local inline_content
9658       = resolve_inline_following_content(t, next_link_closing_index,
9659   false,
9660   t.match_link_attributes)
9661     r.attributes
9662       = join_attributes(r.attributes, inline_content.attributes)
9663     render_link_or_image(t, opening_index, next_link_closing_index,
9664                         closing_index, r)
9665   end
9666 end
9667
```

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.

```
9668 local function resolve_collapsed_link(t, opening_index, closing_index)
9669   local next_link_closing_index
9670     = find_next_link_closing_index(t, closing_index + 4)
9671   local content
9672     = collect_link_content(t, opening_index + 1, closing_index - 1)
9673   local r = self.lookup_reference(content)
9674
9675   if r then
9676     local inline_content
9677       = resolve_inline_following_content(t, closing_index, false,
9678   t.match_link_attributes)
9679     r.attributes
9680       = join_attributes(r.attributes, inline_content.attributes)
9681     render_link_or_image(t, opening_index, next_link_closing_index,
9682                         closing_index, r)
9683   end
9684 end
9685
```

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.

```

9686 local function process_links_and_emphasis(t)
9687   for _,value in ipairs(t) do
9688     value.is_active = true
9689   end
9690
9691   for i,value in ipairs(t) do
9692     if not value.is_closing
9693       or value.type ~= "delimiter"
9694       or not ( value.element == "link"
9695               or value.element == "image"
9696               or value.element == "note")
9697       or value.removed then
9698       goto continue
9699     end
9700
9701     local opener_position = find_link_opener(t, 1, i - 1)
9702     if (opener_position == nil) then
9703       goto continue
9704     end
9705
9706     local opening_delimiter = t[opener_position]
9707     opening_delimiter.removed = true
9708
9709     local link_type = opening_delimiter.link_type
9710
9711     if (link_type == "inline") then
9712       resolve_inline_link(t, opener_position, i)
9713     end
9714     if (link_type == "shortcut") then
9715       resolve_shortcut_link(t, opener_position, i)
9716     end
9717     if (link_type == "full") then
9718       resolve_full_link(t, opener_position, i)
9719     end
9720     if (link_type == "collapsed") then
9721       resolve_collapsed_link(t, opener_position, i)
9722     end
9723     if (link_type == "note_inline") then
9724       resolve_note_inline_link(t, opener_position, i)
9725     end
9726     if (link_type == "raw_note") then
9727       resolve_raw_note_link(t, opener_position, i)

```

```

9728     end
9729
9730     ::continue::
9731 end
9732
9733 t[#t].content = t[#t].content:gsub("%s*$","")
9734
9735 process_emphasis(t, 1, #t)
9736 local final_result = collect_emphasis_content(t, 1, #t)
9737 return final_result
9738 end
9739
9740 function self.defer_link_and_emphasis_processing(delimiter_table)
9741     return writer.defer_call(function()
9742         return process_links_and_emphasis(delimiter_table)
9743     end)
9744 end
9745

```

### 3.1.6.8 Inline Elements (local)

```

9746 parsers.Str      = ( parsers.normalchar
9747     * (parsers.normalchar + parsers.at)^0)
9748     / writer.string
9749
9750 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
9751     / writer.string
9752
9753 parsers.Ellipsis = P("...") / writer.ellipsis
9754
9755 parsers.Smart    = parsers.Ellipsis
9756
9757 parsers.Code     = parsers.inticks / writer.code
9758
9759 if options.blankBeforeBlockquote then
9760     parsers.bqstart = parsers.fail
9761 else
9762     parsers.bqstart = parsers.blockquote_start
9763 end
9764
9765 if options.blankBeforeHeading then
9766     parsers.headerstart = parsers.fail
9767 else
9768     parsers.headerstart = parsers.atx_heading
9769 end
9770
9771 if options.blankBeforeList then

```

```

9772     parsers.interrupting_bullets = parsers.fail
9773     parsers.interrupting_enumerators = parsers.fail
9774 else
9775     parsers.interrupting_bullets
9776     = parsers.bullet(parsers.dash, true)
9777     + parsers.bullet(parsers.asterisk, true)
9778     + parsers.bullet(parsers.plus, true)
9779
9780     parsers.interrupting_enumerators
9781     = parsers.enumerator(parsers.period, true)
9782     + parsers.enumerator(parsers.rparent, true)
9783 end
9784
9785 if options.html then
9786     parsers.html_interrupting
9787     = parsers.check_trail
9788     * ( parsers.html_incomplete_open_tag
9789       + parsers.html_incomplete_close_tag
9790       + parsers.html_incomplete_open_special_tag
9791       + parsers.html_comment_start
9792       + parsers.html_cdatasection_start
9793       + parsers.html_declaration_start
9794       + parsers.html_instruction_start
9795       - parsers.html_close_special_tag
9796       - parsers.html_empty_special_tag)
9797 else
9798     parsers.html_interrupting = parsers.fail
9799 end
9800
9801 parsers.ListStarter = parsers.starter
9802
9803 parsers.EndlineExceptions
9804     = parsers.blankline -- paragraph break
9805     + parsers.eof      -- end of document
9806     + parsers.bqstart
9807     + parsers.thematic_break_lines
9808     + parsers.interrupting_bullets
9809     + parsers.interrupting_enumerators
9810     + parsers.headerstart
9811     + parsers.html_interrupting
9812
9813 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9814
9815 parsers.endline = parsers.newline
9816     * (parsers.check_minimal_indent
9817       * -V("EndlineExceptions")
9818       + parsers.check_optional_indent

```



```

9819             * -V("EndlineExceptions")
9820             * -V("ListStarter")) / function(_) return end
9821             * parsers.spacechar^0
9822
9823 parsers.Endline = parsers.endline
9824                 / writer.soft_line_break
9825
9826 parsers.EndlineNoSub = parsers.endline
9827
9828 parsers.NoSoftLineBreakEndline
9829             = parsers.newline
9830             * (parsers.check_minimal_indent
9831             * -V("NoSoftLineBreakEndlineExceptions")
9832             + parsers.check_optional_indent
9833             * -V("NoSoftLineBreakEndlineExceptions")
9834             * -V("ListStarter"))
9835             * parsers.spacechar^0
9836             / writer.space
9837
9838 parsers.EndlineBreak = parsers.backslash * parsers.endline
9839                       / writer.hard_line_break
9840
9841 parsers.OptionalIndent
9842             = parsers.spacechar^1 / writer.space
9843
9844 parsers.Space = parsers.spacechar^2 * parsers.endline
9845                / writer.hard_line_break
9846             + parsers.spacechar^1
9847             * parsers.endline^-1
9848             * parsers.eof / self.expandtabs
9849             + parsers.spacechar^1 * parsers.endline
9850                / writer.soft_line_break
9851             + parsers.spacechar^1
9852             * -parsers.newline / self.expandtabs
9853             + parsers.spacechar^1
9854
9855 parsers.NoSoftLineBreakSpace
9856             = parsers.spacechar^2 * parsers.endline
9857                / writer.hard_line_break
9858             + parsers.spacechar^1
9859             * parsers.endline^-1
9860             * parsers.eof / self.expandtabs
9861             + parsers.spacechar^1 * parsers.endline
9862                / writer.soft_line_break
9863             + parsers.spacechar^1
9864             * -parsers.newline / self.expandtabs
9865             + parsers.spacechar^1

```

```

9866
9867 parsers.NonbreakingEndline
9868         = parsers.endline
9869         / writer.nbsp
9870
9871 parsers.NonbreakingSpace
9872         = parsers.spacechar^2 * parsers.endline
9873         / writer.nbsp
9874         + parsers.spacechar^1
9875         * parsers.endline^-1 * parsers.eof / ""
9876         + parsers.spacechar^1 * parsers.endline
9877         * parsers.optionalspace
9878         / writer.nbsp
9879         + parsers.spacechar^1 * parsers.optionalspace
9880         / writer.nbsp
9881

```

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.

```

9882 function self.auto_link_url(url, attributes)
9883   return writer.link(writer.escape(url),
9884                     url, nil, attributes)
9885 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.

```

9886 function self.auto_link_email(email, attributes)
9887   return writer.link(writer.escape(email),
9888                     "mailto:".email,
9889                     nil, attributes)
9890 end
9891
9892 parsers.AutoLinkUrl = parsers.auto_link_url
9893                     / self.auto_link_url
9894
9895 parsers.AutoLinkEmail
9896         = parsers.auto_link_email
9897         / self.auto_link_email
9898
9899 parsers.AutoLinkRelativeReference
9900         = parsers.auto_link_relative_reference
9901         / self.auto_link_url
9902
9903 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9904                     / self.defer_link_and_emphasis_processing

```

```

9905
9906 parsers.EscapedChar = parsers.backslash
9907         * C(parsers.escapable) / writer.string
9908
9909 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9910         / writer.inline_html_comment
9911         + Cs(parsers.html_any_empty_inline_tag
9912             + parsers.html_inline_instruction
9913             + parsers.html_inline_cdatasection
9914             + parsers.html_inline_declaration
9915             + parsers.html_any_open_inline_tag
9916             + parsers.html_any_close_tag)
9917         / writer.inline_html_tag
9918
9919 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

9920 parsers.DisplayHtml = Cs(parsers.check_trail
9921         * ( parsers.html_comment
9922           + parsers.html_special_block
9923           + parsers.html_block
9924           + parsers.html_any_block
9925           + parsers.html_instruction
9926           + parsers.html_cdatasection
9927           + parsers.html_declaration))
9928         / writer.block_html_element
9929
9930 parsers.indented_non_blank_line = parsers.indentedline
9931         - parsers.blankline
9932
9933 parsers.Verbatim
9934     = Cs( parsers.check_code_trail
9935         * (parsers.line - parsers.blankline)
9936         * (( parsers.check_minimal_blank_indent_and_full_code_trail
9937           * parsers.blankline)^0
9938           * ( (parsers.check_minimal_indent / "")
9939             * parsers.check_code_trail
9940             * (parsers.line - parsers.blankline))^1)^0)
9941         / self.expandtabs / writer.verbatim
9942
9943 parsers.Blockquote    = parsers.blockquote_body
9944         / writer.blockquote
9945
9946 parsers.ThematicBreak = parsers.thematic_break_lines
9947         / writer.thematic_break
9948

```

```

9949 parsers.Reference = parsers.define_reference_parser
9950 / self.register_link
9951
9952 parsers.Paragraph = parsers.freeze_trail
9953 * (Ct((parsers.Inline)^1)
9954 * (parsers.newline + parsers.eof)
9955 * parsers.unfreeze_trail
9956 / writer.paragraph)
9957
9958 parsers.Plain = parsers.nonindentspace * Ct(parsers.Inline^1)
9959 / writer.plain

```

### 3.1.6.10 Lists (local)

```

9960
9961 if options.taskLists then
9962   parsers.tickbox = ( parsers.ticked_box
9963                       + parsers.halfticked_box
9964                       + parsers.unticked_box
9965                     ) / writer.tickbox
9966 else
9967   parsers.tickbox = parsers.fail
9968 end
9969
9970 parsers.list_blank = parsers.conditionally_indented_blankline
9971
9972 parsers.ref_or_block_list_separated
9973 = parsers.sep_group_no_output(parsers.list_blank)
9974 * parsers.minimally_indented_ref
9975 + parsers.block_sep_group(parsers.list_blank)
9976 * parsers.minimally_indented_block
9977
9978 parsers.ref_or_block_non_separated
9979 = parsers.minimally_indented_ref
9980 + (parsers.succeed / writer.interblocksep)
9981 * parsers.minimally_indented_block
9982 - parsers.minimally_indented_blankline
9983
9984 parsers.tight_list_loop_body_pair =
9985 parsers.create_loop_body_pair(
9986   parsers.ref_or_block_non_separated,
9987   parsers.minimally_indented_par_or_plain_no_blank,
9988   (parsers.succeed / writer.interblocksep),
9989   (parsers.succeed / writer.paragraphsep))
9990
9991 parsers.loose_list_loop_body_pair =
9992 parsers.create_loop_body_pair(

```

```

9993     parsers.ref_or_block_list_separated,
9994     parsers.minimally_indented_par_or_plain,
9995     parsers.block_sep_group(parsers.list_blank),
9996     parsers.par_sep_group(parsers.list_blank))
9997
9998 parsers.tight_list_content_loop
9999     = V("Block")
10000     * parsers.tight_list_loop_body_pair.block^0
10001     + (V("Paragraph") + V("Plain"))
10002     * parsers.ref_or_block_non_separated
10003     * parsers.tight_list_loop_body_pair.block^0
10004     + (V("Paragraph") + V("Plain"))
10005     * parsers.tight_list_loop_body_pair.par^0
10006
10007 parsers.loose_list_content_loop
10008     = V("Block")
10009     * parsers.loose_list_loop_body_pair.block^0
10010     + (V("Paragraph") + V("Plain"))
10011     * parsers.ref_or_block_list_separated
10012     * parsers.loose_list_loop_body_pair.block^0
10013     + (V("Paragraph") + V("Plain"))
10014     * parsers.loose_list_loop_body_pair.par^0
10015
10016 parsers.list_item_tightness_condition
10017     = -( parsers.list_blank^0
10018         * parsers.minimally_indented_ref_or_block_or_par)
10019     * remove_indent("li")
10020     + remove_indent("li")
10021     * parsers.fail
10022
10023 parsers.indented_content_tight
10024     = Ct( (parsers.blankline / "")
10025         * #parsers.list_blank
10026         * remove_indent("li")
10027         + ( (V("Reference") + (parsers.blankline / ""))
10028             * parsers.check_minimal_indent
10029             * parsers.tight_list_content_loop
10030             + (V("Reference") + (parsers.blankline / ""))
10031             + (parsers.tickbox^-1 / writer.escape)
10032             * parsers.tight_list_content_loop
10033             )
10034         * parsers.list_item_tightness_condition)
10035
10036 parsers.indented_content_loose
10037     = Ct( (parsers.blankline / "")
10038         * #parsers.list_blank
10039         + ( (V("Reference") + (parsers.blankline / ""))

```

```

10040     * parsers.check_minimal_indent
10041     * parsers.loose_list_content_loop
10042     + (V("Reference") + (parsers.blankline / ""))
10043     + (parsers.checkbox^-1 / writer.escape)
10044     * parsers.loose_list_content_loop))
10045
10046 parsers.TightListItem = function(starter)
10047     return -parsers.ThematicBreak
10048         * parsers.add_indent(starter, "li")
10049         * parsers.indented_content_tight
10050 end
10051
10052 parsers.LooseListItem = function(starter)
10053     return -parsers.ThematicBreak
10054         * parsers.add_indent(starter, "li")
10055         * parsers.indented_content_loose
10056         * remove_indent("li")
10057 end
10058
10059 parsers.BulletListOfType = function(bullet_type)
10060     local bullet = parsers.bullet(bullet_type)
10061     return ( Ct( parsers.TightListItem(bullet)
10062         * ( (parsers.check_minimal_indent / "")
10063             * parsers.TightListItem(bullet)
10064             )^0
10065         )
10066         * Cc(true)
10067         * -#( (parsers.list_blank^0 / "")
10068             * parsers.check_minimal_indent
10069             * (bullet - parsers.ThematicBreak)
10070         )
10071         + Ct( parsers.LooseListItem(bullet)
10072             * ( (parsers.list_blank^0 / "")
10073                 * (parsers.check_minimal_indent / "")
10074                 * parsers.LooseListItem(bullet)
10075                 )^0
10076             )
10077         * Cc(false)
10078         ) / writer.bulletlist
10079 end
10080
10081 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10082     + parsers.BulletListOfType(parsers.asterisk)
10083     + parsers.BulletListOfType(parsers.plus)
10084
10085 local function ordered_list(items,tight,starter)
10086     local startnum = starter[2][1]

```

```

10087     if options.startNumber then
10088         startnum = tonumber(startnum) or 1 -- fallback for '#'
10089         if startnum ~= nil then
10090             startnum = math.floor(startnum)
10091         end
10092     else
10093         startnum = nil
10094     end
10095     return writer.orderedlist(items,tight,startnum)
10096 end
10097
10098 parsers.OrderedListOfType = function(delimiter_type)
10099     local enumerator = parsers.enumerator(delimiter_type)
10100     return Cg(enumerator, "listtype")
10101         * (Ct( parsers.TightListItem(Cb("listtype"))
10102             * ( (parsers.check_minimal_indent / "")
10103                 * parsers.TightListItem(enumerator))^0)
10104         * Cc(true)
10105         * -#((parsers.list_blank^0 / "")
10106             * parsers.check_minimal_indent * enumerator)
10107     + Ct( parsers.LooseListItem(Cb("listtype"))
10108         * ((parsers.list_blank^0 / "")
10109             * (parsers.check_minimal_indent / "")
10110             * parsers.LooseListItem(enumerator))^0)
10111         * Cc(false)
10112     ) * Ct(Cb("listtype")) / ordered_list
10113 end
10114
10115 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10116     + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

10117 parsers.Blank           = parsers.blankline / ""
10118                         + V("Reference")

```

### 3.1.6.12 Headings (local)

```

10119 function parsers.parse_heading_text(s)
10120     local inlines = self.parser_functions.parse_inlines(s)
10121     local flatten_inlines = self.writer.flatten_inlines
10122     self.writer.flatten_inlines = true
10123     local flat_text = self.parser_functions.parse_inlines(s)
10124     flat_text = util.rope_to_string(flat_text)
10125     self.writer.flatten_inlines = flatten_inlines
10126     return {flat_text, inlines}
10127 end
10128

```

```

10129  -- parse atx header
10130  parsers.AtxHeading = parsers.check_trail_no_rem
10131                        * Cg(parsers.heading_start, "level")
10132                        * ((C( parsers.optionalspace
10133                            * parsers.hash^0
10134                            * parsers.optionalspace
10135                            * parsers.newline)
10136                          + parsers.spacechar^1
10137                          * C(parsers.line))
10138                        / strip_atx_end
10139                        / parsers.parse_heading_text)
10140                        * Cb("level")
10141                        / writer.heading
10142
10143  parsers.heading_line = parsers.linechar^1
10144                        - parsers.thematic_break_lines
10145
10146  parsers.heading_text = parsers.heading_line
10147                        * ( (V("Endline") / "\n")
10148                          * ( parsers.heading_line
10149                            - parsers.heading_level))^0
10150                        * parsers.newline^-1
10151
10152  parsers.SetextHeading = parsers.freeze_trail
10153                        * parsers.check_trail_no_rem
10154                        * #( parsers.heading_text
10155                          * parsers.check_minimal_indent
10156                          * parsers.check_trail
10157                          * parsers.heading_level)
10158                        * Cs(parsers.heading_text)
10159                        / parsers.parse_heading_text
10160                        * parsers.check_minimal_indent_and_trail
10161                        * parsers.heading_level
10162                        * parsers.newline
10163                        * parsers.unfreeze_trail
10164                        / writer.heading
10165
10166  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  $\text{\TeX}$  output.

```

10167  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.

```

10168     local walkable_syntax = (function(global_walkable_syntax)
10169         local local_walkable_syntax = {}
10170         for lhs, rule in pairs(global_walkable_syntax) do
10171             local_walkable_syntax[lhs] = util.table_copy(rule)
10172         end
10173         return local_walkable_syntax
10174     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.

```

10175     local current_extension_name = nil
10176     self.insert_pattern = function(selector, pattern, pattern_name)
10177         assert(pattern_name == nil or type(pattern_name) == "string")
10178         local _, _, lhs, pos, rhs
10179             = selector:find("^(%a+)%s+([%a%s]+%a)%s+(%a+)$")
10180         assert(lhs ~= nil,
10181             [[Expected selector in form ]]
10182             .. [[ "LHS (before|after|instead of) RHS", not "]]
10183             .. selector .. [[ "]])
10184         assert(walkable_syntax[lhs] ~= nil,
10185             [[Rule ]] .. lhs
10186             .. [[ -> ... does not exist in markdown grammar]])
10187         assert(pos == "before" or pos == "after" or pos == "instead of",
10188             [[Expected positional specifier "before", "after", ]]
10189             .. [[or "instead of", not "]]
10190             .. pos .. [[ "]])
10191         local rule = walkable_syntax[lhs]
10192         local index = nil
10193         for current_index, current_rhs in ipairs(rule) do
10194             if type(current_rhs) == "string" and current_rhs == rhs then
10195                 index = current_index
10196                 if pos == "after" then
10197                     index = index + 1
10198                 end
10199                 break
10200             end
10201         end
10202         assert(index ~= nil,
10203             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
10204             .. [[ does not exist in markdown grammar]])
10205         local accountable_pattern
10206         if current_extension_name then
10207             accountable_pattern

```

```

10208         = {pattern, current_extension_name, pattern_name}
10209     else
10210         assert(type(pattern) == "string",
10211             [[reader->insert_pattern() was called outside ]]
10212             .. [[an extension with ]]
10213             .. [[a PEG pattern instead of a rule name]])
10214         accountable_pattern = pattern
10215     end
10216     if pos == "instead of" then
10217         rule[index] = accountable_pattern
10218     else
10219         table.insert(rule, index, accountable_pattern)
10220     end
10221 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.

```

10222     local syntax =
10223         { "Blocks",
10224
10225           Blocks = V("InitializeState")
10226                 * V("ExpectedJekyllData")
10227                 * 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.

```

10228         * ( V("Block")
10229           * ( V("Blank")^0 * parsers.eof
10230             + ( V("Blank")^2 / writer.paragraphsep
10231               + V("Blank")^0 / writer.interblocksep
10232             )
10233           )
10234         + ( V("Paragraph") + V("Plain") )
10235       * ( V("Blank")^0 * parsers.eof
10236         + ( V("Blank")^2 / writer.paragraphsep
10237           + V("Blank")^0 / writer.interblocksep
10238         )
10239       )
10240     * V("Block")
10241     * ( V("Blank")^0 * parsers.eof
10242       + ( V("Blank")^2 / writer.paragraphsep
10243         + V("Blank")^0 / writer.interblocksep
10244       )
10245     )
10246     + ( V("Paragraph") + V("Plain") )
10247     * ( V("Blank")^0 * parsers.eof
10248       + V("Blank")^0 / writer.paragraphsep

```

```

10249         )
10250     )~0,
10251
10252     ExpectedJekyllData = parsers.succeed,
10253
10254     Blank                = parsers.Blank,
10255     Reference            = parsers.Reference,
10256
10257     Blockquote          = parsers.Blockquote,
10258     Verbatim            = parsers.Verbatim,
10259     ThematicBreak       = parsers.ThematicBreak,
10260     BulletList          = parsers.BulletList,
10261     OrderedList         = parsers.OrderedList,
10262     DisplayHtml         = parsers.DisplayHtml,
10263     Heading             = parsers.Heading,
10264     Paragraph           = parsers.Paragraph,
10265     Plain               = parsers.Plain,
10266
10267     ListStarter         = parsers.ListStarter,
10268     EndlineExceptions   = parsers.EndlineExceptions,
10269     NoSoftLineBreakEndlineExceptions
10270         = parsers.NoSoftLineBreakEndlineExceptions,
10271
10272     Str                 = parsers.Str,
10273     Space               = parsers.Space,
10274     NoSoftLineBreakSpace
10275         = parsers.NoSoftLineBreakSpace,
10276     OptionalIndent     = parsers.OptionalIndent,
10277     Endline             = parsers.Endline,
10278     EndlineNoSub       = parsers.EndlineNoSub,
10279     NoSoftLineBreakEndline
10280         = parsers.NoSoftLineBreakEndline,
10281     EndlineBreak       = parsers.EndlineBreak,
10282     LinkAndEmph        = parsers.LinkAndEmph,
10283     Code               = parsers.Code,
10284     AutoLinkUrl        = parsers.AutoLinkUrl,
10285     AutoLinkEmail      = parsers.AutoLinkEmail,
10286     AutoLinkRelativeReference
10287         = parsers.AutoLinkRelativeReference,
10288     InlineHtml         = parsers.InlineHtml,
10289     HtmlEntity         = parsers.HtmlEntity,
10290     EscapedChar        = parsers.EscapedChar,
10291     Smart              = parsers.Smart,
10292     Symbol             = parsers.Symbol,
10293     SpecialChar        = parsers.fail,
10294     InitializeState    = parsers.succeed,
10295 }

```

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].

```

10296     self.update_rule = function(rule_name, get_pattern)
10297         assert(current_extension_name ~= nil)
10298         assert(syntax[rule_name] ~= nil,
10299             [[Rule ]] .. rule_name
10300             .. [[ -> ... does not exist in markdown grammar]])
10301         local previous_pattern
10302         local extension_name
10303         if walkable_syntax[rule_name] then
10304             local previous_accountable_pattern
10305                 = walkable_syntax[rule_name][1]
10306             previous_pattern = previous_accountable_pattern[1]
10307             extension_name
10308                 = previous_accountable_pattern[2]
10309             .. ", " .. current_extension_name
10310         else
10311             previous_pattern = nil
10312             extension_name = current_extension_name
10313         end
10314         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

```

```

10315     if type(get_pattern) == "function" then
10316         pattern = get_pattern(previous_pattern)
10317     else
10318         assert(previous_pattern == nil,
10319             [[Rule ]] .. rule_name ..
10320             [[ has already been updated by ]] .. extension_name)
10321         pattern = get_pattern
10322     end
10323     local accountable_pattern = { pattern, extension_name, rule_name }
10324     walkable_syntax[rule_name] = { accountable_pattern }
10325 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.

```
10326     local special_characters = {}
10327     self.add_special_character = function(c)
10328         table.insert(special_characters, c)
10329         syntax.SpecialChar = S(table.concat(special_characters, ""))
10330     end
10331
10332     self.add_special_character("*")
10333     self.add_special_character("[")
10334     self.add_special_character("]")
10335     self.add_special_character("<")
10336     self.add_special_character("!")
10337     self.add_special_character("\\")
```

Add method `reader->initialize_named_group` that defines named groups with a default capture value.

```
10338     self.initialize_named_group = function(name, value)
10339         local pattern = Ct("")
10340         if value ~= nil then
10341             pattern = pattern / value
10342         end
10343         syntax.InitializeState = syntax.InitializeState
10344                                 * Cg(pattern, name)
10345     end
```

Add a named group for indentation.

```
10346     self.initialize_named_group("indent_info")
```

Apply syntax extensions.

```
10347     for _, extension in ipairs(extensions) do
10348         current_extension_name = extension.name
10349         extension.extend_writer(writer)
10350         extension.extend_reader(self)
10351     end
10352     current_extension_name = nil
```

If the `debugExtensions` option is enabled, serialize `walkable_syntax` to a JSON for debugging purposes.

```
10353     if options.debugExtensions then
10354         local sorted_lhs = {}
10355         for lhs, _ in pairs(walkable_syntax) do
10356             table.insert(sorted_lhs, lhs)
10357         end
10358         table.sort(sorted_lhs)
10359
10360         local output_lines = {"{"}
```

```

10361     for lhs_index, lhs in ipairs(sorted_lhs) do
10362         local encoded_lhs = util.encode_json_string(lhs)
10363         table.insert(output_lines, [[      ]] .. encoded_lhs .. [[: ]])
10364         local rule = walkable_syntax[lhs]
10365         for rhs_index, rhs in ipairs(rule) do
10366             local human_readable_rhs
10367             if type(rhs) == "string" then
10368                 human_readable_rhs = rhs
10369             else
10370                 local pattern_name
10371                 if rhs[3] then
10372                     pattern_name = rhs[3]
10373                 else
10374                     pattern_name = "Anonymous Pattern"
10375                 end
10376                 local extension_name = rhs[2]
10377                 human_readable_rhs = pattern_name .. [[ (]]
10378                     .. extension_name .. [[]]
10379             end
10380             local encoded_rhs
10381                 = util.encode_json_string(human_readable_rhs)
10382             local output_line = [[      ]] .. encoded_rhs
10383             if rhs_index < #rule then
10384                 output_line = output_line .. ","
10385             end
10386             table.insert(output_lines, output_line)
10387         end
10388         local output_line = "    ]"
10389         if lhs_index < #sorted_lhs then
10390             output_line = output_line .. ","
10391         end
10392         table.insert(output_lines, output_line)
10393     end
10394     table.insert(output_lines, "}")
10395
10396     local output = table.concat(output_lines, "\n")
10397     local output_filename = options.debugExtensionsFileName
10398     local output_file = assert(io.open(output_filename, "w"),
10399         [[Could not open file ]] .. output_filename
10400         .. [[ for writing]])
10401     assert(output_file:write(output))
10402     assert(output_file:close())
10403 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`.

```

10404     for lhs, rule in pairs(walkable_syntax) do
10405         syntax[lhs] = parsers.fail
10406         for _, rhs in ipairs(rule) do
10407             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.

```

10408         if type(rhs) == "string" then
10409             pattern = V(rhs)
10410         else
10411             pattern = rhs[1]
10412             if type(pattern) == "string" then
10413                 pattern = V(pattern)
10414             end
10415         end
10416         syntax[lhs] = syntax[lhs] + pattern
10417     end
10418 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.

```

10419     if options.underscores then
10420         self.add_special_character("_")
10421     end
10422
10423     if not options.codeSpans then
10424         syntax.Code = parsers.fail
10425     else
10426         self.add_special_character("`")
10427     end
10428
10429     if not options.html then
10430         syntax.DisplayHtml = parsers.fail
10431         syntax.InlineHtml = parsers.fail
10432         syntax.HtmlEntity = parsers.fail
10433     else
10434         self.add_special_character("&")
10435     end
10436
10437     if options.preserveTabs then
10438         options.stripIndent = false
10439     end
10440
10441     if not options.smartEllipses then

```

```

10442     syntax.Smart = parsers.fail
10443 else
10444     self.add_special_character(".")
10445 end
10446
10447 if not options.relativeReferences then
10448     syntax.AutoLinkRelativeReference = parsers.fail
10449 end
10450
10451 if options.contentLevel == "inline" then
10452     syntax[1] = "Inlines"
10453     syntax.Inlines = V("InitializeState")
10454         * parsers.Inline^0
10455         * ( parsers.spacing^0
10456             * parsers.eof / "" )
10457     syntax.Space = parsers.Space + parsers.blankline / writer.space
10458 end
10459
10460 local blocks_nested_t = util.table_copy(syntax)
10461 blocks_nested_t.ExpectedJekyllData = parsers.succeed
10462 parsers.blocks_nested = Ct(blocks_nested_t)
10463
10464 parsers.blocks = Ct(syntax)
10465
10466 local inlines_t = util.table_copy(syntax)
10467 inlines_t[1] = "Inlines"
10468 inlines_t.Inlines = V("InitializeState")
10469     * parsers.Inline^0
10470     * ( parsers.spacing^0
10471         * parsers.eof / "" )
10472 parsers.inlines = Ct(inlines_t)
10473
10474 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10475 inlines_no_inline_note_t.InlineNote = parsers.fail
10476 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10477
10478 local inlines_no_html_t = util.table_copy(inlines_t)
10479 inlines_no_html_t.DisplayHtml = parsers.fail
10480 inlines_no_html_t.InlineHtml = parsers.fail
10481 inlines_no_html_t.HtmlEntity = parsers.fail
10482 parsers.inlines_no_html = Ct(inlines_no_html_t)
10483
10484 local inlines_nbsp_t = util.table_copy(inlines_t)
10485 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10486 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10487 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10488

```



```

10489     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10490     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10491     inlines_no_link_or_emphasis_t.EndlineExceptions
10492     = parsers.EndlineExceptions - parsers.eof
10493     parsers.inlines_no_link_or_emphasis
10494     = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain TeX output and returns it..

```

10495     return function(input)
Unicode-normalize the input.
10496         if options.unicodeNormalization then
10497             local form = options.unicodeNormalizationForm
10498             if form == "nfc" then
10499                 input = uni_algos.normalize.NFC(input)
10500             elseif form == "nfd" then
10501                 input = uni_algos.normalize.NFD(input)
10502             elseif form == "nfkc" then
10503                 input = uni_algos.normalize.NFKC(input)
10504             elseif form == "nfkd" then
10505                 input = uni_algos.normalize.NFKD(input)
10506             else
10507                 return writer.error(
10508                     format("Unknown normalization form %s.", form))
10509             end
10510         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.

```

10511         input = input:gsub("\r\n?", "\n")
10512         if input:sub(-1) ~= "\n" then
10513             input = input .. "\n"
10514         end

```

Clear the table of references.

```

10515         references = {}
10516         local document = self.parser_functions.parse_blocks(input)
10517         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`.

```

10518         local undosep_start, undosep_end
10519         local potential_secend_start, secend_start
10520         local potential_sep_start, sep_start
10521         while true do
10522             -- find a `writer->undosep`
10523             undosep_start, undosep_end

```

```

10524         = output:find(writer.undosep_text, 1, true)
10525     if undosep_start == nil then break end
10526     -- skip any preceding section ends
10527     secend_start = undosep_start
10528     while true do
10529         potential_secend_start = secend_start - #writer.secend_text
10530         if potential_secend_start < 1
10531             or output:sub(potential_secend_start,
10532                 secend_start - 1) ~= writer.secend_text
10533             then
10534                 break
10535             end
10536             secend_start = potential_secend_start
10537         end
10538         -- find an immediately preceding
10539         -- block element / paragraph separator
10540         sep_start = secend_start
10541         potential_sep_start = sep_start - #writer.interblocksep_text
10542         if potential_sep_start >= 1
10543             and output:sub(potential_sep_start,
10544                 sep_start - 1) == writer.interblocksep_text
10545             then
10546                 sep_start = potential_sep_start
10547             else
10548                 potential_sep_start = sep_start - #writer.paragraphsep_text
10549                 if potential_sep_start >= 1
10550                     and output:sub(potential_sep_start,
10551                         sep_start - 1) == writer.paragraphsep_text
10552                     then
10553                         sep_start = potential_sep_start
10554                     end
10555                 end
10556                 -- remove `writer->undosep` and immediately preceding
10557                 -- block element / paragraph separator
10558                 output = output:sub(1, sep_start - 1)
10559                 .. output:sub(secend_start, undosep_start - 1)
10560                 .. output:sub(undosep_end + 1)
10561             end
10562             return output
10563         end
10564     end
10565     return self
10566 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.

```
10567 M.extensions = {}
```

### 3.1.7.1 Bracketed Spans

The `extensions.bracketed_spans` function implements the Pandoc bracketed span syntax extension.

```
10568 M.extensions.bracketed_spans = function()
10569   return {
10570     name = "built-in bracketed_spans syntax extension",
10571     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.

```
10572     function self.span(s, attr)
10573       if self.flatten_inlines then return s end
10574       return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10575             self.attributes(attr),
10576             s,
10577             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10578   end
10579 end, extend_reader = function(self)
10580   local parsers = self.parsers
10581   local writer = self.writer
10582
10583   local span_label = parsers.lbracket
10584                     * (Cs((parsers.alphanumeric^1
10585                         + parsers.inticks
10586                         + parsers.autolink
10587                         + V("InlineHtml")
10588                         + ( parsers.backslash * parsers.backslash)
10589                         + ( parsers.backslash
10590                           * (parsers.lbracket + parsers.rbracket)
10591                           + V("Space") + V("Endline")
10592                           + (parsers.any
10593                             - ( parsers.newline
10594                               + parsers.lbracket
10595                               + parsers.rbracket
10596                               + parsers.blankline^2))))^1)
10597                     / self.parser_functions.parse_inlines)
10598                     * parsers.rbracket
10599
10600   local Span = span_label
10601             * Ct(parsers.attributes)
```

```

10602             / writer.span
10603
10604         self.insert_pattern("Inline before LinkAndEmph",
10605                             Span, "Span")
10606     end
10607 }
10608 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.

```

10609 M.extensions.citations = function(citation_nbsps)
10610   return {
10611     name = "built-in citations syntax extension",
10612     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.

```

10613     function self.citations(text_cites, cites)
10614       local buffer = {}
10615       if self.flatten_inlines then
10616         for _,cite in ipairs(cites) do
10617           if cite.prenote then
10618             table.insert(buffer, {cite.prenote, " "})
10619           end
10620           table.insert(buffer, cite.name)
10621           if cite.postnote then
10622             table.insert(buffer, {" ", cite.postnote})
10623           end
10624         end
10625       else

```

```

10626         table.insert(buffer,
10627             {"\\markdownRenderer",
10628             text_cites and "TextCite" or "Cite",
10629             "{", #cites, "}"}))
10630     for _,cite in ipairs(cites) do
10631         table.insert(buffer,
10632             {cite.suppress_author and "-" or "+", "{",
10633             cite.prenote or "", "}{" ,
10634             cite.postnote or "", "}{" , cite.name, "}"}))
10635     end
10636 end
10637 return buffer
10638 end
10639 end, extend_reader = function(self)
10640     local parsers = self.parsers
10641     local writer = self.writer
10642
10643     local citation_chars
10644         = parsers.alphanumeric
10645         + S("#$%&-+<>~/_")
10646
10647     local citation_name
10648         = Cs(parsers.dash~-1) * parsers.at
10649         * Cs(citation_chars
10650             * ((( citation_chars
10651                 + parsers.internal_punctuation
10652                 - parsers.comma - parsers.semicolon)
10653             * -#(( parsers.internal_punctuation
10654                 - parsers.comma
10655                 - parsers.semicolon)^0
10656             * -( citation_chars
10657                 + parsers.internal_punctuation
10658                 - parsers.comma
10659                 - parsers.semicolon)))^0
10660             * citation_chars)^-1)
10661
10662     local citation_body_prenote
10663         = Cs((parsers.alphanumeric^1
10664             + parsers.bracketed
10665             + parsers.inticks
10666             + parsers.autolink
10667             + V("InlineHtml")
10668             + V("Space") + V("EndlineNoSub")
10669             + (parsers.anyescaped
10670             - ( parsers.newline
10671                 + parsers.rbracket
10672                 + parsers.blankline^2))

```

```

10673         - ( parsers.spnl
10674           * parsers.dash^-1
10675           * parsers.at))^1)
10676
10677     local citation_body_postnote
10678         = Cs((parsers.alphanumeric^1
10679           + parsers.bracketed
10680           + parsers.inticks
10681           + parsers.autolink
10682           + V("InlineHtml")
10683           + V("Space") + V("EndlineNoSub"))
10684           + (parsers.anyescaped
10685             - ( parsers.newline
10686               + parsers.rbracket
10687               + parsers.semicolon
10688               + parsers.blankline^2))
10689             - (parsers.spnl * parsers.rbracket))^1)
10690
10691     local citation_body_chunk
10692         = ( citation_body_prenote
10693           * parsers.spnlc_sep
10694           + Cc("")
10695           * parsers.spnlc
10696         )
10697         * citation_name
10698         * ( parsers.internal_punctuation
10699           - parsers.semicolon)^-1
10700         * ( parsers.spnlc / function(_) return end
10701           * citation_body_postnote
10702           + Cc("")
10703           * parsers.spnlc
10704         )
10705
10706     local citation_body
10707         = citation_body_chunk
10708         * ( parsers.semicolon
10709           * parsers.spnlc
10710           * citation_body_chunk
10711         )^0
10712
10713     local citation_headless_body_postnote
10714         = Cs((parsers.alphanumeric^1
10715           + parsers.bracketed
10716           + parsers.inticks
10717           + parsers.autolink
10718           + V("InlineHtml")
10719           + V("Space") + V("Endline"))

```

```

10720         + (parsers.anyescaped
10721         - ( parsers.newline
10722         + parsers.rbracket
10723         + parsers.at
10724         + parsers.semicolon + parsers.blankline^2))
10725         - (parsers.spnl * parsers.rbracket))^0)
10726
10727     local citation_headless_body
10728         = citation_headless_body_postnote
10729         * ( parsers.semicolon
10730         * parsers.spnlc
10731         * citation_body_chunk
10732         )^0
10733
10734     local citations
10735         = function(text_cites, raw_cites)
10736         local function normalize(str)
10737             if str == "" then
10738                 str = nil
10739             else
10740                 str = (citation_nbsps and
10741                 self.parser_functions.parse_inlines_nbsp or
10742                 self.parser_functions.parse_inlines)(str)
10743             end
10744             return str
10745         end
10746
10747         local cites = {}
10748         for i = 1,#raw_cites,4 do
10749             cites[#cites+1] = {
10750                 prenote = normalize(raw_cites[i]),
10751                 suppress_author = raw_cites[i+1] == "-",
10752                 name = writer.identifier(raw_cites[i+2]),
10753                 postnote = normalize(raw_cites[i+3]),
10754             }
10755         end
10756         return writer.citations(text_cites, cites)
10757     end
10758
10759     local TextCitations
10760         = Ct((parsers.spnlc
10761         * Cc("")
10762         * citation_name
10763         * ((parsers.spnlc
10764         * parsers.lbracket
10765         * citation_headless_body
10766         * parsers.rbracket) + Cc("")))^1)

```

```

10767         / function(raw_cites)
10768             return citations(true, raw_cites)
10769         end
10770
10771     local ParenthesizedCitations
10772         = Ct((parsers.spnlc
10773             * parsers.lbracket
10774             * citation_body
10775             * parsers.rbracket)^1)
10776     / function(raw_cites)
10777         return citations(false, raw_cites)
10778     end
10779
10780     local Citations = TextCitations + ParenthesizedCitations
10781
10782     self.insert_pattern("Inline before LinkAndEmph",
10783         Citations, "Citations")
10784
10785     self.add_special_character("@")
10786     self.add_special_character("-")
10787 end
10788 }
10789 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.

```
10790 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.

```

10791 local languages_json = (function()
10792     local base, prev, curr
10793     for _, pathname in ipairs{kpse.lookup(language_map,
10794         {all=true})} do
10795         local file = io.open(pathname, "r")
10796         if not file then goto continue end
10797         local input = assert(file:read("*a"))
10798         assert(file:close())
10799         local json = input:gsub('"[^\\n]-"', '[%1]=')
10800         curr = load("_ENV = {}; return "..json")()
10801         if type(curr) == "table" then
10802             if base == nil then

```



```

10803         base = curr
10804     else
10805         setmetatable(prev, { __index = curr })
10806     end
10807     prev = curr
10808 end
10809 ::continue::
10810 end
10811 return base or {}
10812 end)()
10813
10814 return {
10815     name = "built-in content_blocks syntax extension",
10816     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.

```

10817     function self.contentblock(src,suf,type,tit)
10818         if not self.is_writing then return "" end
10819         src = src.." "..suf
10820         suf = suf:lower()
10821         if type == "onlineimage" then
10822             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10823                 "{" ,self.string(src),"} ",
10824                 "{" ,self.uri(src),"} ",
10825                 "{" ,self.string(tit or ""),"} "}
10826         elseif languages_json[suf] then
10827             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10828                 "{" ,self.string(languages_json[suf]),"} ",
10829                 "{" ,self.string(src),"} ",
10830                 "{" ,self.uri(src),"} ",
10831                 "{" ,self.string(tit or ""),"} "}
10832         else
10833             return {"\\markdownRendererContentBlock{" ,suf,"} ",
10834                 "{" ,self.string(src),"} ",
10835                 "{" ,self.uri(src),"} ",
10836                 "{" ,self.string(tit or ""),"} "}
10837         end
10838     end
10839 end, extend_reader = function(self)
10840     local parsers = self.parsers
10841     local writer = self.writer
10842
10843     local contentblock_tail
10844         = parsers.optionaltitle

```

```

10845         * (parsers.newline + parsers.eof)
10846
10847     -- case insensitive online image suffix:
10848     local onlineimagesuffix
10849         = (function(...)
10850             local parser = nil
10851             for _, suffix in ipairs({...}) do
10852                 local pattern=nil
10853                 for i=1,#suffix do
10854                     local char=suffix:sub(i,i)
10855                     char = S(char:lower()..char:upper())
10856                     if pattern == nil then
10857                         pattern = char
10858                     else
10859                         pattern = pattern * char
10860                     end
10861                 end
10862                 if parser == nil then
10863                     parser = pattern
10864                 else
10865                     parser = parser + pattern
10866                 end
10867             end
10868             return parser
10869         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10870
10871     -- online image url for iA Writer content blocks with
10872     -- mandatory suffix, allowing nested brackets:
10873     local onlineimageurl
10874         = (parsers.less
10875             * Cs((parsers.anyescaped
10876                 - parsers.more
10877                 - parsers.spacing
10878                 - #(parsers.period
10879                     * onlineimagesuffix
10880                     * parsers.more
10881                     * contentblock_tail))^0)
10882             * parsers.period
10883             * Cs(onlineimagesuffix)
10884             * parsers.more
10885             + (Cs((parsers.inparens
10886                 + (parsers.anyescaped
10887                     - parsers.spacing
10888                     - parsers.rparent
10889                     - #(parsers.period
10890                         * onlineimagesuffix
10891                         * contentblock_tail))))^0)

```

```

10892         * parsers.period
10893         * Cs(onlineimagesuffix))
10894     ) * Cc("onlineimage")
10895
10896     -- filename for iA Writer content blocks with mandatory suffix:
10897     local localfilepath
10898         = parsers.slash
10899         * Cs((parsers.anyescaped
10900             - parsers.tab
10901             - parsers.newline
10902             - #(parsers.period
10903                 * parsers.alphanumeric^1
10904                 * contentblock_tail))^1)
10905         * parsers.period
10906         * Cs(parsers.alphanumeric^1)
10907         * Cc("localfile")
10908
10909     local ContentBlock
10910         = parsers.check_trail_no_rem
10911         * (localfilepath + onlineimageurl)
10912         * contentblock_tail
10913         / writer.contentblock
10914
10915     self.insert_pattern("Block before Blockquote",
10916                       ContentBlock, "ContentBlock")
10917 end
10918 }
10919 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.

```

10920 M.extensions.definition_lists = function(tight_lists)
10921   return {
10922     name = "built-in definition_lists syntax extension",
10923     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.

```

10924     local function dlitem(term, defs)
10925       local retVal = {"\\markdownRendererDlItem{",term,""}
10926       for _, def in ipairs(defs) do
10927         retVal[#retVal+1]

```

```

10928         = {"\\markdownRendererDlDefinitionBegin ",def,
10929             "\\markdownRendererDlDefinitionEnd "}
10930     end
10931     retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10932     return retVal
10933 end
10934
10935 function self.definitionlist(items,tight)
10936     if not self.is_writing then return "" end
10937     local buffer = {}
10938     for _,item in ipairs(items) do
10939         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
10940     end
10941     if tight and tight_lists then
10942         return {"\\markdownRendererDlBeginTight\n", buffer,
10943             "\n\\markdownRendererDlEndTight"}
10944     else
10945         return {"\\markdownRendererDlBegin\n", buffer,
10946             "\n\\markdownRendererDlEnd"}
10947     end
10948 end
10949 end, extend_reader = function(self)
10950     local parsers = self.parsers
10951     local writer = self.writer
10952
10953     local defstartchar = S("~:")
10954
10955     local defstart
10956     = parsers.check_trail_length(0) * defstartchar
10957     * #parsers.spacing
10958     * (parsers.tab + parsers.space^-3)
10959     + parsers.check_trail_length(1)
10960     * defstartchar * #parsers.spacing
10961     * (parsers.tab + parsers.space^-2)
10962     + parsers.check_trail_length(2)
10963     * defstartchar * #parsers.spacing
10964     * (parsers.tab + parsers.space^-1)
10965     + parsers.check_trail_length(3)
10966     * defstartchar * #parsers.spacing
10967
10968     local indented_line
10969     = (parsers.check_minimal_indent / "")
10970     * parsers.check_code_trail * parsers.line
10971
10972     local blank
10973     = parsers.check_minimal_blank_indent_and_any_trail
10974     * parsers.optionalspace * parsers.newline

```

```

10975
10976     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10977
10978     local indented_blocks = function(bl)
10979         return Cs( bl
10980             * (blank^1 * (parsers.check_minimal_indent / ""))
10981             * parsers.check_code_trail * -parsers.blankline * bl)^0
10982             * (blank^1 + parsers.eof))
10983     end
10984
10985     local function definition_list_item(term, defs, _)
10986         return { term = self.parser_functions.parse_inlines(term),
10987             definitions = defs }
10988     end
10989
10990     local DefinitionListItemLoose
10991         = C(parsers.line) * blank^0
10992         * Ct((parsers.check_minimal_indent * (defstart
10993             * indented_blocks(dlchunk)
10994             / self.parser_functions.parse_blocks_nested))^1)
10995         * Cc(false) / definition_list_item
10996
10997     local DefinitionListItemTight
10998         = C(parsers.line)
10999         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
11000             / self.parser_functions.parse_blocks_nested))^1)
11001         * Cc(true) / definition_list_item
11002
11003     local DefinitionList
11004         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
11005         + Ct(DefinitionListItemTight^1)
11006         * (blank^0
11007             * -DefinitionListItemLoose * Cc(true))
11008         ) / writer.definitionlist
11009
11010     self.insert_pattern("Block after Heading",
11011         DefinitionList, "DefinitionList")
11012 end
11013 }
11014 end

```

### 3.1.7.5 Fancy Lists

The `extensions.fancy_lists` function implements the Pandoc fancy list syntax extension.

```

11015 M.extensions.fancy_lists = function()
11016     return {

```

```

11017     name = "built-in fancy_lists syntax extension",
11018     extend_writer = function(self)
11019         local options = self.options
11020

```

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.

```

11021     function self.fancylist(items,tight,startnum,numstyle,numdelim)
11022         if not self.is_writing then return "" end
11023         local buffer = {}
11024         local num = startnum
11025         for _,item in ipairs(items) do
11026             if item ~= "" then
11027                 buffer[#buffer + 1] = self.fancyitem(item,num)
11028             end
11029             if num ~= nil and item ~= "" then
11030                 num = num + 1
11031             end
11032         end
11033         local contents = util.intersperse(buffer,"\n")
11034         if tight and options.tightLists then
11035             return {"\markdownRendererFancyOlBeginTight{",
11036                 numstyle,"}{",numdelim,"}",contents,
11037                 "\n\markdownRendererFancyOlEndTight "}
11038         else
11039             return {"\markdownRendererFancyOlBegin{",
11040                 numstyle,"}{",numdelim,"}",contents,
11041                 "\n\markdownRendererFancyOlEnd "}

```

```

11042         end
11043     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.

```

11044     function self.fancyitem(s,num)
11045         if num ~= nil then
11046             return {"\\markdownRendererFancyOListItemWithNumber{" ,num,"} ",s,
11047                 "\\markdownRendererFancyOListItemEnd "}
11048         else
11049             return {"\\markdownRendererFancyOItem ",s,
11050                 "\\markdownRendererFancyOItemEnd "}
11051         end
11052     end
11053 end, extend_reader = function(self)
11054     local parsers = self.parsers
11055     local options = self.options
11056     local writer = self.writer
11057
11058     local function combine_markers_and_delims(markers, delims)
11059         local markers_table = {}
11060         for _,marker in ipairs(markers) do
11061             local start_marker
11062             local continuation_marker
11063             if type(marker) == "table" then
11064                 start_marker = marker[1]
11065                 continuation_marker = marker[2]
11066             else
11067                 start_marker = marker
11068                 continuation_marker = marker
11069             end
11070             for _,delim in ipairs(delims) do
11071                 table.insert(markers_table,
11072                     {start_marker, continuation_marker, delim})
11073             end
11074         end
11075         return markers_table
11076     end
11077
11078     local function join_table_with_func(func, markers_table)
11079         local pattern = func(table.unpack(markers_table[1]))
11080         for i = 2, #markers_table do
11081             pattern = pattern + func(table.unpack(markers_table[i]))
11082         end
11083         return pattern
11084     end

```

```

11085
11086     local lowercase_letter_marker = R("az")
11087     local uppercase_letter_marker = R("AZ")
11088
11089     local roman_marker = function(chars)
11090         local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11091         local l, x, v, i
11092             = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11093         return  m-3
11094             * (c*m + c*d + d-1 * c-3)
11095             * (x*c + x*l + l-1 * x-3)
11096             * (i*x + i*v + v-1 * i-3)
11097     end
11098
11099     local lowercase_roman_marker
11100         = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11101     local uppercase_roman_marker
11102         = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11103
11104     local lowercase_opening_roman_marker = P("i")
11105     local uppercase_opening_roman_marker = P("I")
11106
11107     local digit_marker = parsers.dig * parsers.dig-8
11108
11109     local markers = {
11110         {lowercase_opening_roman_marker, lowercase_roman_marker},
11111         {uppercase_opening_roman_marker, uppercase_roman_marker},
11112         lowercase_letter_marker,
11113         uppercase_letter_marker,
11114         lowercase_roman_marker,
11115         uppercase_roman_marker,
11116         digit_marker
11117     }
11118
11119     local delims = {
11120         parsers.period,
11121         parsers.rparent
11122     }
11123
11124     local markers_table = combine_markers_and_delims(markers, delims)
11125
11126     local function enumerator(start_marker, _,
11127                             delimiter_type, interrupting)
11128         local delimiter_range
11129         local allowed_end
11130         if interrupting then
11131             delimiter_range = P("1")

```



```

11132         allowed_end = C(parsers.spacechar~1) * #parsers.linechar
11133     else
11134         delimiter_range = start_marker
11135         allowed_end = C(parsers.spacechar~1)
11136             + #(parsers.newline + parsers.eof)
11137     end
11138
11139     return parsers.check_trail
11140         * Ct(C(delimiter_range) * C(delimiter_type))
11141         * allowed_end
11142 end
11143
11144 local starter = join_table_with_func(enumerator, markers_table)
11145
11146 local TightListItem = function(starter)
11147     return parsers.add_indent(starter, "li")
11148         * parsers.indented_content_tight
11149 end
11150
11151 local LooseListItem = function(starter)
11152     return parsers.add_indent(starter, "li")
11153         * parsers.indented_content_loose
11154         * remove_indent("li")
11155 end
11156
11157 local function roman2number(roman)
11158     local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11159         ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11160     local numeral = 0
11161
11162     local i = 1
11163     local len = string.len(roman)
11164     while i < len do
11165         local z1, z2 = romans[ string.sub(roman, i, i) ],
11166             romans[ string.sub(roman, i+1, i+1) ]
11167         if z1 < z2 then
11168             numeral = numeral + (z2 - z1)
11169             i = i + 2
11170         else
11171             numeral = numeral + z1
11172             i = i + 1
11173         end
11174     end
11175     if i <= len then
11176         numeral = numeral + romans[ string.sub(roman,i,i) ]
11177     end
11178     return numeral

```

```

11179     end
11180
11181     local function sniffstyle(numstr, delimend)
11182         local numdelim
11183         if delimend == ")" then
11184             numdelim = "OneParen"
11185         elseif delimend == "." then
11186             numdelim = "Period"
11187         else
11188             numdelim = "Default"
11189         end
11190
11191         local num
11192         num = numstr:match("^([I])$")
11193         if num then
11194             return roman2number(num), "UpperRoman", numdelim
11195         end
11196         num = numstr:match("^([i])$")
11197         if num then
11198             return roman2number(string.upper(num)), "LowerRoman", numdelim
11199         end
11200         num = numstr:match("^([A-Z])$")
11201         if num then
11202             return string.byte(num) - string.byte("A") + 1,
11203                "UpperAlpha", numdelim
11204         end
11205         num = numstr:match("^([a-z])$")
11206         if num then
11207             return string.byte(num) - string.byte("a") + 1,
11208                "LowerAlpha", numdelim
11209         end
11210         num = numstr:match("^([IVXLCDM]+)")
11211         if num then
11212             return roman2number(num), "UpperRoman", numdelim
11213         end
11214         num = numstr:match("^([ivxlcdm]+)")
11215         if num then
11216             return roman2number(string.upper(num)), "LowerRoman", numdelim
11217         end
11218         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11219     end
11220
11221     local function fancylist(items,tight,start)
11222         local startnum, numstyle, numdelim
11223         = sniffstyle(start[2][1], start[2][2])
11224         return writer.fancylist(items,tight,
11225                                options.startNumber and startnum or 1,

```

```

11226             numstyle or "Decimal",
11227             numdelim or "Default")
11228     end
11229
11230     local FancyListOfType
11231     = function(start_marker, continuation_marker, delimiter_type)
11232         local enumerator_start
11233         = enumerator(start_marker, continuation_marker,
11234                     delimiter_type)
11235         local enumerator_cont
11236         = enumerator(continuation_marker, continuation_marker,
11237                     delimiter_type)
11238         return Cg(enumerator_start, "listtype")
11239             * (Ct( TightListItem(Cb("listtype"))
11240                 * ((parsers.check_minimal_indent / "")
11241                   * TightListItem(enumerator_cont))^0)
11242             * Cc(true)
11243             * -#((parsers.conditionally_indented_blankline^0 / "")
11244                 * parsers.check_minimal_indent * enumerator_cont)
11245             + Ct( LooseListItem(Cb("listtype"))
11246                 * ((parsers.conditionally_indented_blankline^0 / "")
11247                   * (parsers.check_minimal_indent / "")
11248                   * LooseListItem(enumerator_cont))^0)
11249             * Cc(false)
11250             ) * Ct(Cb("listtype")) / fancylist
11251     end
11252
11253     local FancyList
11254     = join_table_with_func(FancyListOfType, markers_table)
11255
11256     local ListStarter = starter
11257
11258     self.update_rule("OrderedList", FancyList)
11259     self.update_rule("ListStarter", ListStarter)
11260 end
11261 }
11262 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.

```
11263 M.extensions.fenced_code = function(blank_before_code_fence,  
11264                                     allow_attributes,  
11265                                     allow_raw_blocks)  
11266   return {  
11267     name = "built-in fenced_code syntax extension",  
11268     extend_writer = function(self)  
11269       local options = self.options  
11270
```

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.

```
11271     function self.fencedCode(s, i, attr)  
11272       if not self.is_writing then return "" end  
11273       s = s:gsub("\n$", "")  
11274       local buf = {}  
11275       if attr ~= nil then  
11276         table.insert(buf,  
11277           {"\\markdownRendererFencedCodeAttributeContextBegin",  
11278             self.attributes(attr)})  
11279       end  
11280       local name = util.cache_verbatim(options.cacheDir, s)  
11281       table.insert(buf,  
11282         {"\\markdownRendererInputFencedCode{",  
11283           name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}"}  
11284       if attr ~= nil then  
11285         table.insert(buf,  
11286           "\\markdownRendererFencedCodeAttributeContextEnd{")  
11287       end  
11288       return buf  
11289     end  
11290
```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```
11291     if allow_raw_blocks then  
11292       function self.rawBlock(s, attr)  
11293         if not self.is_writing then return "" end  
11294         s = s:gsub("\n$", "")  
11295         local name = util.cache_verbatim(options.cacheDir, s)  
11296         return {"\\markdownRendererInputRawBlock{",  
11297           name,"}{" , self.string(attr),"}"}  
11298       end  
11299     end  
11300   end, extend_reader = function(self)  
11301     local parsers = self.parsers
```

```

11302     local writer = self.writer
11303
11304     local function captures_geq_length(_,i,a,b)
11305         return #a >= #b and i
11306     end
11307
11308     local function strip_enclosing_whitespaces(str)
11309         return str:gsub("^%s*(.)%s*$", "%1")
11310     end
11311
11312     local tilde_infostring = Cs(Cs((V("HtmlEntity")
11313         + parsers.anyescaped
11314         - parsers.newline)^0)
11315         / strip_enclosing_whitespaces)
11316
11317     local backtick_infostring
11318         = Cs( Cs((V("HtmlEntity")
11319             + ( -#(parsers.backslash * parsers.backtick)
11320                 * parsers.anyescaped)
11321                 - parsers.newline
11322                 - parsers.backtick)^0)
11323         / strip_enclosing_whitespaces)
11324
11325     local fenceindent
11326
11327     local function has_trail(indent_table)
11328         return indent_table ~= nil and
11329             indent_table.trail ~= nil and
11330             next(indent_table.trail) ~= nil
11331     end
11332
11333     local function has_indents(indent_table)
11334         return indent_table ~= nil and
11335             indent_table.indents ~= nil and
11336             next(indent_table.indents) ~= nil
11337     end
11338
11339     local function get_last_indent_name(indent_table)
11340         if has_indents(indent_table) then
11341             return indent_table.indents[#indent_table.indents].name
11342         end
11343     end
11344
11345     local count_fenced_start_indent =
11346         function(_, _, indent_table, trail)
11347             local last_indent_name = get_last_indent_name(indent_table)
11348             fenceindent = 0

```

```

11349         if last_indent_name ~= "li" then
11350             fenceindent = #trail
11351         end
11352         return true
11353     end
11354
11355     local fencehead = function(char, infostring)
11356         return Cmt( Cb("indent_info")
11357             * parsers.check_trail, count_fenced_start_indent)
11358             * Cg(char^3, "fencelength")
11359             * parsers.optionalspace
11360             * infostring
11361             * (parsers.newline + parsers.eof)
11362     end
11363
11364     local fencetail = function(char)
11365         return parsers.check_trail_no_rem
11366             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11367             * parsers.optionalspace * (parsers.newline + parsers.eof)
11368             + parsers.eof
11369     end
11370
11371     local process_fenced_line =
11372         function(s, i, -- luacheck: ignore s i
11373             indent_table, line_content, is_blank)
11374         local remainder = ""
11375         if has_trail(indent_table) then
11376             remainder = indent_table.trail.internal_remainder
11377         end
11378
11379         if is_blank
11380             and get_last_indent_name(indent_table) == "li" then
11381             remainder = ""
11382         end
11383
11384         local str = remainder .. line_content
11385         local index = 1
11386         local remaining = fenceindent
11387
11388         while true do
11389             local c = str:sub(index, index)
11390             if c == " " and remaining > 0 then
11391                 remaining = remaining - 1
11392                 index = index + 1
11393             elseif c == "\t" and remaining > 3 then
11394                 remaining = remaining - 4
11395                 index = index + 1

```

```

11396         else
11397             break
11398         end
11399     end
11400
11401     return true, str:sub(index)
11402 end
11403
11404 local fencedline = function(char)
11405     return Cmt( Cb("indent_info")
11406         * C(parsers.line - fencetail(char))
11407         * Cc(false), process_fenced_line)
11408 end
11409
11410 local blankfencedline
11411     = Cmt( Cb("indent_info")
11412         * C(parsers.blankline)
11413         * Cc(true), process_fenced_line)
11414
11415 local TildeFencedCode
11416     = fencehead(parsers.tilde, tilde_infostring)
11417     * Cs(( parsers.check_minimal_blank_indent / ""
11418         * blankfencedline
11419         + ( parsers.check_minimal_indent / ""
11420         * fencedline(parsers.tilde))^0)
11421     * ( (parsers.check_minimal_indent / ""
11422         * fencetail(parsers.tilde) + parsers.succeed)
11423
11424 local BacktickFencedCode
11425     = fencehead(parsers.backtick, backtick_infostring)
11426     * Cs(( (parsers.check_minimal_blank_indent / ""
11427         * blankfencedline
11428         + (parsers.check_minimal_indent / ""
11429         * fencedline(parsers.backtick))^0)
11430     * ( (parsers.check_minimal_indent / ""
11431         * fencetail(parsers.backtick) + parsers.succeed)
11432
11433 local infostring_with_attributes
11434     = Ct(C((parsers.linechar
11435         - ( parsers.optionalspace
11436         * parsers.attributes))^0)
11437         * parsers.optionalspace
11438         * Ct(parsers.attributes))
11439
11440 local FencedCode
11441     = ((TildeFencedCode + BacktickFencedCode)
11442     / function(infostring, code)

```

```

11443         local expanded_code = self.expandtabs(code)
11444
11445         if allow_raw_blocks then
11446             local raw_attr = lpeg.match(parsers.raw_attribute,
11447   infostring)
11448             if raw_attr then
11449                 return writer.rawBlock(expanded_code, raw_attr)
11450             end
11451         end
11452
11453         local attr = nil
11454         if allow_attributes then
11455             local match = lpeg.match(infostring_with_attributes,
11456                                     infostring)
11457             if match then
11458                 infostring, attr = table.unpack(match)
11459             end
11460         end
11461         return writer.fencedCode(expanded_code, infostring, attr)
11462     end)
11463
11464     self.insert_pattern("Block after Verbatim",
11465                       FencedCode, "FencedCode")
11466
11467     local fencestart
11468     if blank_before_code_fence then
11469         fencestart = parsers.fail
11470     else
11471         fencestart = fencehead(parsers.backtick, backtick_infostring)
11472                       + fencehead(parsers.tilde, tilde_infostring)
11473     end
11474
11475     self.update_rule("EndlineExceptions", function(previous_pattern)
11476         if previous_pattern == nil then
11477             previous_pattern = parsers.EndlineExceptions
11478         end
11479         return previous_pattern + fencestart
11480     end)
11481
11482     self.add_special_character("`")
11483     self.add_special_character("~")
11484 end
11485 }
11486 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.

```
11487 M.extensions.fenced_divs = function(blank_before_div_fence)
11488   return {
11489     name = "built-in fenced_divs syntax extension",
11490     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.

```
11491     function self.div_begin(attributes)
11492       local start_output
11493       = {"\\markdownRendererFencedDivAttributeContextBegin\n",
11494         self.attributes(attributes)}
11495       local end_output
11496       = {"\\markdownRendererFencedDivAttributeContextEnd{}}
11497       return self.push_attributes(
11498         "div", attributes, start_output, end_output)
11499     end
```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```
11500     function self.div_end()
11501       return self.pop_attributes("div")
11502     end
11503   end, extend_reader = function(self)
11504     local parsers = self.parsers
11505     local writer = self.writer
```

Define basic patterns for matching the opening and the closing tag of a div.

```
11506     local fenced_div_infostring
11507       = C((parsers.linechar
11508         - ( parsers.spacechar^1
11509           * parsers.colon^1))^1)
11510
11511     local fenced_div_begin = parsers.nonindentspace
11512       * parsers.colon^3
11513       * parsers.optionalspace
11514       * fenced_div_infostring
11515       * ( parsers.spacechar^1
11516         * parsers.colon^1)^0
11517       * parsers.optionalspace
11518       * (parsers.newline + parsers.eof)
11519
11520     local fenced_div_end = parsers.nonindentspace
11521       * parsers.colon^3
```

```

11522             * parsers.optionalspace
11523             * (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.

```

11524     self.initialize_named_group("fenced_div_level", "0")
11525     self.initialize_named_group("fenced_div_num_opening_indents")
11526
11527     local function increment_div_level()
11528         local push_indent_table =
11529             function(s, i, indent_table, -- luacheck: ignore s i
11530                 fenced_div_num_opening_indents, fenced_div_level)
11531             fenced_div_level = tonumber(fenced_div_level) + 1
11532             local num_opening_indents = 0
11533             if indent_table.indents ~= nil then
11534                 num_opening_indents = #indent_table.indents
11535             end
11536             fenced_div_num_opening_indents[fenced_div_level]
11537                 = num_opening_indents
11538             return true, fenced_div_num_opening_indents
11539         end
11540
11541         local increment_level =
11542             function(s, i, fenced_div_level) -- luacheck: ignore s i
11543             fenced_div_level = tonumber(fenced_div_level) + 1
11544             return true, tostring(fenced_div_level)
11545         end
11546
11547         return Cg( Cmt( Cb("indent_info")
11548             * Cb("fenced_div_num_opening_indents")
11549             * Cb("fenced_div_level"), push_indent_table)
11550             , "fenced_div_num_opening_indents")
11551             * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11552             , "fenced_div_level")
11553     end
11554
11555     local function decrement_div_level()
11556         local pop_indent_table =
11557             function(s, i, -- luacheck: ignore s i
11558                 fenced_div_indent_table, fenced_div_level)
11559             fenced_div_level = tonumber(fenced_div_level)
11560             fenced_div_indent_table[fenced_div_level] = nil
11561             return true, tostring(fenced_div_level - 1)

```

```

11562         end
11563
11564         return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11565             * Cb("fenced_div_level"), pop_indent_table)
11566             , "fenced_div_level")
11567     end
11568
11569
11570     local non_fenced_div_block
11571         = parsers.check_minimal_indent * V("Block")
11572         - parsers.check_minimal_indent_and_trail * fenced_div_end
11573
11574     local non_fenced_div_paragraph
11575         = parsers.check_minimal_indent * V("Paragraph")
11576         - parsers.check_minimal_indent_and_trail * fenced_div_end
11577
11578     local blank = parsers.minimally_indented_blank
11579
11580     local block_separated = parsers.block_sep_group(blank)
11581         * non_fenced_div_block
11582
11583     local loop_body_pair
11584         = parsers.create_loop_body_pair(block_separated,
11585             non_fenced_div_paragraph,
11586             parsers.block_sep_group(blank),
11587             parsers.par_sep_group(blank))
11588
11589     local content_loop = ( non_fenced_div_block
11590         * loop_body_pair.block^0
11591         + non_fenced_div_paragraph
11592         * block_separated
11593         * loop_body_pair.block^0
11594         + non_fenced_div_paragraph
11595         * loop_body_pair.par^0)
11596         * blank^0
11597
11598     local FencedDiv = fenced_div_begin
11599         / function (infostring)
11600             local attr
11601                 = lpeg.match(Ct(parsers.attributes),
11602                     infostring)
11603             if attr == nil then
11604                 attr = {"." .. infostring}
11605             end
11606             return attr
11607         end
11608         / writer.div_begin

```

```

11609         * increment_div_level()
11610         * parsers.skipblanklines
11611         * Ct(content_loop)
11612         * parsers.minimally_indented_blank^0
11613         * parsers.check_minimal_indent_and_trail
11614         * fenced_div_end
11615         * decrement_div_level()
11616         * (Cc("") / writer.div_end)
11617
11618     self.insert_pattern("Block after Verbatim",
11619                       FencedDiv, "FencedDiv")
11620
11621     self.add_special_character(":")
11622

```

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.

```

11623     local function is_inside_div()
11624         local check_div_level =
11625             function(s, i, fenced_div_level) -- luacheck: ignore s i
11626                 fenced_div_level = tonumber(fenced_div_level)
11627                 return fenced_div_level > 0
11628             end
11629
11630     return Cmt(Cb("fenced_div_level"), check_div_level)
11631 end
11632
11633 local function check_indent()
11634     local compare_indent =
11635         function(s, i, indent_table, -- luacheck: ignore s i
11636                fenced_div_num_opening_indents, fenced_div_level)
11637             fenced_div_level = tonumber(fenced_div_level)
11638             local num_current_indents
11639                 = ( indent_table.current_line_indents ~= nil and
11640                   #indent_table.current_line_indents) or 0
11641             local num_opening_indents
11642                 = fenced_div_num_opening_indents[fenced_div_level]
11643             return num_current_indents == num_opening_indents
11644         end
11645
11646     return Cmt( Cb("indent_info")
11647               * Cb("fenced_div_num_opening_indents")
11648               * Cb("fenced_div_level"), compare_indent)
11649 end
11650
11651 local fencestart = is_inside_div()

```

```

11652             * fenced_div_end
11653             * check_indent()
11654
11655     if not blank_before_div_fence then
11656       self.update_rule("EndlineExceptions", function(previous_pattern)
11657         if previous_pattern == nil then
11658           previous_pattern = parsers.EndlineExceptions
11659         end
11660         return previous_pattern + fencestart
11661       end)
11662     end
11663   end
11664 }
11665 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11666 M.extensions.header_attributes = function()
11667   return {
11668     name = "built-in header_attributes syntax extension",
11669     extend_writer = function()
11670     end, extend_reader = function(self)
11671       local parsers = self.parsers
11672       local writer = self.writer
11673
11674       local function strip_atx_end(s)
11675         return s:gsub("%s+##%s*$", "")
11676       end
11677
11678       local AtxHeading = Cg(parsers.heading_start, "level")
11679         * parsers.optionalspace
11680         * (C(((parsers.linechar
11681           - (parsers.attributes
11682             * parsers.optionalspace
11683             * parsers.newline))
11684           * (parsers.linechar
11685             - parsers.lbrace)^0)^1)
11686         / strip_atx_end
11687         / parsers.parse_heading_text)
11688       * Cg(Ct(parsers.newline
11689         + (parsers.attributes
11690           * parsers.optionalspace
11691           * parsers.newline)), "attributes")
11692       * Cb("level")
11693       * Cb("attributes")

```

```

11694         / writer.heading
11695
11696     local function strip_trailing_spaces(s)
11697         return s:gsub("%s*$","")
11698     end
11699
11700     local heading_line = (parsers.linechar
11701                          - (parsers.attributes
11702                             * parsers.optionalspace
11703                             * parsers.newline))^1
11704                          - parsers.thematic_break_lines
11705
11706     local heading_text
11707         = heading_line
11708         * ( (V("Endline") / "\n")
11709            * (heading_line - parsers.heading_level))^0
11710         * parsers.newline^-1
11711
11712     local SettextHeading
11713         = parsers.freeze_trail * parsers.check_trail_no_rem
11714         * #(heading_text
11715            * (parsers.attributes
11716               * parsers.optionalspace
11717               * parsers.newline)^-1
11718            * parsers.check_minimal_indent
11719            * parsers.check_trail
11720            * parsers.heading_level)
11721         * Cs(heading_text) / strip_trailing_spaces
11722         / parsers.parse_heading_text
11723         * Cg(Ct((parsers.attributes
11724                * parsers.optionalspace
11725                * parsers.newline)^-1), "attributes")
11726         * parsers.check_minimal_indent_and_trail * parsers.heading_level
11727         * Cb("attributes")
11728         * parsers.newline
11729         * parsers.unfreeze_trail
11730         / writer.heading
11731
11732     local Heading = AtxHeading + SettextHeading
11733     self.update_rule("Heading", Heading)
11734 end
11735 }
11736 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc in-line code attribute syntax extension.

```
11737 M.extensions.inline_code_attributes = function()
11738   return {
11739     name = "built-in inline_code_attributes syntax extension",
11740     extend_writer = function()
11741     end, extend_reader = function(self)
11742       local writer = self.writer
11743
11744       local CodeWithAttributes = parsers.inticks
11745         * Ct(parsers.attributes)
11746         / writer.code
11747
11748       self.insert_pattern("Inline before Code",
11749         CodeWithAttributes,
11750         "CodeWithAttributes")
11751     end
11752   }
11753 end
```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```
11754 M.extensions.line_blocks = function()
11755   return {
11756     name = "built-in line_blocks syntax extension",
11757     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.

```
11758       function self.lineblock(lines)
11759         if not self.is_writing then return "" end
11760         local buffer = {}
11761         for i = 1, #lines - 1 do
11762           buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11763         end
11764         buffer[#buffer + 1] = lines[#lines]
11765
11766         return {"\\markdownRendererLineBlockBegin\n"
11767           ,buffer,
11768           "\n\\markdownRendererLineBlockEnd "}
11769       end
11770     end, extend_reader = function(self)
11771       local parsers = self.parsers
11772       local writer = self.writer
11773
```

```

11774     local LineBlock
11775         = Ct((Cs(( (parsers.pipe * parsers.space) / ""
11776                 * ((parsers.space)/entities.char_entity("nbsp"))^0
11777                 * parsers.linechar^0 * (parsers.newline/""))
11778                 * (-parsers.pipe
11779                 * (parsers.space^1/" ")
11780                 * parsers.linechar^1
11781                 * (parsers.newline/"")
11782                 )^0
11783                 * (parsers.blankline/"")^0)
11784             / self.parser_functions.parse_inlines)^1)
11785     / writer.lineblock
11786
11787     self.insert_pattern("Block after Blockquote",
11788                       LineBlock, "LineBlock")
11789 end
11790 }
11791 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11792 M.extensions.mark = function()
11793     return {
11794         name = "built-in mark syntax extension",
11795         extend_writer = function(self)

```

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

```

11796         function self.mark(s)
11797             if self.flatten_inlines then return s end
11798             return {"\\markdownRendererMark{" , s, "}"}
11799         end
11800     end, extend_reader = function(self)
11801         local parsers = self.parsers
11802         local writer = self.writer
11803
11804         local doubleequals = P("==")
11805
11806         local Mark
11807             = parsers.between(V("Inline"), doubleequals, doubleequals)
11808             / function (inlines) return writer.mark(inlines) end
11809
11810         self.add_special_character("=")
11811         self.insert_pattern("Inline before LinkAndEmph",
11812                           Mark, "Mark")
11813     end
11814 }

```



11815 end

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11816 M.extensions.link_attributes = function()
11817   return {
11818     name = "built-in link_attributes syntax extension",
11819     extend_writer = function()
11820     end, extend_reader = function(self)
11821       local parsers = self.parsers
11822       local options = self.options
11823
```

The following patterns define link reference definitions with attributes.

```
11824     local define_reference_parser
11825       = (parsers.check_trail / "")
11826       * parsers.link_label
11827       * parsers.colon
11828       * parsers.spnlc * parsers.url
11829       * ( parsers.spnlc_sep * parsers.title
11830         * (parsers.spnlc * Ct(parsers.attributes))
11831         * parsers.only_blank
11832         + parsers.spnlc_sep * parsers.title * parsers.only_blank
11833         + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11834         * parsers.only_blank
11835         + Cc("") * parsers.only_blank)
11836
11837     local ReferenceWithAttributes = define_reference_parser
11838                                   / self.register_link
11839
11840     self.update_rule("Reference", ReferenceWithAttributes)
11841
```

The following patterns define direct and indirect links with attributes.

```
11842
11843     local LinkWithAttributesAndEmph
11844       = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11845         "match_link_attributes"))
11846       / self.defer_link_and_emphasis_processing
11847
11848     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11849
```

The following patterns define autolinks with attributes.

```
11850     local AutoLinkUrlWithAttributes
11851       = parsers.auto_link_url
```

```

11852             * Ct(parsers.attributes)
11853             / self.auto_link_url
11854
11855     self.insert_pattern("Inline before AutoLinkUrl",
11856                       AutoLinkUrlWithAttributes,
11857                       "AutoLinkUrlWithAttributes")
11858
11859     local AutoLinkEmailWithAttributes
11860           = parsers.auto_link_email
11861           * Ct(parsers.attributes)
11862           / self.auto_link_email
11863
11864     self.insert_pattern("Inline before AutoLinkEmail",
11865                       AutoLinkEmailWithAttributes,
11866                       "AutoLinkEmailWithAttributes")
11867
11868     if options.relativeReferences then
11869
11870         local AutoLinkRelativeReferenceWithAttributes
11871               = parsers.auto_link_relative_reference
11872               * Ct(parsers.attributes)
11873               / self.auto_link_url
11874
11875         self.insert_pattern(
11876           "Inline before AutoLinkRelativeReference",
11877           AutoLinkRelativeReferenceWithAttributes,
11878           "AutoLinkRelativeReferenceWithAttributes")
11879
11880     end
11881
11882 end
11883 }
11884 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.

```

11885 M.extensions.notes = function(notes, inline_notes)
11886   assert(notes or inline_notes)
11887   return {
11888     name = "built-in notes syntax extension",
11889     extend_writer = function(self)

```

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

```
11890     function self.note(s)
11891         if self.flatten_inlines then return "" end
11892         return {"\\markdownRendererNote{" ,s,"}"}
11893     end
11894 end, extend_reader = function(self)
11895     local parsers = self.parsers
11896     local writer = self.writer
11897
11898     local rawnotes = parsers.rawnotes
11899
11900     if inline_notes then
11901         local InlineNote
11902             = parsers.circumflex
11903             * ( parsers.link_label
11904               / self.parser_functions.parse_inlines_no_inline_note)
11905             / writer.note
11906
11907         self.insert_pattern("Inline after LinkAndEmph",
11908                             InlineNote, "InlineNote")
11909     end
11910     if notes then
11911         local function strip_first_char(s)
11912             return s:sub(2)
11913         end
11914
11915         local RawNoteRef
11916             = #(parsers.lbracket * parsers.circumflex)
11917             * parsers.link_label / strip_first_char
11918
11919         -- like indirect_link
11920         local function lookup_note(ref)
11921             return writer.defer_call(function()
11922                 local found = rawnotes[self.normalize_tag(ref)]
11923                 if found then
11924                     return writer.note(
11925                         self.parser_functions.parse_blocks_nested(found))
11926                 else
11927                     return {"[",
11928                             self.parser_functions.parse_inlines("^" .. ref), "]" }
11929                 end
11930             end)
11931         end
11932
11933         local function register_note(ref,rawnote)
11934             local normalized_tag = self.normalize_tag(ref)
```

```

11935         if rawnotes[normalized_tag] == nil then
11936             rawnotes[normalized_tag] = rawnote
11937         end
11938         return ""
11939     end
11940
11941     local NoteRef = RawNoteRef / lookup_note
11942
11943     local optionally_indented_line
11944         = parsers.check_optional_indent_and_any_trail * parsers.line
11945
11946     local blank
11947         = parsers.check_optional_blank_indent_and_any_trail
11948         * parsers.optionalspace * parsers.newline
11949
11950     local chunk
11951         = Cs(parsers.line
11952             * (optionally_indented_line - blank)^0)
11953
11954     local indented_blocks = function(bl)
11955         return Cs( bl
11956             * ( blank^1 * (parsers.check_optional_indent / "")
11957             * parsers.check_code_trail
11958             * -parsers.blankline * bl)^0)
11959     end
11960
11961     local NoteBlock
11962         = parsers.check_trail_no_rem
11963         * RawNoteRef * parsers.colon
11964         * parsers.spnlc * indented_blocks(chunk)
11965         / register_note
11966
11967     self.update_rule("Reference", function(previous_pattern)
11968         if previous_pattern == nil then
11969             previous_pattern = parsers.Reference
11970         end
11971         return NoteBlock + previous_pattern
11972     end)
11973
11974     self.insert_pattern("Inline before LinkAndEmph",
11975                         NoteRef, "NoteRef")
11976 end
11977
11978 self.add_special_character("^")
11979 end
11980 }
11981 end

```

### 3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```
11982 M.extensions.pipe_tables = function(table_captions, table_attributes)
11983
11984   local function make_pipe_table_rectangular(rows)
11985     local num_columns = #rows[2]
11986     local rectangular_rows = {}
11987     for i = 1, #rows do
11988       local row = rows[i]
11989       local rectangular_row = {}
11990       for j = 1, num_columns do
11991         rectangular_row[j] = row[j] or ""
11992       end
11993       table.insert(rectangular_rows, rectangular_row)
11994     end
11995     return rectangular_rows
11996   end
11997
11998   local function pipe_table_row(allow_empty_first_column
11999                               , nonempty_column
12000                               , column_separator
12001                               , column)
12002     local row_beginning
12003     if allow_empty_first_column then
12004       row_beginning = -- empty first column
12005                       #(parsers.spacechar^4
12006                       * column_separator)
12007                       * parsers.optionalspace
12008                       * column
12009                       * parsers.optionalspace
12010                       -- non-empty first column
12011                       + parsers.nonindentSPACE
12012                       * nonempty_column^-1
12013                       * parsers.optionalspace
12014     else
12015       row_beginning = parsers.nonindentSPACE
12016                       * nonempty_column^-1
12017                       * parsers.optionalspace
12018     end
12019
12020   return Ct(row_beginning
```

```

12021         * (-- single column with no leading pipes
12022           #(column_separator
12023             * parsers.optionalspace
12024             * parsers.newline)
12025         * column_separator
12026         * parsers.optionalspace
12027         -- single column with leading pipes or
12028         -- more than a single column
12029         + (column_separator
12030           * parsers.optionalspace
12031           * column
12032           * parsers.optionalspace)^1
12033         * (column_separator
12034           * parsers.optionalspace)^-1))
12035     end
12036
12037     return {
12038       name = "built-in pipe_tables syntax extension",
12039       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.

```

12040     function self.table(rows, caption, attributes)
12041       if not self.is_writing then return "" end
12042       local buffer = {}
12043       if attributes ~= nil then
12044         table.insert(buffer,
12045           "\\markdownRendererTableAttributeContextBegin\n")
12046         table.insert(buffer, self.attributes(attributes))
12047       end
12048       table.insert(buffer,
12049         {"\\markdownRendererTable{",
12050           caption or "", "}{" , #rows - 1, "}{" ,
12051           #rows[1], "}"}))
12052       local temp = rows[2] -- put alignments on the first row
12053       rows[2] = rows[1]
12054       rows[1] = temp
12055       for i, row in ipairs(rows) do
12056         table.insert(buffer, "{")
12057         for _, column in ipairs(row) do
12058           if i > 1 then -- do not use braces for alignments
12059             table.insert(buffer, "{")
12060           end
12061           table.insert(buffer, column)
12062           if i > 1 then
12063             table.insert(buffer, "}")
12064           end

```

```

12065         end
12066         table.insert(buffer, "}")
12067     end
12068     if attributes ~= nil then
12069         table.insert(buffer,
12070             "\\markdownRendererTableAttributeContextEnd{")
12071     end
12072     return buffer
12073 end
12074 end, extend_reader = function(self)
12075     local parsers = self.parsers
12076     local writer = self.writer
12077
12078     local table_hline_separator = parsers.pipe + parsers.plus
12079
12080     local table_hline_column = (parsers.dash
12081         - #(parsers.dash
12082             * (parsers.spacechar
12083                 + table_hline_separator
12084                 + parsers.newline)))^1
12085     * (parsers.colon * Cc("r")
12086         + parsers.dash * Cc("d"))
12087     + parsers.colon
12088     * (parsers.dash
12089         - #(parsers.dash
12090             * (parsers.spacechar
12091                 + table_hline_separator
12092                 + parsers.newline)))^1
12093     * (parsers.colon * Cc("c")
12094         + parsers.dash * Cc("l"))
12095
12096     local table_hline = pipe_table_row(false
12097         , table_hline_column
12098         , table_hline_separator
12099         , table_hline_column)
12100
12101     local table_caption_beginning
12102     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
12103         * parsers.optionalspace * parsers.newline)^0
12104     * parsers.check_minimal_indent_and_trail
12105     * (P("Table")^-1 * parsers.colon)
12106     * parsers.optionalspace
12107
12108     local function strip_trailing_spaces(s)
12109         return s:gsub("%s*$", "")
12110     end
12111

```

```

12112     local table_row
12113         = pipe_table_row(true
12114             , (C((parsers.linechar - parsers.pipe)^1)
12115               / strip_trailing_spaces
12116               / self.parser_functions.parse_inlines)
12117             , parsers.pipe
12118             , (C((parsers.linechar - parsers.pipe)^0)
12119               / strip_trailing_spaces
12120               / self.parser_functions.parse_inlines))
12121
12122     local table_caption
12123     if table_captions then
12124         table_caption = #table_caption_beginning
12125             * table_caption_beginning
12126         if table_attributes then
12127             table_caption = table_caption
12128                 * (C(((( parsers.linechar
12129                     - (parsers.attributes
12130                       * parsers.optionalspace
12131                       * parsers.newline
12132                       * -( parsers.optionalspace
12133                         * parsers.linechar)))
12134                   + ( parsers.newline
12135                     * #( parsers.optionalspace
12136                       * parsers.linechar)
12137                     * C(parsers.optionalspace)
12138                       / writer.space))
12139                   * (parsers.linechar
12140                     - parsers.lbrace)^0)^1)
12141                 / self.parser_functions.parse_inlines)
12142             * (parsers.newline
12143               + ( Ct(parsers.attributes)
12144                 * parsers.optionalspace
12145                 * parsers.newline))
12146         else
12147             table_caption = table_caption
12148                 * C(( parsers.linechar
12149                   + ( parsers.newline
12150                     * #( parsers.optionalspace
12151                       * parsers.linechar)
12152                     * C(parsers.optionalspace)
12153                       / writer.space))^1)
12154                 / self.parser_functions.parse_inlines
12155                 * parsers.newline
12156         end
12157     else
12158         table_caption = parsers.fail

```



```

12159     end
12160
12161     local PipeTable
12162     = Ct( table_row * parsers.newline
12163         * (parsers.check_minimal_indent_and_trail / {})
12164         * table_hline * parsers.newline
12165         * ( (parsers.check_minimal_indent / {})
12166           * table_row * parsers.newline)^0)
12167     / make_pipe_table_rectangular
12168     * table_caption^-1
12169     / writer.table
12170
12171     self.insert_pattern("Block after Blockquote",
12172                       PipeTable, "PipeTable")
12173   end
12174 }
12175 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

12176 M.extensions.raw_inline = function()
12177   return {
12178     name = "built-in raw_inline syntax extension",
12179     extend_writer = function(self)
12180       local options = self.options
12181

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

12182     function self.rawInline(s, attr)
12183       if not self.is_writing then return "" end
12184       if self.flatten_inlines then return s end
12185       local name = util.cache_verbatim(options.cacheDir, s)
12186       return {"\\markdownRendererInputRawInline{" ,
12187             name,"}{" , self.string(attr),"}" }
12188     end
12189   end, extend_reader = function(self)
12190     local writer = self.writer
12191
12192     local RawInline = parsers.inticks
12193                       * parsers.raw_attribute
12194                       / writer.rawInline
12195
12196     self.insert_pattern("Inline before Code",
12197                       RawInline, "RawInline")
12198   end

```

```
12199 }
12200 end
```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
12201 M.extensions.strike_through = function()
12202   return {
12203     name = "built-in strike_through syntax extension",
12204     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.

```
12205       function self.strike_through(s)
12206         if self.flatten_inlines then return s end
12207         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
12208       end
12209     end, extend_reader = function(self)
12210       local parsers = self.parsers
12211       local writer = self.writer
12212
12213       local StrikeThrough = (
12214         parsers.between(parsers.Inline, parsers.doubletildes,
12215           parsers.doubletildes)
12216       ) / writer.strike_through
12217
12218       self.insert_pattern("Inline after LinkAndEmph",
12219         StrikeThrough, "StrikeThrough")
12220
12221       self.add_special_character("~")
12222     end
12223   }
12224 end
```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
12225 M.extensions.subscripts = function()
12226   return {
12227     name = "built-in subscripts syntax extension",
12228     extend_writer = function(self)
```

Define `writer->subscript` as a function that will transform a subscript span `s` of input text to the output format.

```
12229       function self.subscript(s)
12230         if self.flatten_inlines then return s end
```

```

12231     return {"\\markdownRendererSubscript{" ,s,"}"}
12232   end
12233 end, extend_reader = function(self)
12234   local parsers = self.parsers
12235   local writer = self.writer
12236
12237   local Subscript = (
12238     parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12239   ) / writer.subscript
12240
12241   self.insert_pattern("Inline after LinkAndEmph",
12242     Subscript, "Subscript")
12243
12244   self.add_special_character("~")
12245 end
12246 }
12247 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

12248 M.extensions.superscripts = function()
12249   return {
12250     name = "built-in superscripts syntax extension",
12251     extend_writer = function(self)

```

Define `writer->superscript` as a function that will transform a superscript span `s` of input text to the output format.

```

12252       function self.superscript(s)
12253         if self.flatten_inlines then return s end
12254         return {"\\markdownRendererSuperscript{" ,s,"}"}
12255       end
12256     end, extend_reader = function(self)
12257       local parsers = self.parsers
12258       local writer = self.writer
12259
12260       local Superscript = (
12261         parsers.between(parsers.Str, parsers.circumflex,
12262           parsers.circumflex)
12263       ) / writer.superscript
12264
12265       self.insert_pattern("Inline after LinkAndEmph",
12266         Superscript, "Superscript")
12267
12268       self.add_special_character("^")
12269     end
12270   }

```

12271 end

### 3.1.7.19 T<sub>E</sub>X Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12272 M.extensions.tex_math = function(tex_math_dollars,  
12273                                     tex_math_single_backslash,  
12274                                     tex_math_double_backslash)  
12275   return {  
12276     name = "built-in tex_math syntax extension",  
12277     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.

```
12278       function self.display_math(s)  
12279         if self.flatten_inlines then return s end  
12280         return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}  
12281       end
```

Define `writer->inline_math` as a function that will transform a math span `s` of input text to the output format.

```
12282       function self.inline_math(s)  
12283         if self.flatten_inlines then return s end  
12284         return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}  
12285       end  
12286     end, extend_reader = function(self)  
12287       local parsers = self.parsers  
12288       local writer = self.writer  
12289  
12290       local function between(p, starter, ender)  
12291         return (starter * Cs(p * (p - ender)^0) * ender)  
12292       end  
12293  
12294       local function strip_preceding_whitespaces(str)  
12295         return str:gsub("^%s*(.-)$", "%1")  
12296       end  
12297  
12298       local allowed_before_closing  
12299         = B( parsers.backslash * parsers.any  
12300           + parsers.any * (parsers.any - parsers.backslash))  
12301  
12302       local allowed_before_closing_no_space  
12303         = B( parsers.backslash * parsers.any  
12304           + parsers.any * (parsers.nonspacechar - parsers.backslash))  
12305     end
```

The following patterns implement the Pandoc dollar math syntax extension.

```

12306     local dollar_math_content
12307         = (parsers.newline * (parsers.check_optional_indent / ""))
12308           + parsers.backslash^-1
12309           * parsers.linechar)
12310         - parsers.blankline^2
12311         - parsers.dollar
12312
12313     local inline_math_opening_dollars = parsers.dollar
12314   * #(parsers.nonspacechar)
12315
12316     local inline_math_closing_dollars
12317         = allowed_before_closing_no_space
12318         * parsers.dollar
12319         * -#(parsers.digit)
12320
12321     local inline_math_dollars = between(Cs( dollar_math_content),
12322   inline_math_opening_dollars,
12323   inline_math_closing_dollars)
12324
12325     local display_math_opening_dollars = parsers.dollar
12326   * parsers.dollar
12327
12328     local display_math_closing_dollars = parsers.dollar
12329   * parsers.dollar
12330
12331     local display_math_dollars = between(Cs( dollar_math_content),
12332   display_math_opening_dollars,
12333   display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12334     local backslash_math_content
12335         = (parsers.newline * (parsers.check_optional_indent / ""))
12336           + parsers.linechar)
12337         - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12338     local inline_math_opening_double = parsers.backslash
12339   * parsers.backslash
12340   * parsers.lparent
12341
12342     local inline_math_closing_double = allowed_before_closing
12343   * parsers.spacechar^0
12344   * parsers.backslash
12345   * parsers.backslash
12346   * parsers.rparent
12347

```

```

12348     local inline_math_double = between(Cs( backslash_math_content),
12349                                     inline_math_opening_double,
12350                                     inline_math_closing_double)
12351                                     / strip_preceding_whitespaces
12352
12353     local display_math_opening_double = parsers.backslash
12354                                     * parsers.backslash
12355                                     * parsers.lbracket
12356
12357     local display_math_closing_double = allowed_before_closing
12358                                     * parsers.spacechar^0
12359                                     * parsers.backslash
12360                                     * parsers.backslash
12361                                     * parsers.rbracket
12362
12363     local display_math_double = between(Cs( backslash_math_content),
12364                                     display_math_opening_double,
12365                                     display_math_closing_double)
12366                                     / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

12367     local inline_math_opening_single = parsers.backslash
12368                                     * parsers.lparent
12369
12370     local inline_math_closing_single = allowed_before_closing
12371                                     * parsers.spacechar^0
12372                                     * parsers.backslash
12373                                     * parsers.rparent
12374
12375     local inline_math_single = between(Cs( backslash_math_content),
12376                                     inline_math_opening_single,
12377                                     inline_math_closing_single)
12378                                     / strip_preceding_whitespaces
12379
12380     local display_math_opening_single = parsers.backslash
12381                                     * parsers.lbracket
12382
12383     local display_math_closing_single = allowed_before_closing
12384                                     * parsers.spacechar^0
12385                                     * parsers.backslash
12386                                     * parsers.rbracket
12387
12388     local display_math_single = between(Cs( backslash_math_content),
12389                                     display_math_opening_single,
12390                                     display_math_closing_single)
12391                                     / strip_preceding_whitespaces
12392
12393     local display_math = parsers.fail

```

```

12394
12395     local inline_math = parsers.fail
12396
12397     if tex_math_dollars then
12398         display_math = display_math + display_math_dollars
12399         inline_math = inline_math + inline_math_dollars
12400     end
12401
12402     if tex_math_double_backslash then
12403         display_math = display_math + display_math_double
12404         inline_math = inline_math + inline_math_double
12405     end
12406
12407     if tex_math_single_backslash then
12408         display_math = display_math + display_math_single
12409         inline_math = inline_math + inline_math_single
12410     end
12411
12412     local TexMath = display_math / writer.display_math
12413                   + inline_math / writer.inline_math
12414
12415     self.insert_pattern("Inline after LinkAndEmph",
12416                       TexMath, "TexMath")
12417
12418     if tex_math_dollars then
12419         self.add_special_character("$")
12420     end
12421
12422     if tex_math_single_backslash or tex_math_double_backslash then
12423         self.add_special_character("\\")
12424         self.add_special_character("[")
12425         self.add_special_character("]")
12426         self.add_special_character("(")
12427         self.add_special_character("(")
12428     end
12429 end
12430 }
12431 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.

```
12432 M.extensions.jekyll_data = function(expect_jekyll_data,
12433   ensure_jekyll_data)
12434   return {
12435     name = "built-in jekyll_data syntax extension",
12436     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.

```
12437     function self.jekyllData(d, t, p)
12438       if not self.is_writing then return "" end
12439
12440       local buf = {}
12441
12442       local keys = {}
12443       for k, _ in pairs(d) do
12444         table.insert(keys, k)
12445       end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
12446         table.sort(keys, function(first, second)
12447           if type(first) ~= type(second) then
12448             return type(first) < type(second)
12449           else
12450             return first < second
12451           end
12452         end)
12453
12454         if not p then
12455           table.insert(buf, "\\markdownRendererJekyllDataBegin")
12456         end
12457
12458         local is_sequence = false
12459         if #d > 0 and #d == #keys then
12460           for i=1, #d do
12461             if d[i] == nil then
12462               goto not_a_sequence
12463             end
12464           end
12465           is_sequence = true
12466         end
```



```

12467     ::not_a_sequence::
12468
12469     if is_sequence then
12470         table.insert(buf,
12471             "\\markdownRendererJekyllDataSequenceBegin{")
12472         table.insert(buf, self.identifier(p or "null"))
12473         table.insert(buf, "}{"")
12474         table.insert(buf, #keys)
12475         table.insert(buf, "}")
12476     else
12477         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12478         table.insert(buf, self.identifier(p or "null"))
12479         table.insert(buf, "}{"")
12480         table.insert(buf, #keys)
12481         table.insert(buf, "}")
12482     end
12483
12484     for _, k in ipairs(keys) do
12485         local v = d[k]
12486         local typ = type(v)
12487         k = tostring(k or "null")
12488         if typ == "table" and next(v) ~= nil then
12489             table.insert(
12490                 buf,
12491                 self.jekyllData(v, t, k)
12492             )
12493         else
12494             k = self.identifier(k)
12495             v = tostring(v)
12496             if typ == "boolean" then
12497                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12498                 table.insert(buf, k)
12499                 table.insert(buf, "}{"")
12500                 table.insert(buf, v)
12501                 table.insert(buf, "}")
12502             elseif typ == "number" then
12503                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12504                 table.insert(buf, k)
12505                 table.insert(buf, "}{"")
12506                 table.insert(buf, v)
12507                 table.insert(buf, "}")
12508             elseif typ == "string" then
12509                 table.insert(buf,
12510                     "\\markdownRendererJekyllDataProgrammaticString{")
12511                 table.insert(buf, k)
12512                 table.insert(buf, "}{"")
12513                 table.insert(buf, self.identifier(v))

```

```

12514         table.insert(buf, "}")
12515         table.insert(buf,
12516             "\\markdownRendererJekyllDataTypographicString{")
12517         table.insert(buf, k)
12518         table.insert(buf, "}{")
12519         table.insert(buf, t(v))
12520         table.insert(buf, "}")
12521     elseif typ == "table" then
12522         table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12523         table.insert(buf, k)
12524         table.insert(buf, "}")
12525     else
12526         local error = self.error(format(
12527             "Unexpected type %s for value of "
12528             .. "YAML key %s.", typ, k))
12529         table.insert(buf, error)
12530     end
12531 end
12532 end
12533
12534 if is_sequence then
12535     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12536 else
12537     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12538 end
12539
12540 if not p then
12541     table.insert(buf, "\\markdownRendererJekyllDataEnd")
12542 end
12543
12544 return buf
12545 end
12546 end, extend_reader = function(self)
12547     local parsers = self.parsers
12548     local writer = self.writer
12549
12550     local JekyllData
12551     = Cmt( C((parsers.line - P("---") - P("..."))~0)
12552         , function(s, i, text) -- luacheck: ignore s i
12553             local data
12554             local ran_ok, _ = pcall(function()
12555                 local tinyyaml = require("tinyyaml")
12556                 data = tinyyaml.parse(text, {timestamps=false})
12557             end)
12558             if ran_ok and data ~= nil then
12559                 return true, writer.jekyllData(data, function(s)
12560                     return self.parser_functions.parse_blocks_nested(s)

```

```

12561         end, nil)
12562     else
12563         return false
12564     end
12565     end
12566 )
12567
12568 local UnexpectedJekyllData
12569 = P("----")
12570 * parsers.blankline / 0
12571 -- if followed by blank, it's thematic break
12572 * #(-parsers.blankline)
12573 * JekyllData
12574 * (P("----") + P("..."))
12575
12576 local ExpectedJekyllData
12577 = ( P("----")
12578     * parsers.blankline / 0
12579     -- if followed by blank, it's thematic break
12580     * #(-parsers.blankline)
12581     )^-1
12582 * JekyllData
12583 * (P("----") + P("..."))^-1
12584
12585 if ensure_jekyll_data then
12586     ExpectedJekyllData = ExpectedJekyllData
12587                         * parsers.eof
12588 else
12589     ExpectedJekyllData = ( ExpectedJekyllData
12590                           * (V("Blank")^0 / writer.interblocksep)
12591                           )^-1
12592 end
12593
12594 self.insert_pattern("Block before Blockquote",
12595                   UnexpectedJekyllData, "UnexpectedJekyllData")
12596 if expect_jekyll_data then
12597     self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12598 end
12599 end
12600 }
12601 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`.

```
12602 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```
12603   options = options or {}
```

```
12604   setmetatable(options, { __index = function (_, key)
```

```
12605     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.

```
12606   local parser_convert = nil
```

```
12607   return function(input)
```

```
12608     local function convert(input)
```

```
12609       if parser_convert == nil then
```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```
12610         local parser = require("markdown-parser")
```

```
12611         if metadata.version ~= parser.metadata.version then
```

```
12612           warn("markdown.lua " .. metadata.version .. " used with " ..
```

```
12613             "markdown-parser.lua " .. parser.metadata.version .. ".")
```

```
12614         end
```

```
12615         parser_convert = parser.new(options)
```

```
12616       end
```

```
12617       return parser_convert(input)
```

```
12618     end
```

If we cache markdown documents, produce the cache file and transform its filename to plain  $\text{T}_{\text{E}}\text{X}$  output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```
12619   local output
```

```
12620   if options.eagerCache or options.finalizeCache then
```

```
12621     local salt = util.salt(options)
```

```
12622     local name = util.cache(options.cacheDir, input, salt, convert,
```

```
12623       ".md.tex")
```

```
12624     output = [[\input{}}] .. name .. [[\relax]]
```

Otherwise, return the result of the conversion directly.

```
12625   else
```

```
12626     output = convert(input)
```

```
12627   end
```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

12628     if options.finalizeCache then
12629         local file, mode
12630         if options.frozenCacheCounter > 0 then
12631             mode = "a"
12632         else
12633             mode = "w"
12634         end
12635         file = assert(io.open(options.frozenCacheFileName, mode),
12636             [[Could not open file ]] .. options.frozenCacheFileName
12637             .. [[ for writing]])
12638         assert(file:write(
12639             [[\expandafter\global\expandafter\def\csname ]]
12640             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12641             .. [[\endcsname{}]] .. output .. [[]]] .. "\n"))
12642         assert(file:close())
12643     end
12644     return output
12645 end
12646 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain T<sub>E</sub>X output. See Section 2.1.1.

```
12647 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```

12648     options = options or {}
12649     setmetatable(options, { __index = function (_, key)
12650         return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12651     if options.singletonCache and singletonCache.convert then
12652         for k, v in pairs(defaultOptions) do
12653             if type(v) == "table" then
12654                 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12655                     if singletonCache.options[k][i] ~= options[k][i] then
12656                         goto miss
12657                     end
12658                 end

```

The `cacheDir` option is disregarded.

```

12659         elseif k ~= "cacheDir"
12660             and singletonCache.options[k] ~= options[k] then
12661             goto miss
12662         end
12663     end
12664     return singletonCache.convert
12665 end
12666 :::miss::

```

Apply built-in syntax extensions based on `options`.

```
12667   local extensions = {}
12668
12669   if options.bracketedSpans then
12670     local bracketed_spans_extension = M.extensions.bracketed_spans()
12671     table.insert(extensions, bracketed_spans_extension)
12672   end
12673
12674   if options.contentBlocks then
12675     local content_blocks_extension = M.extensions.content_blocks(
12676       options.contentBlocksLanguageMap)
12677     table.insert(extensions, content_blocks_extension)
12678   end
12679
12680   if options.definitionLists then
12681     local definition_lists_extension = M.extensions.definition_lists(
12682       options.tightLists)
12683     table.insert(extensions, definition_lists_extension)
12684   end
12685
12686   if options.fencedCode then
12687     local fenced_code_extension = M.extensions.fenced_code(
12688       options.blankBeforeCodeFence,
12689       options.fencedCodeAttributes,
12690       options.rawAttribute)
12691     table.insert(extensions, fenced_code_extension)
12692   end
12693
12694   if options.fencedDivs then
12695     local fenced_div_extension = M.extensions.fenced_divs(
12696       options.blankBeforeDivFence)
12697     table.insert(extensions, fenced_div_extension)
12698   end
12699
12700   if options.headerAttributes then
12701     local header_attributes_extension = M.extensions.header_attributes()
12702     table.insert(extensions, header_attributes_extension)
12703   end
12704
12705   if options.inlineCodeAttributes then
12706     local inline_code_attributes_extension =
12707       M.extensions.inline_code_attributes()
12708     table.insert(extensions, inline_code_attributes_extension)
12709   end
12710
12711   if options.jekyllData then
12712     local jekyll_data_extension = M.extensions.jekyll_data(
```

```

12713     options.expectJekyllData, options.ensureJekyllData)
12714     table.insert(extensions, jekyll_data_extension)
12715 end
12716
12717 if options.linkAttributes then
12718     local link_attributes_extension =
12719         M.extensions.link_attributes()
12720     table.insert(extensions, link_attributes_extension)
12721 end
12722
12723 if options.lineBlocks then
12724     local line_block_extension = M.extensions.line_blocks()
12725     table.insert(extensions, line_block_extension)
12726 end
12727
12728 if options.mark then
12729     local mark_extension = M.extensions.mark()
12730     table.insert(extensions, mark_extension)
12731 end
12732
12733 if options.pipeTables then
12734     local pipe_tables_extension = M.extensions.pipe_tables(
12735         options.tableCaptions, options.tableAttributes)
12736     table.insert(extensions, pipe_tables_extension)
12737 end
12738
12739 if options.rawAttribute then
12740     local raw_inline_extension = M.extensions.raw_inline()
12741     table.insert(extensions, raw_inline_extension)
12742 end
12743
12744 if options.strikeThrough then
12745     local strike_through_extension = M.extensions.strike_through()
12746     table.insert(extensions, strike_through_extension)
12747 end
12748
12749 if options.subscripts then
12750     local subscript_extension = M.extensions.subscripts()
12751     table.insert(extensions, subscript_extension)
12752 end
12753
12754 if options.superscripts then
12755     local superscript_extension = M.extensions.superscripts()
12756     table.insert(extensions, superscript_extension)
12757 end
12758
12759 if options.texMathDollars or

```

```

12760     options.texMathSingleBackslash or
12761     options.texMathDoubleBackslash then
12762     local tex_math_extension = M.extensions.tex_math(
12763         options.texMathDollars,
12764         options.texMathSingleBackslash,
12765         options.texMathDoubleBackslash)
12766     table.insert(extensions, tex_math_extension)
12767 end
12768
12769 if options.notes or options.inlineNotes then
12770     local notes_extension = M.extensions.notes(
12771         options.notes, options.inlineNotes)
12772     table.insert(extensions, notes_extension)
12773 end
12774
12775 if options.citations then
12776     local citations_extension
12777         = M.extensions.citations(options.citationNbsps)
12778     table.insert(extensions, citations_extension)
12779 end
12780
12781 if options.fancyLists then
12782     local fancy_lists_extension = M.extensions.fancy_lists()
12783     table.insert(extensions, fancy_lists_extension)
12784 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12785 for _, user_extension_filename in ipairs(options.extensions) do
12786     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

12787     local pathname = assert(kpse.find_file(filename),
12788         [[Could not locate user-defined syntax extension "]]
12789         .. filename)
12790     local input_file = assert(io.open(pathname, "r"),
12791         [[Could not open user-defined syntax extension "]]
12792         .. pathname .. [{" for reading}]]
12793     local input = assert(input_file:read("*a"))
12794     assert(input_file:close())
12795     local user_extension, err = load([[
12796         local sandbox = {}
12797         setmetatable(sandbox, {__index = _G})
12798         _ENV = sandbox
12799     ]] .. input)()
12800     assert(user_extension,
12801         [[Failed to compile user-defined syntax extension "]]
12802         .. pathname .. [{": }]] .. (err or [{"}]))

```



Then, validate the user-defined syntax extension.

```
12803     assert(user_extension.api_version ~= nil,
12804           [[User-defined syntax extension ]] .. pathname
12805           .. [[ " does not specify mandatory field "api_version" ]])
12806     assert(type(user_extension.api_version) == "number",
12807           [[User-defined syntax extension ]] .. pathname
12808           .. [[ " specifies field "api_version" of type "]]
12809           .. type(user_extension.api_version)
12810           .. [[ " but "number" was expected ]])
12811     assert(user_extension.api_version > 0
12812           and user_extension.api_version
12813           <= metadata.user_extension_api_version,
12814           [[User-defined syntax extension ]] .. pathname
12815           .. [[ " uses syntax extension API version "]]
12816           .. user_extension.api_version .. [[ but markdown.lua ]]
12817           .. metadata.version .. [[ uses API version ]]
12818           .. metadata.user_extension_api_version
12819           .. [[, which is incompatible ]])
12820
12821     assert(user_extension.grammar_version ~= nil,
12822           [[User-defined syntax extension ]] .. pathname
12823           .. [[ " does not specify mandatory field "grammar_version" ]])
12824     assert(type(user_extension.grammar_version) == "number",
12825           [[User-defined syntax extension ]] .. pathname
12826           .. [[ " specifies field "grammar_version" of type "]]
12827           .. type(user_extension.grammar_version)
12828           .. [[ " but "number" was expected ]])
12829     assert(user_extension.grammar_version == metadata.grammar_version,
12830           [[User-defined syntax extension ]] .. pathname
12831           .. [[ " uses grammar version "]]
12832           .. user_extension.grammar_version
12833           .. [[ but markdown.lua ]] .. metadata.version
12834           .. [[ uses grammar version ]] .. metadata.grammar_version
12835           .. [[, which is incompatible ]])
12836
12837     assert(user_extension.finalize_grammar ~= nil,
12838           [[User-defined syntax extension ]] .. pathname
12839           .. [[ " does not specify mandatory "finalize_grammar" field ]])
12840     assert(type(user_extension.finalize_grammar) == "function",
12841           [[User-defined syntax extension ]] .. pathname
12842           .. [[ " specifies field "finalize_grammar" of type "]]
12843           .. type(user_extension.finalize_grammar)
12844           .. [[ " 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.)

```
12845     local extension = {
```

```

12846     name = [[user-defined "]] .. pathname .. [{" syntax extension}],
12847     extend_reader = user_extension.finalize_grammar,
12848     extend_writer = function() end,
12849   }
12850   return extension
12851 end)(user_extension_filename)
12852 table.insert(extensions, user_extension)
12853 end

```

Produce a conversion function from markdown to plain TeX.

```

12854 local writer = M.writer.new(options)
12855 local reader = M.reader.new(writer, options)
12856 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`.

```

12857 collectgarbage("collect")

```

Update the singleton cache.

```

12858 if options.singletonCache then
12859   local singletonCacheOptions = {}
12860   for k, v in pairs(options) do
12861     singletonCacheOptions[k] = v
12862   end
12863   setmetatable(singletonCacheOptions,
12864     { __index = function (_, key)
12865       return defaultOptions[key] end })
12866   singletonCache.options = singletonCacheOptions
12867   singletonCache.convert = convert
12868 end

```

Return the conversion function from markdown to plain TeX.

```

12869 return convert
12870 end
12871 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.

```

12872
12873 local input
12874 if input_filename then
12875   local input_file = assert(io.open(input_filename, "r"),
12876     [[Could not open file "]] .. input_filename .. [{" for reading}])
12877   input = assert(input_file:read("*a"))
12878   assert(input_file:close())
12879 else

```

```

12880  input = assert(io.read("*a"))
12881 end
12882

```

First, ensure that the `options.cacheDir` directory exists.

```

12883 local lfs = require("lfs")
12884 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12885   assert(lfs.mkdir(options["cacheDir"]))
12886 end

```

If Kpathsea has not been loaded before or if LuaTeX has not yet been initialized, configure Kpathsea on top of loading it.

```

12887 local kpse
12888 (function()
12889   local should_initialize = package.loaded.kpse == nil
12890                               or tex.initialize ~= nil
12891   kpse = require("kpse")
12892   if should_initialize then
12893     kpse.set_program_name("luatex")
12894   end
12895 end)()
12896 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.

```

12897 if metadata.version ~= md.metadata.version then
12898   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12899       "markdown.lua " .. md.metadata.version .. ".")
12900 end
12901 local convert = md.new(options)
12902 local output = convert(input)
12903
12904 if output_filename then
12905   local output_file = assert(io.open(output_filename, "w"),
12906     [[Could not open file ]] .. output_filename .. [[ for writing]])
12907   assert(output_file:write(output))
12908   assert(output_file:close())
12909 else
12910   assert(io.write(output))
12911 end

```

Remove the `options.cacheDir` directory if it is empty.

```

12912 if options.cacheDir then
12913   lfs.rmdir(options.cacheDir)
12914 end

```

## 3.2 Plain T<sub>E</sub>X Implementation

The plain T<sub>E</sub>X implementation provides macros for the interfacing between T<sub>E</sub>X and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T<sub>E</sub>X exposed by the plain T<sub>E</sub>X interface (see Section 2.2).

### 3.2.1 Logging Facilities

```
12915 \ExplSyntaxOn
12916 \cs_if_free:NT
12917   \markdownInfo
12918   {
12919     \cs_new:Npn
12920       \markdownInfo #1
12921       {
12922         \msg_info:nne
12923           { markdown }
12924           { generic-message }
12925           { #1 }
12926       }
12927   }
12928 \cs_if_free:NT
12929   \markdownWarning
12930   {
12931     \cs_new:Npn
12932       \markdownWarning #1
12933       {
12934         \msg_warning:nne
12935           { markdown }
12936           { generic-message }
12937           { #1 }
12938       }
12939   }
12940 \cs_if_free:NT
12941   \markdownError
12942   {
12943     \cs_new:Npn
12944       \markdownError #1 #2
12945       {
12946         \msg_error:nnee
12947           { markdown }
12948           { generic-message-with-help-text }
12949           { #1 }
12950           { #2 }
12951       }
12952   }
```

```

12953 \msg_new:nnn
12954   { markdown }
12955   { generic-message }
12956   { #1 }
12957 \msg_new:nnnn
12958   { markdown }
12959   { generic-message-with-help-text }
12960   { #1 }
12961   { #2 }
12962 \cs_generate_variant:Nn
12963   \msg_info:nnn
12964   { nne }
12965 \cs_generate_variant:Nn
12966   \msg_warning:nnn
12967   { nne }
12968 \cs_generate_variant:Nn
12969   \msg_error:nnnn
12970   { nnee }
12971 \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.

```

12972 \ExplSyntaxOn
12973 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
12974 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
12975 \cs_new:Nn
12976   \@@_plain_tex_load_theme:nnn
12977   {
12978     \prop_get:NnNTF
12979       \g_@@_plain_tex_loaded_themes_linenos_prop
12980       { #1 }
12981     \l_tmpa_tl
12982     {
12983       \prop_get:NnN
12984         \g_@@_plain_tex_loaded_themes_versions_prop
12985         { #1 }
12986       \l_tmpb_tl
12987       \str_if_eq:nVTF
12988         { #2 }
12989         \l_tmpb_tl
12990       {
12991         \msg_warning:nnnVn
12992           { markdown }
12993         { repeatedly-loaded-plain-tex-theme }

```

```

12994         { #1 }
12995         \l_tmpa_tl
12996         { #2 }
12997     }
12998     {
12999         \msg_error:nnnnVV
13000         { markdown }
13001         { different-versions-of-plain-tex-theme }
13002         { #1 }
13003         { #2 }
13004         \l_tmpb_tl
13005         \l_tmpa_tl
13006     }
13007 }
13008 {
13009     \prop_gput:Nnx
13010     \g_@@_plain_tex_loaded_themes_linenos_prop
13011     { #1 }
13012     { \tex_the:D \tex_inputlineno:D } % noqa: W200
13013     \prop_gput:Nnn
13014     \g_@@_plain_tex_loaded_themes_versions_prop
13015     { #1 }
13016     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

13017     \prop_if_in:NnTF
13018     \g_@@_plain_tex_built_in_themes_prop
13019     { #1 }
13020     {
13021         \msg_info:nnnn
13022         { markdown }
13023         { loading-built-in-plain-tex-theme }
13024         { #1 }
13025         { #2 }
13026         \prop_item:Nn
13027         \g_@@_plain_tex_built_in_themes_prop
13028         { #1 }
13029     }
13030     {
13031         \msg_info:nnnn
13032         { markdown }
13033         { loading-plain-tex-theme }
13034         { #1 }
13035         { #2 }
13036         \file_input:n
13037         { markdown theme #3 }

```

```

13038     }
13039   }
13040 }
13041 \msg_new:nnn
13042 { markdown }
13043 { loading-plain-tex-theme }
13044 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
13045 \msg_new:nnn
13046 { markdown }
13047 { loading-built-in-plain-tex-theme }
13048 { Loading~version~#2~of~built-in-plain~TeX~Markdown~theme~#1 }
13049 \msg_new:nnn
13050 { markdown }
13051 { repeatedly-loaded-plain-tex-theme }
13052 {
13053   Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
13054   loaded~on~line~#2,~not~loading~it~again
13055 }
13056 \msg_new:nnn
13057 { markdown }
13058 { different-versions-of-plain-tex-theme }
13059 {
13060   Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
13061   but~version~#3~has~already~been~loaded~on~line~#4
13062 }
13063 \cs_generate_variant:Nn
13064 \prop_gput:Nnn
13065 { Nnx }
13066 \cs_gset_eq:NN
13067 \@@_load_theme:nnn
13068 \@@_plain_tex_load_theme:nnn
13069 \cs_generate_variant:Nn
13070 \@@_load_theme:nnn
13071 { VeV }
13072 \cs_generate_variant:Nn
13073 \msg_error:nnnnn
13074 { nnnnVV }
13075 \cs_generate_variant:Nn
13076 \msg_warning:nnnnn
13077 { 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.

```

13078 \cs_new:Npn
13079 \markdownLoadPlainTeXTheme
13080 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

13081 \tl_set:NV
13082   \l_tmpa_tl
13083   \g_@@_current_theme_tl
13084 \tl_reverse:N
13085   \l_tmpa_tl
13086 \tl_set:Ne
13087   \l_tmpb_tl
13088   {
13089     \tl_tail:V
13090     \l_tmpa_tl
13091   }
13092 \tl_reverse:N
13093   \l_tmpb_tl

```

Next, we munge the theme name.

```

13094 \str_set:NV
13095   \l_tmpa_str
13096   \l_tmpb_tl
13097 \str_replace_all:Nnn
13098   \l_tmpa_str
13099   { / }
13100   { _ }

```

Finally, we load the plain TeX theme.

```

13101 \@@_plain_tex_load_theme:VeV
13102   \l_tmpb_tl
13103   { \markdownThemeVersion }
13104   \l_tmpa_str
13105 }
13106 \cs_generate_variant:Nn
13107   \tl_set:Nn
13108   { Ne }
13109 \cs_generate_variant:Nn
13110   \@@_plain_tex_load_theme:nnn
13111   { VeV }

```

The `witiko/dot` theme nags users that they should use the name `witiko/diagrams@v1` instead.

```

13112 \prop_gput:Nnn
13113   \g_@@_plain_tex_built_in_themes_prop
13114   { witiko / dot }
13115   {
13116     \str_if_eq:enF
13117     { \markdownThemeVersion }
13118     { silent }
13119     {

```



```

13120     \markdownWarning
13121     {
13122         The~theme~name~"witiko/dot"~has~been~soft~deprecated.
13123         \iow_newline:
13124         Consider~changing~the~name~to~"witiko/diagrams@v1".
13125     }
13126 }

```

We enable the `fencedCode` Lua option.

```
13127 \markdownSetup { fencedCode }
```

We store the previous definition of the fenced code token renderer prototype:

```

13128 \cs_set_eq:NN
13129   \c_@@_dot_previous_definition:nmn
13130   \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:

```

13131 \regex_const:Nn
13132   \c_@@_dot_infostring_regex
13133   { ^dot(\s+(.+))? }
13134 \seq_new:N
13135   \l_@@_dot_matches_seq
13136 \markdownSetup {
13137   rendererPrototypes = {
13138     inputFencedCode = {
13139       \regex_extract_once:NnNTF
13140         \c_@@_dot_infostring_regex
13141         { #2 }
13142         \l_@@_dot_matches_seq
13143         {
13144           \@@_if_option:nF
13145             { frozenCache }
13146             {
13147               \sys_shell_now:n
13148               {
13149                 if~!~test~-e~#1.pdf.source~
13150                 ||~!~diff~#1~#1.pdf.source;
13151                 then~
13152                   dot~-Tpdf~-o~#1.pdf~#1;
13153                   cp~#1~#1.pdf.source;
13154                 fi
13155               }
13156             }

```

We include the typeset image using the image token renderer:

```

13157         \exp_args:NNne
13158         \exp_last_unbraced:No
13159         \markdownRendererImage
13160         {
13161             { Graphviz~image }
13162             { #1.pdf }
13163             { #1.pdf }
13164         }
13165         {
13166             \seq_item:Nn
13167             \l_@@_dot_matches_seq
13168             { 3 }
13169         }
13170     }

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

13171     {
13172         \@@_dot_previous_definition:nnn
13173         { #1 }
13174         { #2 }
13175         { #3 }
13176     }
13177 },
13178 },
13179 }
13180 }

```

The theme `witiko/diagrams` loads either the theme `witiko/dot` for version `v1` or the theme `witiko/diagrams/v2` for version `v2`.

```

13181 \prop_gput:Nnn
13182 \g_@@_plain_tex_built_in_themes_prop
13183 { witiko / diagrams }
13184 {
13185     \str_case:enF
13186     { \markdownThemeVersion }
13187     {
13188         { latest }
13189         {
13190             \markdownWarning
13191             {
13192                 Write~"witiko/diagrams@v2"~to~pin~version~"v2"~of~the~
13193                 theme~"witiko/diagrams".~This~will~keep~your~documents~
13194                 from~suddenly~breaking~when~we~have~released~future~
13195                 versions~of~the~theme~with~backwards~incompatible~
13196                 syntax~and~behavior.
13197             }
13198         }

```

```

13199         {
13200             import = witiko/diagrams/v2,
13201         }
13202     }
13203     { v2 }
13204     {
13205         \markdownSetup
13206         {
13207             import = witiko/diagrams/v2,
13208         }
13209     }
13210     { v1 }
13211     {
13212         \markdownSetup
13213         {
13214             import = witiko/dot@silent,
13215         }
13216     }
13217 }
13218 {
13219     \msg_error:nnnn
13220     { markdown }
13221     { unknown-theme-version }
13222     { witiko/diagrams }
13223     { \markdownThemeVersion }
13224     { v1 }
13225 }
13226 }
13227 \cs_generate_variant:Nn
13228 \msg_error:nnnnn
13229 { nnnn }
13230 \msg_new:nnnn
13231 { markdown }
13232 { unknown-theme-version }
13233 { Unknown~version~"#2"~of~theme~"#1"~has~been~requested. }
13234 { Known~versions~are:~#3 }

```

Next, we implement the theme `witiko/diagrams/v2`.

```

13235 \prop_gput:Nnn
13236 \g_@@_plain_tex_built_in_themes_prop
13237 { witiko / diagrams / v2 }
13238 {

```

We enable the `fencedCode` and `fencedCodeAttributes` Lua option.

```

13239     \@@_setup:n
13240     {
13241         fencedCode = true,
13242         fencedCodeAttributes = true,

```

```
13243     }
```

Store the previous fenced code token renderer prototype.

```
13244     \cs_set_eq:NN
13245     \@@_diagrams_previous_fenced_code:nnn
13246     \markdownRendererInputFencedCodePrototype
```

Store the caption of the diagram.

```
13247     \tl_new:N
13248     \l_@@_diagrams_caption_tl
13249     \@@_setup:n
13250     {
13251     rendererPrototypes = {
```

Route attributes on fenced code blocks to the image attribute renderer prototypes.

```
13252         fencedCodeAttributeContextBegin = {
13253             \group_begin:
13254             \markdownRendererImageAttributeContextBegin
13255             \cs_set_eq:NN
13256             \@@_diagrams_previous_key_value:nn
13257             \markdownRendererAttributeKeyValuePrototype
13258             \@@_setup:n
13259             {
13260                 rendererPrototypes = {
13261                     attributeKeyValue = {
13262                         \str_if_eq:nnTF
13263                         { ##1 }
13264                         { caption }
13265                         {
13266                             \tl_set:Nn
13267                             \l_@@_diagrams_caption_tl
13268                             { ##2 }
13269                         }
13270                     }
13271                     \@@_diagrams_previous_key_value:nn
13272                     { ##1 }
13273                     { ##2 }
13274                 }
13275             },
13276         },
13277     },
13278     fencedCodeAttributeContextEnd = {
13279         \markdownRendererImageAttributeContextEnd
13280         \group_end:
13281     },
13282 },
13283 },
13284 }
```

```

13285 \cs_new:Nn
13286 \@@_diagrams_render_diagram:nnnn
13287 {
13288   \@@_if_option:nF
13289     { frozenCache }
13290     {
13291       \sys_shell_now:n
13292       {
13293         if~!~test~-e~#2.source~
13294         ||~!~diff~#1~#2.source;
13295         then~
13296           (#3);
13297           cp~#1~#2.source;
13298         fi
13299       }
13300       \exp_args:NNnV
13301       \exp_last_unbraced:No
13302       \markdownRendererImage
13303       {
13304         { #4 }
13305         { #2 }
13306         { #2 }
13307       }
13308       \l_@@_diagrams_caption_tl
13309     }
13310 }

```

Use the prop `\g_markdown_diagrams_infostrings_prop` to determine how the code with a given infostring should be processed and routed to the token renderer prototype(s) for images.

```

13311 \prop_new:N
13312 \g_markdown_diagrams_infostrings_prop

```

If we know a processing function for a given infostring, use it.

```

13313 \@@_setup:n
13314 {
13315   rendererPrototypes = {
13316     inputFencedCode = {
13317       \prop_get:NnNTF
13318       \g_markdown_diagrams_infostrings_prop
13319       { #2 }
13320       \l_tmpa_tl
13321       {
13322         \cs_set:NV
13323         \@@_diagrams_infostrings_current:n
13324         \l_tmpa_tl
13325         \@@_diagrams_infostrings_current:n
13326         { #1 }

```

```
13327         }
```

Otherwise, use the previous fenced code token renderer prototype.

```
13328         {
13329             \@@_diagrams_previous_fenced_code:n
13330             { #1 }
13331             { #2 }
13332             { #3 }
13333         }
13334     },
13335 },
13336 }
13337 \cs_generate_variant:Nn
13338 \cs_set:Nn
13339 { NV }
```

Typeset fenced code with infostring `dot` using the command `dot` from the package `Graphviz`.

```
13340 \cs_set:Nn
13341 \@@_diagrams_infostrings_current:n
13342 {
13343     \@@_diagrams_render_diagram:n
13344     { #1 }
13345     { #1.pdf }
13346     { dot~--Tpdf~-o~#1.pdf~#1 }
13347     { Graphviz~image }
13348 }
13349 \@@_tl_set_from_cs:NNn
13350 \l_tmpa_tl
13351 \@@_diagrams_infostrings_current:n
13352 { 1 }
13353 \prop_gput:NnV
13354 \g_markdown_diagrams_infostrings_prop
13355 { dot }
13356 \l_tmpa_tl
```

Typeset fenced code with infostring `mermaid` using the command `mmdc` from the npm package `@mermaid-js/mermaid-cli`.

```
13357 \cs_set:Nn
13358 \@@_diagrams_infostrings_current:n
13359 {
13360     \@@_diagrams_render_diagram:n
13361     { #1 }
13362     { #1.pdf }
13363     { mmdc~--pdfFit~-i~#1~-o~#1.pdf }
13364     { Mermaid~image }
13365 }
13366 \@@_tl_set_from_cs:NNn
```

```

13367     \l_tmpa_tl
13368     \@@_diagrams_infostrings_current:n
13369     { 1 }
13370     \prop_gput:NnV
13371     \g_markdown_diagrams_infostrings_prop
13372     { mermaid }
13373     \l_tmpa_tl

```

Typeset fenced code with infostring `plantuml` using the command `plantuml` from the package PlantUML.

```

13374     \regex_const:Nn
13375     \c_@@_diagrams_filename_suffix_regex
13376     { \.[^\.]*$ }
13377     \cs_set:Nn
13378     \@@_diagrams_infostrings_current:n
13379     {
13380         \tl_set:Nn
13381         \l_tmpa_tl
13382         { #1 }
13383         \regex_replace_once:NnN
13384         \c_@@_diagrams_filename_suffix_regex
13385         { .pdf }
13386         \l_tmpa_tl
13387         \@@_diagrams_render_diagram:nVnn
13388         { #1 }
13389         \l_tmpa_tl
13390         { plantuml~--tpdf~#1 }
13391         { PlantUML~image }
13392     }
13393     \cs_generate_variant:Nn
13394     \@@_diagrams_render_diagram:nnnn
13395     { nVnn }
13396     \@@_tl_set_from_cs:NNn
13397     \l_tmpa_tl
13398     \@@_diagrams_infostrings_current:n
13399     { 1 }
13400     \prop_gput:NnV
13401     \g_markdown_diagrams_infostrings_prop
13402     { plantuml }
13403     \l_tmpa_tl
13404 }

```

We locally change the category code of percent signs, so that we can use them in the shell code:

```

13405 \group_begin:
13406 \char_set_catcode_other:N \%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

13407 \prop_gput:Nnn
13408 \g_@@_plain_tex_built_in_themes_prop
13409 { witiko / graphicx / http }
13410 {
13411   \cs_set_eq:NN
13412     \@@_graphicx_http_previous_definition:nnnn
13413     \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.

```

13414   \int_new:N
13415     \g_@@_graphicx_http_image_number_int
13416   \int_gset:Nn
13417     \g_@@_graphicx_http_image_number_int
13418     { 0 }
13419   \cs_new:Nn
13420     \@@_graphicx_http_filename:
13421     {
13422       \markdownOptionCacheDir
13423       / witiko_graphicx_http .
13424       \int_use:N
13425         \g_@@_graphicx_http_image_number_int
13426     }

```

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.

```

13427   \cs_new:Nn
13428     \@@_graphicx_http_download:nn
13429     {
13430       wget~-0~#2~#1~
13431       ||~curl~---location~-o~#2~#1~
13432       ||~rm~-f~#2
13433     }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

13434   \str_new:N
13435     \l_@@_graphicx_http_filename_str
13436   \ior_new:N
13437     \g_@@_graphicx_http_filename_ior
13438   \markdownSetup {
13439     rendererPrototypes = {
13440       image = {

```



```

13441     \@@_if_option:nF
13442     { frozenCache }
13443     {

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```

13444     \sys_shell_now:e
13445     {
13446     mkdir~p~" \markdownOptionCacheDir ";
13447     if~printf~'%s'~"#3"~|~grep~q~E~'^https?:';
13448     then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

13449     OUTPUT_PREFIX=" \markdownOptionCacheDir ";
13450     OUTPUT_BODY="$ (printf~'%s'~'#3'
13451     |~md5sum~|~cut~-d'~'~-f1)";
13452     OUTPUT_SUFFIX="$ (printf~'%s'~'#3'
13453     |~sed~'s/.*[.]//)";
13454     OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

13455     if~!~[~e~"$OUTPUT"~];
13456     then~
13457     \@@_graphicx_http_download:nn
13458     { '#3' }
13459     { "$OUTPUT" } ;
13460     printf~'%s'~"$OUTPUT"~
13461     >~" \@@_graphicx_http_filename: ";
13462     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:

```

13463     else~
13464     printf~'%s'~'#3'~
13465     >~" \@@_graphicx_http_filename: ";
13466     fi
13467     }
13468     }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

13469     \ior_open:Ne
13470     \g_@@_graphicx_http_filename_ior
13471     { \@@_graphicx_http_filename: }
13472     \ior_str_get:NN
13473     \g_@@_graphicx_http_filename_ior
13474     \l_@@_graphicx_http_filename_str
13475     \ior_close:N

```

```

13476         \g_@@_graphicx_http_filename_ior
13477     \@@_graphicx_http_previous_definition:nnVn
13478         { #1 }
13479         { #2 }
13480         \l_@@_graphicx_http_filename_str
13481         { #4 }
13482     \int_gincr:N
13483         \g_@@_graphicx_http_image_number_int
13484     }
13485 }
13486 }
13487 \cs_generate_variant:Nn
13488     \ior_open:Nn
13489     { Ne }
13490 \cs_generate_variant:Nn
13491     \@@_graphicx_http_previous_definition:nnnn
13492     { nnVn }
13493 }
13494 \group_end:

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

13495 \prop_gput:Nnn
13496     \g_@@_plain_tex_built_in_themes_prop
13497     { witiko / tilde }
13498     {
13499         \markdownSetup {
13500             rendererPrototypes = {
13501                 tilde = {-},
13502             },
13503         }
13504     }

```

The themes `witiko/example/foo` and `witiko/example/bar` are supposed to be used in code examples. They don't do anything.

```

13505 \clist_map_inline:nn
13506     { foo, bar }
13507     {
13508         \prop_gput:Nnn
13509             \g_@@_plain_tex_built_in_themes_prop
13510             { witiko / example / #1 }
13511             {
13512                 \markdownWarning
13513                 {
13514                     The~theme~witiko/example/#1~is~supposed~to~be~used~in~code~
13515                     examples.~Using~it~in~actual~code~has~no~effect,~except~
13516                     this~warning~message,~and~is~usually~a~mistake.
13517                 }

```

```

13518     }
13519   }
13520 \ExplSyntaxOff

```

The `witiko/markdown/defaults` plain  $\TeX$  theme provides default definitions for token renderer prototypes. See Section 3.2.3 for the actual definitions.

### 3.2.3 Token Renderer Prototypes

The following definitions should be considered placeholder.

```

13521 \def\markdownRendererInterblockSeparatorPrototype{\par}%
13522 \def\markdownRendererParagraphSeparatorPrototype{%
13523   \markdownRendererInterblockSeparator}%
13524 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
13525 \def\markdownRendererSoftLineBreakPrototype{ }%
13526 \let\markdownRendererEllipsisPrototype\dots
13527 \def\markdownRendererNbspPrototype{~}%
13528 \def\markdownRendererLeftBracePrototype{\char`\{}%
13529 \def\markdownRendererRightBracePrototype{\char`\}%
13530 \def\markdownRendererDollarSignPrototype{\char`\$}%
13531 \def\markdownRendererPercentSignPrototype{\char`\}%
13532 \def\markdownRendererAmpersandPrototype{\&%
13533 \def\markdownRendererUnderscorePrototype{\char`\_}%
13534 \def\markdownRendererHashPrototype{\char`\#}%
13535 \def\markdownRendererCircumflexPrototype{\char`\^}%
13536 \def\markdownRendererBackslashPrototype{\char`\}%
13537 \def\markdownRendererTildePrototype{\char`\~}%
13538 \def\markdownRendererPipePrototype{|}%
13539 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
13540 \def\markdownRendererLinkPrototype#1#2#3#4#2}%
13541 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13542   \markdownInput{#3}}%
13543 \def\markdownRendererContentBlockOnlineImagePrototype{%
13544   \markdownRendererImage}%
13545 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
13546   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
13547 \def\markdownRendererImagePrototype#1#2#3#4#2}%
13548 \def\markdownRendererUlBeginPrototype{}%
13549 \def\markdownRendererUlBeginTightPrototype{}%
13550 \def\markdownRendererUlItemPrototype{}%
13551 \def\markdownRendererUlItemEndPrototype{}%
13552 \def\markdownRendererUlEndPrototype{}%
13553 \def\markdownRendererUlEndTightPrototype{}%
13554 \def\markdownRendererOlBeginPrototype{}%
13555 \def\markdownRendererOlBeginTightPrototype{}%
13556 \def\markdownRendererFancyOlBeginPrototype#1#2{%
13557   \markdownRendererOlBegin}%

```

13558 \def\markdownRendererFancyO1BeginTightPrototype#1#2{%  
13559 \markdownRendererO1BeginTight}%  
13560 \def\markdownRendererO1ItemPrototype{%}  
13561 \def\markdownRendererO1ItemWithNumberPrototype#1{%}  
13562 \def\markdownRendererO1ItemEndPrototype{%}  
13563 \def\markdownRendererFancyO1ItemPrototype{\markdownRendererO1Item}%  
13564 \def\markdownRendererFancyO1ItemWithNumberPrototype{%  
13565 \markdownRendererO1ItemWithNumber}%  
13566 \def\markdownRendererFancyO1ItemEndPrototype{%}  
13567 \def\markdownRendererO1EndPrototype{%}  
13568 \def\markdownRendererO1EndTightPrototype{%}  
13569 \def\markdownRendererFancyO1EndPrototype{\markdownRendererO1End}%  
13570 \def\markdownRendererFancyO1EndTightPrototype{%  
13571 \markdownRendererO1EndTight}%  
13572 \def\markdownRendererDlBeginPrototype{%}  
13573 \def\markdownRendererDlBeginTightPrototype{%}  
13574 \def\markdownRendererDlItemPrototype#1{#1}%  
13575 \def\markdownRendererDlItemEndPrototype{%}  
13576 \def\markdownRendererDlDefinitionBeginPrototype{%}  
13577 \def\markdownRendererDlDefinitionEndPrototype{\par}%  
13578 \def\markdownRendererDlEndPrototype{%}  
13579 \def\markdownRendererDlEndTightPrototype{%}  
13580 \def\markdownRendererEmphasisPrototype#1{\it#1}%  
13581 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%  
13582 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%  
13583 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%  
13584 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=Opt}%  
13585 \def\markdownRendererLineBlockEndPrototype{\endgroup}%  
13586 \def\markdownRendererInputVerbatimPrototype#1{%  
13587 \par{\tt\input#1\relax}\par}%  
13588 \def\markdownRendererInputFencedCodePrototype#1#2#3{%  
13589 \markdownRendererInputVerbatim{#1}}%  
13590 \def\markdownRendererHeadingOnePrototype#1{#1}%  
13591 \def\markdownRendererHeadingTwoPrototype#1{#1}%  
13592 \def\markdownRendererHeadingThreePrototype#1{#1}%  
13593 \def\markdownRendererHeadingFourPrototype#1{#1}%  
13594 \def\markdownRendererHeadingFivePrototype#1{#1}%  
13595 \def\markdownRendererHeadingSixPrototype#1{#1}%  
13596 \def\markdownRendererThematicBreakPrototype{%}  
13597 \def\markdownRendererNotePrototype#1{#1}%  
13598 \def\markdownRendererCitePrototype#1{%}  
13599 \def\markdownRendererTextCitePrototype#1{%}  
13600 \def\markdownRendererTickedBoxPrototype{[X]}%  
13601 \def\markdownRendererHalfTickedBoxPrototype{[/]}%  
13602 \def\markdownRendererUntickedBoxPrototype{[ ]}%  
13603 \def\markdownRendererStrikeThroughPrototype#1{#1}%  
13604 \def\markdownRendererSuperscriptPrototype#1{#1}%

```

13605 \def\markdownRendererSubscriptPrototype#1{#1}%
13606 \def\markdownRendererDisplayMathPrototype#1{##1$$}%
13607 \def\markdownRendererInlineMathPrototype#1{##1$}%
13608 \ExplSyntaxOn
13609 \cs_gset:Npn
13610 \markdownRendererHeaderAttributeContextBeginPrototype
13611 {
13612   \group_begin:
13613   \color_group_begin:
13614 }
13615 \cs_gset:Npn
13616 \markdownRendererHeaderAttributeContextEndPrototype
13617 {
13618   \color_group_end:
13619   \group_end:
13620 }
13621 \cs_gset_eq:NN
13622 \markdownRendererBracketedSpanAttributeContextBeginPrototype
13623 \markdownRendererHeaderAttributeContextBeginPrototype
13624 \cs_gset_eq:NN
13625 \markdownRendererBracketedSpanAttributeContextEndPrototype
13626 \markdownRendererHeaderAttributeContextEndPrototype
13627 \cs_gset_eq:NN
13628 \markdownRendererFencedDivAttributeContextBeginPrototype
13629 \markdownRendererHeaderAttributeContextBeginPrototype
13630 \cs_gset_eq:NN
13631 \markdownRendererFencedDivAttributeContextEndPrototype
13632 \markdownRendererHeaderAttributeContextEndPrototype
13633 \cs_gset_eq:NN
13634 \markdownRendererFencedCodeAttributeContextBeginPrototype
13635 \markdownRendererHeaderAttributeContextBeginPrototype
13636 \cs_gset_eq:NN
13637 \markdownRendererFencedCodeAttributeContextEndPrototype
13638 \markdownRendererHeaderAttributeContextEndPrototype
13639 \cs_gset:Npn
13640 \markdownRendererReplacementCharacterPrototype
13641 { \codepoint_str_generate:n { fffd } }
13642 \ExplSyntaxOff
13643 \def\markdownRendererSectionBeginPrototype{}%
13644 \def\markdownRendererSectionEndPrototype{}%
13645 \ExplSyntaxOn
13646 \cs_gset:Npn
13647 \markdownRendererWarningPrototype
13648 #1#2#3#4
13649 {
13650   \tl_set:Nn
13651     \l_tmpa_tl

```

```

13652     { #2 }
13653   \tl_if_empty:nF
13654     { #4 }
13655     {
13656       \tl_put_right:Nn
13657         \l_tmpa_tl
13658         { \iow_newline: #4 }
13659     }
13660   \exp_args:NV
13661     \markdownWarning
13662     \l_tmpa_tl
13663 }
13664 \ExplSyntaxOff
13665 \def\markdownRendererErrorPrototype#1#2#3#4{%
13666   \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.

```

13667 \ExplSyntaxOn
13668 \cs_new:Nn
13669   \@@_plain_tex_default_input_raw_inline:nn
13670   {
13671     \str_case:nn
13672       { #2 }
13673       {
13674         { md } { \markdownInput{#1} }
13675         { tex } { \markdownEscape{#1} \unskip }
13676       }
13677   }
13678 \cs_new:Nn
13679   \@@_plain_tex_default_input_raw_block:nn
13680   {
13681     \str_case:nn
13682       { #2 }
13683       {
13684         { md } { \markdownInput{#1} }
13685         { tex } { \markdownEscape{#1} }
13686       }
13687   }
13688 \cs_gset:Npn
13689   \markdownRendererInputRawInlinePrototype#1#2
13690   {
13691     \@@_plain_tex_default_input_raw_inline:nn
13692     { #1 }

```

```

13693     { #2 }
13694   }
13695 \cs_gset:Npn
13696   \markdownRendererInputRawBlockPrototype#1#2
13697   {
13698     \@@_plain_tex_default_input_raw_block:nn
13699     { #1 }
13700     { #2 }
13701   }
13702 \ExplSyntaxOff

```

### 3.2.3.2 Simple YAML Metadata Renderer Prototypes

In this section, we implement the simple high-level interface for processing simple YAML metadata using the key-value `markdown/jekyllData`. See also Section 2.2.6.1.

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position  $p$ :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth  $p$ .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth  $p$ .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth  $p$ .

```

13703 \ExplSyntaxOn
13704 \seq_new:N \g_@@_jekyll_data_datatypes_seq
13705 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
13706 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
13707 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\@@_jekyll_data_push_address_segment:n` macro.

```

13708 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
13709 \cs_new:Nn \@@_jekyll_data_push_address_segment:n
13710   {
13711     \seq_if_empty:NF
13712     \g_@@_jekyll_data_datatypes_seq
13713     {
13714       \seq_get_right:NN
13715       \g_@@_jekyll_data_datatypes_seq
13716       \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.

```

13717     \str_if_eq:NNTF
13718         \l_tmpa_tl
13719         \c_@@_jekyll_data_sequence_tl
13720         {
13721             \seq_put_right:Nn
13722                 \g_@@_jekyll_data_wildcard_absolute_address_seq
13723                 { * }
13724         }
13725         {
13726             \seq_put_right:Nn
13727                 \g_@@_jekyll_data_wildcard_absolute_address_seq
13728                 { #1 }
13729         }
13730     }
13731 }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\@@_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\@@_jekyll_data_update_address_tls:` macro.



```

13732 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
13733 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
13734 \cs_new:Nn \@@_jekyll_data_concatenate_address:NN
13735 {
13736   \seq_pop_left:NN #1 \l_tmpa_tl
13737   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
13738   \seq_put_left:NV #1 \l_tmpa_tl
13739 }
13740 \cs_new:Nn \@@_jekyll_data_update_address_tls:
13741 {
13742   \@@_jekyll_data_concatenate_address:NN
13743   \g_@@_jekyll_data_wildcard_absolute_address_seq
13744   \g_@@_jekyll_data_wildcard_absolute_address_tl
13745   \seq_get_right:NN
13746   \g_@@_jekyll_data_wildcard_absolute_address_seq
13747   \g_@@_jekyll_data_wildcard_relative_address_tl
13748 }

```

To make sure that the stacks and token lists stay in sync, we will use the `\@@_jekyll_data_push:nN` and `\@@_jekyll_data_pop:` macros.

```

13749 \cs_new:Nn \@@_jekyll_data_push:nN
13750 {
13751   \@@_jekyll_data_push_address_segment:n
13752   { #1 }
13753   \seq_put_right:NV
13754   \g_@@_jekyll_data_datatypes_seq
13755   #2
13756   \@@_jekyll_data_update_address_tls:
13757 }
13758 \cs_new:Nn \@@_jekyll_data_pop:
13759 {
13760   \seq_pop_right:NN
13761   \g_@@_jekyll_data_wildcard_absolute_address_seq
13762   \l_tmpa_tl
13763   \seq_pop_right:NN
13764   \g_@@_jekyll_data_datatypes_seq
13765   \l_tmpa_tl
13766   \@@_jekyll_data_update_address_tls:
13767 }

```

To set a single key–value, we will use the `\@@_jekyll_data_set_keyval_known:nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\@@_jekyll_data_set_keyvals_known:nn` macro.

```

13768 \cs_new:Nn \@@_jekyll_data_set_keyval_known:nn
13769 {
13770   \keys_set_known:nn
13771   { markdown/jekyllData }
13772   { { #1 } = { #2 } }

```

```

13773 }
13774 \cs_generate_variant:Nn
13775   \@@_jekyll_data_set_keyval_known:nn
13776   { Vn }
13777 \cs_new:Nn \@@_jekyll_data_set_keyvals_known:nn
13778   {
13779   \@@_jekyll_data_push:nN
13780     { #1 }
13781     \c_@@_jekyll_data_scalar_tl
13782   \@@_jekyll_data_set_keyval_known:Vn
13783     \g_@@_jekyll_data_wildcard_absolute_address_tl
13784     { #2 }
13785   \@@_jekyll_data_set_keyval_known:Vn
13786     \g_@@_jekyll_data_wildcard_relative_address_tl
13787     { #2 }
13788   \@@_jekyll_data_pop:
13789   }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

13790 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13791   \@@_jekyll_data_push:nN
13792     { #1 }
13793   \c_@@_jekyll_data_sequence_tl
13794 }
13795 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13796   \@@_jekyll_data_push:nN
13797     { #1 }
13798   \c_@@_jekyll_data_mapping_tl
13799 }
13800 \def\markdownRendererJekyllDataSequenceEndPrototype{
13801   \@@_jekyll_data_pop:
13802 }
13803 \def\markdownRendererJekyllDataMappingEndPrototype{
13804   \@@_jekyll_data_pop:
13805 }
13806 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13807   \@@_jekyll_data_set_keyvals_known:nn
13808     { #1 }
13809     { #2 }
13810 }
13811 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13812 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13813   \@@_jekyll_data_set_keyvals_known:nn
13814     { #1 }
13815     { #2 }
13816 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13817 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13818 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13819   \@@_jekyll_data_set_keyvals_known:nn
13820   { #1 }
13821   { #2 }
13822 }
13823 \ExplSyntaxOff

```

### 3.2.3.3 Complex YAML Metadata Renderer Prototypes

In this section, we implement the high-level interface for routing complex YAML metadata to expl3 key-values using the option `jekyllDataKeyValue=<module>`. See also Section 2.2.6.1.

```

13824 \ExplSyntaxOn
13825 \@@_with_various_cases:nn
13826 { jekyllDataKeyValue }
13827 {
13828   \keys_define:nn
13829   { markdown/options }
13830   {
13831     #1 .code:n = {
13832       \@@_route_jekyll_data_to_key_values:n
13833       { ##1 }
13834     },

```

When no `<module>` has been provided, assume that the YAML metadata specify absolute paths to key-values.

```

13835     #1 .default:n = { },
13836   }
13837 }
13838 \seq_new:N
13839 \l_@@_jekyll_data_current_position_seq
13840 \tl_new:N
13841 \l_@@_jekyll_data_current_position_tl
13842 \cs_new:Nn
13843 \@@_route_jekyll_data_to_key_values:n
13844 {
13845   \markdownSetup
13846   {
13847     renderers = {
13848       jekyllData(Sequence|Mapping)Begin = {
13849         \bool_lazy_and:nnTF
13850         {
13851           \seq_if_empty_p:N
13852           \l_@@_jekyll_data_current_position_seq

```

```

13853     }
13854     {
13855         \str_if_eq_p:nn
13856         { ##1 }
13857         { null }
13858     }
13859     {
13860         \tl_if_empty:nF
13861         { #1 }
13862         {
13863             \seq_put_right:Nn
13864             \l_@@_jekyll_data_current_position_seq
13865             { #1 }
13866         }
13867     }
13868     {
13869         \seq_put_right:Nn
13870         \l_@@_jekyll_data_current_position_seq
13871         { ##1 }
13872     }
13873 },
13874 jekyllData(Sequence|Mapping)End = {
13875     \seq_pop_right:NN
13876     \l_@@_jekyll_data_current_position_seq
13877     \l_tmpa_tl
13878 },

```

For every YAML key `path.to.<key>` with a value of type *<non-string type>*, set the key *<non-string type>* of the key-value `<module>/path/to/<key>` if it is known and the key *<key>* of the key-value `<module>/path/to` otherwise. *<Non-string type>* is one of `boolean`, `number`, and `empty`.

```

13879     jekyllDataBoolean = {
13880         \tl_set:Nx
13881         \l_@@_jekyll_data_current_position_tl
13882         {
13883             \seq_use:Nn
13884             \l_@@_jekyll_data_current_position_seq
13885             { / }
13886         }
13887         \keys_if_exist:VnTF
13888         \l_@@_jekyll_data_current_position_tl
13889         { ##1 / boolean }
13890         {
13891             \@@_keys_set:xn
13892             {
13893                 \tl_use:N
13894                 \l_@@_jekyll_data_current_position_tl

```

```

13895         / ##1 / boolean
13896     }
13897     { ##2 }
13898 }
13899 {
13900     \@@_keys_set:xn
13901     {
13902         \tl_use:N
13903         \l_@@_jekyll_data_current_position_tl
13904         / ##1
13905     }
13906     { ##2 }
13907 }
13908 },
13909 jekyllDataNumber = {
13910     \tl_set:Nx
13911     \l_@@_jekyll_data_current_position_tl
13912     {
13913         \seq_use:Nn
13914         \l_@@_jekyll_data_current_position_seq
13915         { / }
13916     }
13917     \keys_if_exist:VnTF
13918     \l_@@_jekyll_data_current_position_tl
13919     { ##1 / number }
13920     {
13921         \@@_keys_set:xn
13922         {
13923             \tl_use:N
13924             \l_@@_jekyll_data_current_position_tl
13925             / ##1 / number
13926         }
13927         { ##2 }
13928     }
13929     {
13930         \@@_keys_set:xn
13931         {
13932             \tl_use:N
13933             \l_@@_jekyll_data_current_position_tl
13934             / ##1
13935         }
13936         { ##2 }
13937     }
13938 },

```

For the  $\langle non-string\ type \rangle$  of `empty`, no value is passed to the key-value. Therefore,

a default value should always be defined for nullable keys using the key property `.default:n`.

```

13939     jekyllDataEmpty = {
13940         \tl_set:Nx
13941         \l_@@_jekyll_data_current_position_tl
13942         {
13943             \seq_use:Nn
13944                 \l_@@_jekyll_data_current_position_seq
13945                 { / }
13946         }
13947         \keys_if_exist:VnTF
13948             \l_@@_jekyll_data_current_position_tl
13949             { ##1 / empty }
13950         {
13951             \keys_set:xn
13952             {
13953                 \tl_use:N
13954                     \l_@@_jekyll_data_current_position_tl
13955                 / ##1
13956             }
13957             { empty }
13958         }
13959         {
13960             \keys_set:Vn
13961                 \l_@@_jekyll_data_current_position_tl
13962                 { ##1 }
13963         }
13964     },

```

For every YAML key `path.to.<key>` with a value of type `string`, set the keys `typographicString` and `programmaticString` of the key–value `<module>/path/to/<key>` if they are known with the typographic and programmatic strings of the value, respectively. Furthermore, set the key `<key>` of the key–value `<module>/path/to` with the typographic string of the value unless the key `typographicString` is known. If the key `programmaticString` is known, only set the key `<key>` if it is known. In contrast, if neither `typographicString` nor `programmaticString` are known, set `<key>` normally, i.e. regardless of whether it is known or unknown.

```

13965     jekyllDataTypographicString = {
13966         \tl_set:Nx
13967         \l_@@_jekyll_data_current_position_tl
13968         {
13969             \seq_use:Nn
13970                 \l_@@_jekyll_data_current_position_seq
13971                 { / }
13972         }

```

```

13973 \keys_if_exist:VnTF
13974 \l_@@_jekyll_data_current_position_tl
13975 { ##1 / typographicString }
13976 {
13977 \@@_keys_set:xn
13978 {
13979 \tl_use:N
13980 \l_@@_jekyll_data_current_position_tl
13981 / ##1 / typographicString
13982 }
13983 { ##2 }
13984 }
13985 {
13986 \keys_if_exist:VnTF
13987 \l_@@_jekyll_data_current_position_tl
13988 { ##1 / programmaticString }
13989 {
13990 \@@_keys_set_known:xn
13991 {
13992 \tl_use:N
13993 \l_@@_jekyll_data_current_position_tl
13994 / ##1
13995 }
13996 { ##2 }
13997 }
13998 {
13999 \@@_keys_set:xn
14000 {
14001 \tl_use:N
14002 \l_@@_jekyll_data_current_position_tl
14003 / ##1
14004 }
14005 { ##2 }
14006 }
14007 }
14008 },
14009 jekyllDataProgrammaticString = {
14010 \tl_set:Nx
14011 \l_@@_jekyll_data_current_position_tl
14012 {
14013 \seq_use:Nn
14014 \l_@@_jekyll_data_current_position_seq
14015 { / }
14016 }
14017 \keys_if_exist:VnT
14018 \l_@@_jekyll_data_current_position_tl
14019 { ##1 / programmaticString }

```

```

14020         {
14021         \@@_keys_set:xn
14022         {
14023         \tl_use:N
14024         \l_@@_jekyll_data_current_position_tl
14025         / ##1 / programmaticString
14026         }
14027         { ##2 }
14028         }
14029     },
14030 },
14031 }
14032 }
14033 \cs_new:Nn
14034 \@@_keys_set:nn
14035 {
14036     \keys_set:nn
14037     { }
14038     { { #1 } = { #2 } }
14039 }
14040 \cs_new:Nn
14041 \@@_keys_set_known:nn
14042 {
14043     \keys_set_known:nn
14044     { }
14045     { { #1 } = { #2 } }
14046 }
14047 \cs_generate_variant:Nn
14048 \@@_keys_set:nn
14049 { xn }
14050 \cs_generate_variant:Nn
14051 \@@_keys_set_known:nn
14052 { xn }
14053 \cs_generate_variant:Nn
14054 \keys_set:nn
14055 { xn, Vn }
14056 \prg_generate_conditional_variant:Nnn
14057 \keys_if_exist:nn
14058 { Vn }
14059 { T, TF }
14060 \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).

```

14061 \ExplSyntaxOn
14062 \str_if_eq:VVT

```



```

14063 \c_@@_top_layer_tl
14064 \c_@@_option_layer_plain_tex_tl
14065 {
14066   \use:c
14067     { ExplSyntaxOff }
14068   \@@_if_option:nF
14069     { noDefaults }
14070     {
14071       \@@_if_option:nTF
14072         { experimental }
14073         {
14074           \@@_setup:n
14075             { theme = witiko/markdown/defaults@experimental }
14076         }
14077         {
14078           \@@_setup:n
14079             { theme = witiko/markdown/defaults }
14080         }
14081     }
14082   \use:c
14083     { ExplSyntaxOn }
14084 }
14085 \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).

```

14086 \ExplSyntaxOn
14087 \tl_new:N \g_@@_formatted_lua_options_tl
14088 \cs_new:Nn \@@_format_lua_options:
14089 {
14090   \tl_gclear:N
14091     \g_@@_formatted_lua_options_tl
14092   \seq_map_function:NN
14093     \g_@@_lua_options_seq
14094     \@@_format_lua_option:n
14095 }
14096 \cs_new:Nn \@@_format_lua_option:n
14097 {
14098   \@@_typecheck_option:n
14099     { #1 }
14100   \@@_get_option_type:nN
14101     { #1 }
14102   \l_tmpa_tl

```

```

14103 \bool_case_true:nF
14104 {
14105   {
14106     \str_if_eq_p:VV
14107     \l_tmpa_tl
14108     \c_@@_option_type_boolean_tl ||
14109     \str_if_eq_p:VV
14110     \l_tmpa_tl
14111     \c_@@_option_type_number_tl ||
14112     \str_if_eq_p:VV
14113     \l_tmpa_tl
14114     \c_@@_option_type_counter_tl
14115   }
14116   {
14117     \@@_get_option_value:nN
14118     { #1 }
14119     \l_tmpa_tl
14120     \tl_gput_right:Nx
14121     \g_@@_formatted_lua_options_tl
14122     { #1~::~ \l_tmpa_tl ,~ }
14123   }
14124 {
14125   \str_if_eq_p:VV
14126   \l_tmpa_tl
14127   \c_@@_option_type_clist_tl
14128 }
14129 {
14130   \@@_get_option_value:nN
14131   { #1 }
14132   \l_tmpa_tl
14133   \tl_gput_right:Nx
14134   \g_@@_formatted_lua_options_tl
14135   { #1~::~\c_left_brace_str }
14136   \clist_map_inline:Vn
14137   \l_tmpa_tl
14138   {
14139     \@@_lua_escape:xN
14140     { ##1 }
14141     \l_tmpb_tl
14142     \tl_gput_right:Nn
14143     \g_@@_formatted_lua_options_tl
14144     { " }
14145     \tl_gput_right:NV
14146     \g_@@_formatted_lua_options_tl
14147     \l_tmpb_tl
14148     \tl_gput_right:Nn
14149     \g_@@_formatted_lua_options_tl

```

```

14150         { " ,~ }
14151     }
14152     \tl_gput_right:Nx
14153     \g_@@_formatted_lua_options_tl
14154     { \c_right_brace_str ,~ }
14155 }
14156 }
14157 {
14158     \@@_get_option_value:nN
14159     { #1 }
14160     \l_tmpa_tl
14161     \@@_lua_escape:xN
14162     { \l_tmpa_tl }
14163     \l_tmpb_tl
14164     \tl_gput_right:Nn
14165     \g_@@_formatted_lua_options_tl
14166     { #1~~~ " }
14167     \tl_gput_right:NV
14168     \g_@@_formatted_lua_options_tl
14169     \l_tmpb_tl
14170     \tl_gput_right:Nn
14171     \g_@@_formatted_lua_options_tl
14172     { " ,~ }
14173 }
14174 }
14175 \cs_generate_variant:Nn
14176 \clist_map_inline:nn
14177 { Vn }
14178 \let
14179 \markdownPrepareLuaOptions
14180 \@@_format_lua_options:
14181 \def
14182 \markdownLuaOptions
14183 {
14184     {
14185         \g_@@_formatted_lua_options_tl
14186     }
14187 }
14188 \sys_if_engine luatex:TF
14189 {
14190     \cs_new:Nn
14191     \@@_lua_escape:nN
14192     {
14193         \tl_set:Nx
14194         #2
14195         {
14196             \lua_escape:n

```

```

14197             { #1 }
14198         }
14199     }
14200 }
14201 {
14202     \regex_const:Nn
14203     \c_@@_lua_escape_regex
14204     { [\\"' ] }
14205     \cs_new:Nn
14206     \@@_lua_escape:nN
14207     {
14208         \tl_set:Nn
14209         #2
14210         { #1 }
14211         \regex_replace_all:NnN
14212         \c_@@_lua_escape_regex
14213         { \u { c_backslash_str } \0 }
14214         #2
14215     }
14216 }
14217 \cs_generate_variant:Nn
14218 \@@_lua_escape:nN
14219 { 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.

```

14220 \tl_new:N
14221 \markdownInputFilename
14222 \cs_new:Npn
14223 \markdownPrepareInputFilename
14224 #1
14225 {
14226     \@@_lua_escape:xN
14227     { #1 }
14228     \markdownInputFilename
14229     \tl_gset:Nx
14230     \markdownInputFilename
14231     { " \markdownInputFilename " }
14232 }

```

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.

```

14233 \cs_new:Npn
14234 \markdownPrepare
14235 {

```

First, ensure that the `cacheDir` directory exists.

```
14236     local~lfs = require("lfs")
14237     local~options = \markdownLuaOptions
14238     if~not~lfs.isdir(options.cacheDir) then~
14239         assert(lfs.mkdir(options.cacheDir))
14240     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.

```
14241     local~md = require("markdown")
14242     local~convert = md.new(options)
14243 }
```

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.

```
14244 \cs_new:Npn
14245   \markdownConvert
14246   {
14247     local~filename = \markdownInputFilename
14248     local~file = assert(io.open(filename, "r"),
14249       [[Could~not~open~file~]] .. filename .. [[~for~reading]])
14250     local~input = assert(file:read("*a"))
14251     assert(file:close())
14252     print(convert(input))
14253   }
14254 \ExplSyntaxOff
```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain `TeX`.

```
14255 \def\markdownCleanup{%
```

Remove the `options.cacheDir` directory if it is empty.

```
14256   if options.cacheDir then
14257     lfs.rmdir(options.cacheDir)
14258   end
14259 }%
```

### 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.

```
14260 \csname newread\endcsname\markdownInputFileStream
14261 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```
14262 \begingroup
```

```

14263 \catcode\^^I=12%
14264 \gdef\markdownReadAndConvertTab{^^I}%
14265 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX} 2_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

14266 \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.

```

14267 \catcode\^^M=13%
14268 \catcode\^^I=13%
14269 \catcode|=0%
14270 \catcode\=12%
14271 |catcode@=14%
14272 |catcode|=12@
14273 |gdef|markdownReadAndConvert#1#2{@
14274 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

14275 |markdownIfOption{frozenCache}{-}{@
14276 |immediate|openout|markdownOutputFileStream@
14277 |markdownOptionInputTempFileName|relax@
14278 |markdownInfo{@
14279 |Buffering block-level markdown input into the temporary @
14280 |input file "|markdownOptionInputTempFileName" and scanning @
14281 |for the closing token sequence "#1"}@
14282 }@

```

Locally change the category of the special plain  $\text{\TeX}$  characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

14283 |def|do##1{|catcode`##1=12}|dospecials@
14284 |catcode`|=12@
14285 |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (`%`) when `stripPercentSigns` is enabled. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

14286 |def|markdownReadAndConvertStripPercentSign##1{@
14287 |markdownIfOption{stripPercentSigns}{@
14288 |if##1%@
14289 |expandafter|expandafter|expandafter@

```

```

14290         |markdownReadAndConvertProcessLine@
14291     |else@
14292         |expandafter|expandafter|expandafter@
14293         |markdownReadAndConvertProcessLine@
14294         |expandafter|expandafter|expandafter##1@
14295     |fi@
14296 }{@
14297     |expandafter@
14298     |markdownReadAndConvertProcessLine@
14299     |expandafter##1@
14300 }@
14301 }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

14302     |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

14303     |ifx|relax##3|relax@
14304         |markdownIfOption{frozenCache}{-}{@
14305         |immediate|write|markdownOutputFileStream{##1}@
14306     }@
14307     |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.

```

14308     |def^^M{@
14309         |markdownInfo{The ending token sequence was found}@
14310         |markdownIfOption{frozenCache}{-}{@
14311         |immediate|closeout|markdownOutputFileStream@
14312     }@
14313     |endgroup@
14314     |markdownInput{@
14315         |markdownOptionOutputDir@
14316         /|markdownOptionInputTempFileName@
14317     }@
14318     #2}@
14319     |fi@

```

Repeat with the next line.

```

14320     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```
14321 |catcode`\^^I=13@
14322 |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.

```
14323 |catcode`\^^M=13@
14324 |def^^M##1^^M{@
14325 |def^^M###1^^M{@
14326 |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
14327 ^^M}@
14328 ^^M}@
```

Reset the character categories back to the former state.

```
14329 |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.

```
14330 \ExplSyntaxOn
14331 \cs_new:Npn
14332 \markdownLuaExecute
14333 #1
14334 {
14335 \int_compare:nNt
14336 { \g_luabridge_method_int }
14337 =
14338 { \c_luabridge_method_shell_int }
14339 {
14340 \sys_if_shell_unrestricted:F
14341 {
14342 \sys_if_shell:TF
14343 {
14344 \msg_error:nn
14345 { markdown }
14346 { restricted-shell-access }
14347 }
14348 {
14349 \msg_error:nn
14350 { markdown }
14351 { disabled-shell-access }
14352 }
14353 }
14354 }
14355 \str_gset:NV
14356 \g_luabridge_output_dirname_str
14357 \markdownOptionOutputDir
```



```

14358     \luabridge_now:e
14359     { #1 }
14360   }
14361 \cs_generate_variant:Nn
14362   \msg_new:nnnn
14363   { nnnV }
14364 \tl_set:Nn
14365   \l_tmpa_tl
14366   {
14367     You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
14368     --enable-write18~flag,~or~write~shell_escape=t~in~the~
14369     texmf.cnf~file.
14370   }
14371 \msg_new:nnnV
14372   { markdown }
14373   { restricted-shell-access }
14374   { Shell~escape~is~restricted }
14375   \l_tmpa_tl
14376 \msg_new:nnnV
14377   { markdown }
14378   { disabled-shell-access }
14379   { Shell~escape~is~disabled }
14380   \l_tmpa_tl
14381 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

14382 \ExplSyntaxOn
14383 \tl_new:N
14384   \g_@@_after_markinline_tl
14385 \tl_gset:Nn
14386   \g_@@_after_markinline_tl
14387   { \unskip }
14388 \cs_new:Npn
14389   \markinline
14390   {

```

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.

```

14391     \group_begin:
14392     \cctab_select:N
14393     \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

14394     \@@_if_option:nF

```

```

14395     { frozenCache }
14396     {
14397         \immediate
14398         \openout
14399         \markdownOutputFileStream
14400         \markdownOptionInputTempFileName
14401         \relax
14402         \msg_info:nne
14403         { markdown }
14404         { buffering-markinline }
14405         { \markdownOptionInputTempFileName }
14406     }

```

Peek ahead and extract the inline markdown text.

```

14407     \peek_regex_replace_once:nnF
14408     { { (.*) } }
14409     {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

14410         \c { @@_if_option:nF }
14411         \cB { frozenCache \cE }
14412         \cB {
14413             \c { immediate }
14414             \c { write }
14415             \c { markdownOutputFileStream }
14416             \cB { \1 \cE }
14417             \c { immediate }
14418             \c { closeout }
14419             \c { markdownOutputFileStream }
14420         \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

14421         \c { group_end: }
14422         \c { group_begin: }
14423         \c { @@_setup:n }
14424         \cB { contentLevel = inline \cE }
14425         \c { markdownInput }
14426         \cB {
14427             \c { markdownOptionOutputDir } /
14428             \c { markdownOptionInputTempFileName }
14429         \cE }
14430         \c { group_end: }
14431         \c { tl_use:N }
14432         \c { g_@@_after_markinline_tl }
14433     }
14434     {
14435         \msg_error:nn

```

```

14436         { markdown }
14437         { markinline-peek-failure }
14438     \group_end:
14439     \tl_use:N
14440         \g_@@_after_markinline_tl
14441     }
14442 }
14443 \msg_new:nnn
14444 { markdown }
14445 { buffering-markinline }
14446 { Buffering~inline~markdown~input~into~
14447   the~temporary~input~file~"#1". }
14448 \msg_new:nnnn
14449 { markdown }
14450 { markinline-peek-failure }
14451 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
14452 { The~macro~should~be~followed~by~inline~
14453   markdown~text~in~curly~braces }
14454 \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.

```

14455 \ExplSyntaxOn
14456 \cs_new:Npn
14457   \markdownInput
14458   #1
14459   {
14460     \@@_if_option:nTF
14461       { frozenCache }
14462       {
14463         \markdownInputRaw
14464         { #1 }
14465       }
14466     {

```

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`.

```

14467     \tl_set:Nx
14468     \l_tmpa_tl
14469     { #1 }
14470     \file_get_full_name:VNTF
14471     \l_tmpa_tl
14472     \l_tmpb_tl

```

```

14473     {
14474         \exp_args:NV
14475         \markdownInputRaw
14476         \l_tmpb_tl
14477     }
14478     {
14479         \msg_error:nnV
14480         { markdown }
14481         { markdown-file-does-not-exist }
14482         \l_tmpa_tl
14483     }
14484 }
14485 }
14486 \msg_new:nnn
14487 { markdown }
14488 { markdown-file-does-not-exist }
14489 {
14490     Markdown~file~#1~does~not~exist
14491 }
14492 \ExplSyntaxOff
14493 \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.

```

14494 \catcode`\|=0%
14495 \catcode`\|=12%
14496 \catcode`\&=6%
14497 \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.

```

14498 |begingroup
14499 |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.

```

14500 |catcode`\#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```

14501 |markdownIfOption{frozenCache}{%
14502 |ifnum|markdownOptionFrozenCacheCounter=0|relax
14503 |markdownInfo{Reading frozen cache from
14504     "|markdownOptionFrozenCacheFileName"%
14505 |input|markdownOptionFrozenCacheFileName|relax

```

```

14506     |fi
14507     |markdownInfo{Including markdown document number
14508         "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
14509     |csname markdownFrozenCache%
14510         |the|markdownOptionFrozenCacheCounter|endcsname
14511     |global|advance|markdownOptionFrozenCacheCounter by 1|relax
14512     }-%
14513     |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as `LATEX`Mk to track changes to the markdown document.

```

14514     |openin|markdownInputFileStream&1
14515     |closein|markdownInputFileStream
14516     |markdownPrepareLuaOptions
14517     |markdownPrepareInputFilename{&1}%
14518     |markdownLuaExecute{%
14519         |markdownPrepare
14520         |markdownConvert
14521         |markdownCleanup}%

```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```

14522     |markdownIfOption{finalizeCache}{%
14523         |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
14524     }%
14525     |endgroup
14526     }%
14527 |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.

```

14528 \gdef\markdownEscape#1{%
14529   \catcode`\%=14\relax
14530   \catcode`\#=6\relax
14531   \input #1\relax
14532   \catcode`\%=12\relax
14533   \catcode`\#=12\relax
14534 }%

```

### 3.3 L<sup>A</sup>T<sub>E</sub>X Implementation

The L<sup>A</sup>T<sub>E</sub>X implementation makes use of the fact that, apart from some subtle differences, L<sup>A</sup>T<sub>E</sub>X implements the majority of the plain T<sub>E</sub>X format [17, Section 9]. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation.

```

14535 \def\markdownVersionSpace{ }%

```

```

14536 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
14537 \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).

```

14538 \ExplSyntaxOn
14539 \cs_gset_eq:NN
14540 \markinlinePlainTeX
14541 \markinline
14542 \cs_gset:Npn
14543 \markinline
14544 {
14545 \peek_regex_replace_once:nn
14546 { ( \[ (.*) \] ) ? }
14547 {

```

Apply the options locally.

```

14548 \c { group_begin: }
14549 \c { @@_setup:n }
14550 \cB { \2 \cE }
14551 \c { tl_put_right:Nn }
14552 \c { g_@@_after_markinline_tl }
14553 \cB { \c { group_end: } \cE }
14554 \c { markinlinePlainTeX }
14555 }
14556 }
14557 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain  $\TeX$  implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the  $\LaTeX$  interface (see Section 2.3.3).

```

14558 \let\markdownInputPlainTeX\markdownInput
14559 \renewcommand\markdownInput [2] [] {%
14560 \begingroup
14561 \markdownSetup{#1}%
14562 \markdownInputPlainTeX{#2}%
14563 \endgroup}%
14564 \renewcommand\yamlInput [2] [] {%
14565 \begingroup
14566 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
14567 \markdownInputPlainTeX{#2}%
14568 \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```
14569 \ExplSyntaxOn
14570 \renewenvironment
14571   { markdown }
14572   {
```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

|                                               |                               |
|-----------------------------------------------|-------------------------------|
| <code>\begin{markdown} [smartEllipses]</code> | <code>\begin{markdown}</code> |
| <code>% This is an optional argument ^</code> | <code>[smartEllipses]</code>  |
| <code>% ...</code>                            | <code>% ^ This is link</code> |
| <code>\end{markdown}</code>                   | <code>\end{markdown}</code>   |

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
14573   \group_begin:
14574   \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).

```
14575   \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
14576   \peek_regex_replace_once:nnF
14577   { \ *[\r*([^\r]*)\][^\r]* }
14578   {
```

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.

```
14579     \c { group_end: }
14580     \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
14581     \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}
```

```
14582     \c { exp_args:NV }
14583     \c { markdownReadAndConvert@ }
14584     \c { @currenvir }
14585 }
14586 {
14587   \group_end:
14588   \exp_args:NV
14589   \markdownReadAndConvert@
14590   \@currenvir
14591 }
14592 }
14593 { \markdownEnd }
14594 \renewenvironment
14595 { markdown* }
14596 [ 1 ]
14597 {
14598   \@@_if_option:nTF
14599   { experimental }
14600   {
14601     \msg_error:nnn
14602     { markdown }
14603     { latex-markdown-star-deprecated }
14604     { #1 }
14605   }
14606   {
14607     \msg_warning:nnn
14608     { markdown }
14609     { latex-markdown-star-deprecated }
14610     { #1 }
```



```

14611     }
14612     \@@_setup:n
14613     { #1 }
14614     \markdownReadAndConvert@
14615     { markdown* }
14616   }
14617   { \markdownEnd }
14618 \renewenvironment
14619 { yaml }
14620 {
14621   \group_begin:
14622   \yamlSetup
14623     { jekyllData, expectJekyllData, ensureJekyllData }
14624   \markdown
14625 }
14626 { \yamlEnd }
14627 \msg_new:nnn
14628 { markdown }
14629 { latex-markdown-star-deprecated }
14630 {
14631   The~markdown*-LaTeX~environment~has~been~deprecated~and~will~
14632   be~removed~in~the~next~major~version~of~the~Markdown~package.
14633 }
14634 \cs_generate_variant:Nn
14635 \@@_setup:n
14636 { V }
14637 \ExplSyntaxOff
14638 \begingroup

```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```

14639   \catcode`\|=0\catcode`\<=1\catcode`\>=2%
14640   \catcode`\|=12\catcode`\{=12\catcode`\}=12%
14641   |gdef|markdownReadAndConvert@#1<%
14642     |markdownReadAndConvert<\end{#1}>%
14643     <|end<#1>>>%
14644 |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.

```

14645 \ExplSyntaxOn

```

```

14646 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
14647 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
14648 \cs_gset:Nn
14649   \@@_load_theme:nnn
14650   {

```

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.

```

14651   \ifmarkdownLaTeXLoaded
14652     \ifx\@onlypreamble\@notprerr

```

If both conditions are true, end with an error, since we cannot load L<sup>A</sup>T<sub>E</sub>X themes after the preamble.

```

14653       \bool_if:nTF
14654       {
14655         \bool_lazy_or_p:nn
14656         {
14657           \prop_if_in_p:Nn
14658             \g_@@_latex_built_in_themes_prop
14659             { #1 }
14660         }
14661         {
14662           \file_if_exist_p:n
14663             { markdown theme #3.sty }
14664         }
14665       }
14666       {
14667         \msg_error:nnn
14668           { markdown }
14669           { latex-theme-after-preamble }
14670           { #1 }
14671       }

```

Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

14672       {
14673         \@@_plain_tex_load_theme:nnn
14674         { #1 }
14675         { #2 }
14676         { #3 }
14677       }
14678   \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.

```

14679       \bool_if:nTF
14680       {

```

```

14681 \bool_lazy_or_p:nn
14682 {
14683   \prop_if_in_p:Nn
14684   \g_@@_latex_built_in_themes_prop
14685   { #1 }
14686 }
14687 {
14688   \file_if_exist_p:n
14689   { markdown theme #3.sty }
14690 }
14691 }
14692 {
14693   \prop_get:NnNTF
14694   \g_@@_latex_loaded_themes_linenos_prop
14695   { #1 }
14696   \l_tmpa_tl
14697   {
14698     \prop_get:NnN
14699     \g_@@_latex_loaded_themes_versions_prop
14700     { #1 }
14701     \l_tmpb_tl
14702     \str_if_eq:nVTF
14703     { #2 }
14704     \l_tmpb_tl
14705     {
14706       \msg_warning:nnnVn
14707       { markdown }
14708       { repeatedly-loaded-latex-theme }
14709       { #1 }
14710       \l_tmpa_tl
14711       { #2 }
14712     }
14713     {
14714       \msg_error:nnnnVV
14715       { markdown }
14716       { different-versions-of-latex-theme }
14717       { #1 }
14718       { #2 }
14719       \l_tmpb_tl
14720       \l_tmpa_tl
14721     }
14722   }
14723 }
14724 \prop_gput:Nnx
14725 \g_@@_latex_loaded_themes_linenos_prop
14726 { #1 }
14727 { \tex_the:D \tex_inputlineno:D } % noqa: W200

```

```

14728             \prop_gput:Nnn
14729             \g_@@_latex_loaded_themes_versions_prop
14730             { #1 }
14731             { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

14732             \prop_if_in:NnTF
14733             \g_@@_latex_built_in_themes_prop
14734             { #1 }
14735             {
14736               \msg_info:nxxx
14737               { markdown }
14738               { loading-built-in-latex-theme }
14739               { #1 }
14740               { #2 }
14741             \prop_item:Nn
14742             \g_@@_latex_built_in_themes_prop
14743             { #1 }
14744             }
14745             {
14746               \msg_info:nxxx
14747               { markdown }
14748               { loading-latex-theme }
14749               { #1 }
14750               { #2 }
14751             \RequirePackage
14752             { markdown theme #3 }
14753             }
14754             }
14755             }
14756             {
14757             \@@_plain_tex_load_theme:nnn
14758             { #1 }
14759             { #2 }
14760             { #3 }
14761             }
14762             \fi
14763             \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

14764             \msg_info:nxxx
14765             { markdown }
14766             { theme-loading-postponed }
14767             { #1 }
14768             { #2 }
14769             \AtEndOfPackage

```

```

14770     {
14771         \@_set_theme:n
14772         { #1 @ #2 }
14773     }
14774     \fi
14775 }
14776 \msg_new:nnn
14777 { markdown }
14778 { theme-loading-postponed }
14779 {
14780     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
14781     Markdown~package~has~finished~loading
14782 }
14783 \msg_new:nnn
14784 { markdown }
14785 { loading-built-in-latex-theme }
14786 { Loading~version~#2~of~built-in~LaTeX~Markdown~theme~#1 }
14787 \msg_new:nnn
14788 { markdown }
14789 { loading-latex-theme }
14790 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
14791 \msg_new:nnn
14792 { markdown }
14793 { repeatedly-loaded-latex-theme }
14794 {
14795     Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
14796     loaded~on~line~#2,~not~loading~it~again
14797 }
14798 \msg_new:nnn
14799 { markdown }
14800 { different-versions-of-latex-theme }
14801 {
14802     Tried~to~load~version~#2~of~LaTeX~Markdown~theme~#1~
14803     but~version~#3~has~already~been~loaded~on~line~#4
14804 }
14805 \cs_generate_variant:Nn
14806 \msg_new:nnnn
14807 { nnVV }
14808 \tl_set:Nn
14809 \l_tmpa_tl
14810 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
14811 \tl_put_right:NV
14812 \l_tmpa_tl
14813 \c_backslash_str
14814 \tl_put_right:Nn
14815 \l_tmpa_tl
14816 { begin { document } }

```

```

14817 \tl_set:Nn
14818   \l_tmpb_tl
14819   { Load-Markdown-theme~#1~before~ }
14820 \tl_put_right:NV
14821   \l_tmpb_tl
14822   \c_backslash_str
14823 \tl_put_right:Nn
14824   \l_tmpb_tl
14825   { begin { document } }
14826 \msg_new:nnVV
14827   { markdown }
14828   { latex-theme-after-preamble }
14829   \l_tmpa_tl
14830   \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.

```

14831 \tl_set:Nn
14832   \l_tmpa_tl
14833   {
14834     \RequirePackage
14835       { graphicx }
14836     \markdownLoadPlainTeXTheme
14837   }
14838 \prop_gput:NnV
14839   \g_@@_latex_built_in_themes_prop
14840   { witiko / dot }
14841   \l_tmpa_tl
14842 \prop_gput:NnV
14843   \g_@@_latex_built_in_themes_prop
14844   { witiko / graphicx / http }
14845   \l_tmpa_tl
14846 \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.

```
14847 \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.

```

14848 \DeclareOption*{%
14849   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
14850 \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.

```
14851 \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`.

```
14852 \ExplSyntaxOn
14853 \bool_new:N
14854   \g_@@_tight_or_fancy_lists_bool
14855 \bool_gset_false:N
14856   \g_@@_tight_or_fancy_lists_bool
14857 \@@_if_option:nTF
14858   { tightLists }
14859   {
14860     \bool_gset_true:N
14861       \g_@@_tight_or_fancy_lists_bool
14862   }
14863   {
14864     \@@_if_option:nT
14865     { fancyLists }
14866     {
14867       \bool_gset_true:N
14868         \g_@@_tight_or_fancy_lists_bool
14869     }
14870   }
14871 \bool_new:N
14872   \g_@@_beamer_paralist_or_enumitem_bool
14873 \bool_gset_true:N
14874   \g_@@_beamer_paralist_or_enumitem_bool
14875 \@ifclassloaded
14876   { beamer }
14877   { }
14878   {
14879     \@ifpackageloaded
14880     { paralist }
14881     { }
14882     {
14883       \@ifpackageloaded
14884       { enumitem }

```

```

14885     { }
14886     {
14887         \bool_gset_false:N
14888         \g_@@_beamer_paralist_or_enumitem_bool
14889     }
14890 }
14891 }
14892 \bool_if:nT
14893 {
14894     \g_@@_tight_or_fancy_lists_bool &&
14895     ! \g_@@_beamer_paralist_or_enumitem_bool
14896 }
14897 {
14898     \bool_if:nTF
14899     {
14900         \bool_lazy_or_p:nn
14901         {
14902             \str_if_eq_p:en
14903             { \markdownThemeVersion }
14904             { experimental }
14905         }
14906         {
14907             \bool_lazy_and_p:nn
14908             {
14909                 \prop_if_exist_p:N
14910                 \g__pdfmanagement_documentproperties_prop
14911             }
14912             {
14913                 \bool_lazy_any_p:n
14914                 {
14915                     {
14916                         \prop_if_in_p:Nn
14917                         \g__pdfmanagement_documentproperties_prop
14918                         { document / testphase / phase-I }
14919                     }
14920                     {
14921                         \prop_if_in_p:Nn
14922                         \g__pdfmanagement_documentproperties_prop
14923                         { document / testphase / phase-II }
14924                     }
14925                     {
14926                         \prop_if_in_p:Nn
14927                         \g__pdfmanagement_documentproperties_prop
14928                         { document / testphase / phase-III }
14929                     }
14930                 }
14931                 \prop_if_in_p:Nn

```



```

14932         \g__pdfmanagement_documentproperties_prop
14933         { document / testphase / phase-IV }
14934     }
14935     {
14936         \prop_if_in_p:Nn
14937         \g__pdfmanagement_documentproperties_prop
14938         { document / testphase / phase-V }
14939     }
14940     {
14941         \prop_if_in_p:Nn
14942         \g__pdfmanagement_documentproperties_prop
14943         { document / testphase / phase-VI }
14944     }
14945 }
14946 }
14947 }
14948 }
14949 {
14950     \RequirePackage
14951     { enumitem }
14952 }
14953 {
14954     \RequirePackage
14955     { paralist }
14956 }
14957 }
14958 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

14959 \ExplSyntaxOn
14960 \cs_new:Nn
14961 \@@_latex_fancy_list_item_label_number:nn
14962 {
14963     \str_case:nn
14964     { #1 }
14965     {
14966         { Decimal } { #2 }
14967         { LowerRoman } { \int_to_roman:n { #2 } }
14968         { UpperRoman } { \int_to_Roman:n { #2 } }
14969         { LowerAlpha } { \int_to_alph:n { #2 } }
14970         { UpperAlpha } { \int_to_Alph:n { #2 } }
14971     }
14972 }
14973 \cs_new:Nn
14974 \@@_latex_fancy_list_item_label_delimiter:n
14975 {

```

```

14976 \str_case:nn
14977   { #1 }
14978   {
14979     { Default } { . }
14980     { OneParen } { ) }
14981     { Period } { . }
14982   }
14983 }
14984 \cs_new:Nn
14985 \@@_latex_fancy_list_item_label:nnn
14986 {
14987   \@@_latex_fancy_list_item_label_number:nn
14988     { #1 }
14989     { #3 }
14990   \@@_latex_fancy_list_item_label_delimiter:n
14991     { #2 }
14992 }
14993 \cs_generate_variant:Nn
14994 \@@_latex_fancy_list_item_label:nnn
14995 { VVn }
14996 \tl_new:N
14997 \l_@@_latex_fancy_list_item_label_number_style_tl
14998 \tl_new:N
14999 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15000 \ifpackageloaded { enumitem } {
15001   \markdownSetup { rendererPrototypes = {

```

First, let's define the tight list item renderer prototypes.

```

15002   ulBeginTight = {
15003     \begin
15004       { itemize }
15005       [ noitemsep ]
15006   },
15007   ulEndTight = {
15008     \end
15009     { itemize }
15010   },
15011   olBeginTight = {
15012     \begin
15013       { enumerate }
15014       [ noitemsep ]
15015   },
15016   olEndTight = {
15017     \end
15018     { enumerate }
15019   },
15020   dlBeginTight = {
15021     \begin

```

```

15022     { description }
15023     [ noitemsep ]
15024 },
15025 dlEndTight = {
15026     \end
15027     { description }
15028 },

```

Second, let's define the fancy list item renderer prototypes.

```

15029 fancyOlBegin = {
15030     \group_begin:
15031     \tl_set:Nn
15032     \l_@@_latex_fancy_list_item_label_number_style_tl
15033     { #1 }
15034     \tl_set:Nn
15035     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15036     { #2 }
15037     \begin
15038     { enumerate }
15039 },
15040 fancyOlBeginTight = {
15041     \group_begin:
15042     \tl_set:Nn
15043     \l_@@_latex_fancy_list_item_label_number_style_tl
15044     { #1 }
15045     \tl_set:Nn
15046     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15047     { #2 }
15048     \begin
15049     { enumerate }
15050     [ noitemsep ]
15051 },
15052 fancyOlEnd(|Tight) = {
15053     \end { enumerate }
15054     \group_end:
15055 },
15056 fancyOlItemWithNumber = {
15057     \item
15058     [
15059         \@@_latex_fancy_list_item_label:VVn
15060         \l_@@_latex_fancy_list_item_label_number_style_tl
15061         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15062         { #1 }
15063     ]
15064 },
15065 } }

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```
15066 }
15067 { \@ifpackageloaded { paralist } {
15068   \markdownSetup { rendererPrototypes = {
```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```
15069     ulBeginTight = {
15070       \group_begin:
15071       \pltopsep=\topsep
15072       \plpartopsep=\partopsep
15073       \begin { compactitem }
15074     },
15075     ulEndTight = {
15076       \end { compactitem }
15077       \group_end:
15078     },
15079     fancyOlBegin = {
15080       \group_begin:
15081       \tl_set:Nn
15082         \l_@@_latex_fancy_list_item_label_number_style_tl
15083         { #1 }
15084       \tl_set:Nn
15085         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15086         { #2 }
15087       \begin { enumerate }
15088     },
15089     fancyOlEnd = {
15090       \end { enumerate }
15091       \group_end:
15092     },
```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```
15093     olBeginTight = {
15094       \group_begin:
15095       \plpartopsep=\partopsep
15096       \pltopsep=\topsep
15097       \begin { compactenum }
15098     },
15099     olEndTight = {
15100       \end { compactenum }
15101       \group_end:
15102     },
15103     fancyOlBeginTight = {
15104       \group_begin:
```

```

15105     \tl_set:Nn
15106         \l_@@_latex_fancy_list_item_label_number_style_tl
15107         { #1 }
15108     \tl_set:Nn
15109         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15110         { #2 }
15111     \plpartopsep=\partopsep
15112     \pltopsep=\topsep
15113     \begin { compactenum }
15114 },
15115 fancyOlEndTight = {
15116     \end { compactenum }
15117     \group_end:
15118 },
15119 fancyOlItemWithNumber = {
15120     \item
15121     [
15122         \@@_latex_fancy_list_item_label:VVn
15123         \l_@@_latex_fancy_list_item_label_number_style_tl
15124         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
15125         { #1 }
15126     ]
15127 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

15128     dlBeginTight = {
15129         \group_begin:
15130         \plpartopsep=\partopsep
15131         \pltopsep=\topsep
15132         \begin { compactdesc }
15133     },
15134     dlEndTight = {
15135         \end { compactdesc }
15136         \group_end:
15137     }
15138 } }
15139 }
15140 {

```

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.

```

15141     \markdownSetup
15142     {
15143         rendererPrototypes = {
15144             ulBeginTight = \markdownRendererUlBegin,
15145             ulEndTight = \markdownRendererUlEnd,

```

```

15146     fancyO1Begin = \markdownRendererO1Begin,
15147     fancyO1End = \markdownRendererO1End,
15148     olBeginTight = \markdownRendererO1Begin,
15149     olEndTight = \markdownRendererO1End,
15150     fancyO1BeginTight = \markdownRendererO1Begin,
15151     fancyO1EndTight = \markdownRendererO1End,
15152     dlBeginTight = \markdownRendererDlBegin,
15153     dlEndTight = \markdownRendererDlEnd,
15154   },
15155 }
15156 } }
15157 \ExplSyntaxOff
15158 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

15159 \@ifpackageloaded{unicode-math}{
15160   \markdownSetup{rendererPrototypes={
15161     untickedBox = {\$ \mdlgwhtsquare$},
15162   }}
15163 }{
15164   \RequirePackage{amssymb}
15165   \markdownSetup{rendererPrototypes={
15166     untickedBox = {\$ \square$},
15167   }}
15168 }
15169 \RequirePackage{csvsimple}
15170 \RequirePackage{fancyvrb}
15171 \RequirePackage{graphicx}
15172 \markdownSetup{rendererPrototypes={
15173   hardLineBreak = {\},
15174   leftBrace = {\textbraceleft},
15175   rightBrace = {\textbraceright},
15176   dollarSign = {\textdollar},
15177   underscore = {\textunderscore},
15178   circumflex = {\textasciicircum},
15179   backslash = {\textbackslash},
15180   tilde = {\textasciitilde},
15181   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by  $\TeX$  during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</sup> we can reliably detect math mode inside the renderer.

---

<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.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

15182 codeSpan = {%
15183   \ifmmode
15184     \text{#1}%
15185   \else
15186     \texttt{#1}%
15187   \fi
15188   }}

```

### 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.

```

15189 \ExplSyntaxOn
15190 \markdownSetup{
15191   rendererPrototypes = {
15192     contentBlock = {
15193       \str_case:nnF
15194         { #1 }
15195         {
15196           { csv }
15197           {
15198             \begin { table }
15199             \begin { center }
15200               \csvautotabular { #3 }
15201             \end{ center }
15202             \tl_if_empty:nF
15203               { #4 }
15204               { \caption { #4 } }
15205             \end { table }
15206           }
15207           { html }
15208           {

```

If we are using `TeX4ht`<sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

15209       \cs_if_exist:NTF
15210         \HCode
15211         {
15212           \if_mode_vertical:
15213             \IgnorePar

```

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

```

15214         \fi:
15215         \EndP
15216         \special
15217           { t4ht* < #3 }
15218         \par
15219         \ShowPar
15220       }
15221     {
15222       \@_luaxml_print_html:n
15223       { #3 }
15224     }
15225   }
15226   { tex }
15227   {
15228     \markdownEscape
15229     { #3 }
15230   }
15231 }
15232 {
15233   \markdownInput
15234   { #3 }
15235 }
15236 },
15237 },
15238 }
15239 \ExplSyntaxOff
15240 \markdownSetup{rendererPrototypes={
15241   ulBegin = {\begin{itemize}},
15242   ulEnd = {\end{itemize}},
15243   olBegin = {\begin{enumerate}},
15244   olItem = {\item{}},
15245   olItemWithNumber = {\item[#1.]},
15246   olEnd = {\end{enumerate}},
15247   dlBegin = {\begin{description}},
15248   dlItem = {\item[#1]},
15249   dlEnd = {\end{description}},
15250   emphasis = {\emph{#1}},
15251   tickedBox = {\$\boxtimes$},
15252   halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

15253 \ExplSyntaxOn
15254 \seq_new:N
15255   \g_@@_header_identifiers_seq
15256 \markdownSetup
15257 {
15258   rendererPrototypes = {
15259     headerAttributeContextBegin = {

```



```

15260     \markdownSetup
15261     {
15262         rendererPrototypes = {
15263             attributeIdentifier = {
15264                 \seq_gput_right:Nn
15265                 \g_@@_header_identifiers_seq
15266                 { ##1 }
15267             },
15268         },
15269     }
15270 },
15271 headerAttributeContextEnd = {
15272     \seq_map_inline:Nn
15273     \g_@@_header_identifiers_seq
15274     { \label { ##1 } }
15275     \seq_gclear:N
15276     \g_@@_header_identifiers_seq
15277 },
15278 },
15279 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

15280 \bool_new:N
15281 \l_@@_header_unnumbered_bool
15282 \markdownSetup
15283 {
15284     rendererPrototypes = {
15285         headerAttributeContextBegin += {
15286             \markdownSetup
15287             {
15288                 rendererPrototypes = {
15289                     attributeClassName = {
15290                         \bool_if:nT
15291                         {
15292                             \str_if_eq_p:nn
15293                             { ##1 }
15294                             { unnumbered } &&
15295                             ! \l_@@_header_unnumbered_bool
15296                         }
15297                     {
15298                         \group_begin:
15299                         \bool_set_true:N
15300                         \l_@@_header_unnumbered_bool
15301                         \c@secnumdepth = 0
15302                         \markdownSetup
15303                         {

```

```

15304             rendererPrototypes = {
15305                 sectionBegin = {
15306                     \group_begin:
15307                 },
15308                 sectionEnd = {
15309                     \group_end:
15310                 },
15311             },
15312         },
15313     },
15314 },
15315 },
15316 }
15317 },
15318 },
15319 }
15320 \ExplSyntaxOff
15321 \markdownSetup{rendererPrototypes={
15322     superscript = {\textsuperscript{#1}},
15323     subscript = {\textsubscript{#1}},
15324     blockQuoteBegin = {\begin{quotation}},
15325     blockQuoteEnd = {\end{quotation}},
15326     inputVerbatim = {\VerbatimInput{#1}},
15327     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
15328     note = {\footnote{#1}}}}

```

### 3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

15329 \RequirePackage{ltxcmds}
15330 \ExplSyntaxOn
15331 \cs_gset_protected:Npn
15332   \markdownRendererInputFencedCodePrototype#1#2#3
15333   {
15334     \tl_if_empty:nTF
15335       { #2 }
15336     { \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.

```

15337     {
15338       \regex_extract_once:nnN
15339         { \w* }
15340         { #2 }
15341         \l_tmpa_seq
15342       \seq_pop_left:NN
15343         \l_tmpa_seq
15344         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```
15345     \ltx@ifpackageloaded
15346     { minted }
15347     {
15348         \catcode`\%=14\relax
15349         \catcode`\#=6\relax
15350         \exp_args:NV
15351         \inputminted
15352         \l_tmpa_tl
15353         { #1 }
15354         \catcode`\%=12\relax
15355         \catcode`\#=12\relax
15356     }
15357     {
```

When the listings package is loaded, use it for syntax highlighting.

```
15358     \ltx@ifpackageloaded
15359     { listings }
15360     { \lstinputlisting [ language = \l_tmpa_tl ] { #1 } }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
15361         { \markdownRendererInputFencedCode { #1 } { } { } }
15362     }
15363 }
15364 }
15365 \ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
15366 \ExplSyntaxOn
15367 \def\markdownLATEXStrongEmphasis#1{
15368   \str_if_in:NnTF
15369     \f@series
15370     { b }
15371     { \textnormal{#1} }
15372     { \textbf{#1} }
15373 }
15374 \ExplSyntaxOff
15375 \markdownSetup{rendererPrototypes={strongEmphasis={%
15376   \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support L<sup>A</sup>T<sub>E</sub>X document classes that do not provide chapters.

```
15377 \@ifundefined{chapter}{%
15378   \markdownSetup{rendererPrototypes = {
15379     headingOne = {\section{#1}},
15380     headingTwo = {\subsection{#1}},
15381     headingThree = {\subsubsection{#1}},
15382     headingFour = {\paragraph{#1}},
```

```

15383     headingFive = {\subparagraph{#1}}}}
15384 }{%
15385   \markdownSetup{rendererPrototypes = {
15386     headingOne = {\chapter{#1}},
15387     headingTwo = {\section{#1}},
15388     headingThree = {\subsection{#1}},
15389     headingFour = {\subsubsection{#1}},
15390     headingFive = {\paragraph{#1}},
15391     headingSix = {\subparagraph{#1}}}}
15392 }%

```

#### 3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

15393 \markdownSetup{
15394   rendererPrototypes = {
15395     ulItem = {%
15396       \futurelet\markdownLaTeXCheckbox\markdownLaTeXULItem
15397     },
15398   },
15399 }
15400 \def\markdownLaTeXULItem{%
15401   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
15402     \item[\markdownLaTeXCheckbox]%
15403     \expandafter\@gobble
15404   \else
15405     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
15406       \item[\markdownLaTeXCheckbox]%
15407       \expandafter\expandafter\expandafter\@gobble
15408     \else
15409       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
15410         \item[\markdownLaTeXCheckbox]%
15411         \expandafter\expandafter\expandafter\expandafter
15412         \expandafter\expandafter\expandafter\@gobble
15413       \else
15414         \item{}%
15415       \fi
15416     \fi
15417   \fi
15418 }

```

#### 3.3.4.5 HTML elements

If the `html` option is enabled and we are using  $\text{T}_{\text{E}}\text{X}4\text{ht}$ <sup>36</sup>, we will pass HTML elements to the output HTML document unchanged.

---

<sup>36</sup>See <https://tug.org/tex4ht/>.

```

15419 \@ifundefined{HCode}{-}{
15420   \markdownSetup{
15421     rendererPrototypes = {
15422       inlineHtmlTag = {%
15423         \ifvmode
15424           \IgnorePar
15425         \EndP
15426         \fi
15427         \HCode{#1}%
15428       },
15429       inputBlockHtmlElement = {%
15430         \ifvmode
15431           \IgnorePar
15432         \fi
15433         \EndP
15434         \special{t4ht*<#1}%
15435         \par
15436         \ShowPar
15437       },
15438     },
15439   }
15440 }

```

### 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.

```

15441 \newcount\markdownLaTeXCitationsCounter
15442
15443 % Basic implementation
15444 \long\def@gobblethree#1#2#3{}%
15445 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
15446   \advance\markdownLaTeXCitationsCounter by 1\relax
15447   \ifx\relax#4\relax
15448     \ifx\relax#5\relax
15449       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15450         \relax
15451         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
15452         \expandafter\expandafter\expandafter
15453         \expandafter\expandafter\expandafter\expandafter
15454         \@gobblethree
15455       \fi
15456     \else% Before a postnote (#5), dump the accumulator
15457       \ifx\relax#1\relax\else
15458         \cite{#1}%

```

```

15459     \fi
15460     \cite[#5]{#6}%
15461     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15462     \relax
15463     \else
15464         \expandafter\expandafter\expandafter
15465         \expandafter\expandafter\expandafter\expandafter
15466         \expandafter\expandafter\expandafter
15467         \expandafter\expandafter\expandafter\expandafter
15468         \markdownLaTeXBasicCitations
15469     \fi
15470     \expandafter\expandafter\expandafter
15471     \expandafter\expandafter\expandafter\expandafter{%
15472     \expandafter\expandafter\expandafter
15473     \expandafter\expandafter\expandafter\expandafter}%
15474     \expandafter\expandafter\expandafter
15475     \expandafter\expandafter\expandafter\expandafter{%
15476     \expandafter\expandafter\expandafter
15477     \expandafter\expandafter\expandafter\expandafter}%
15478     \expandafter\expandafter\expandafter
15479     \@gobblethree
15480     \fi
15481 \else% Before a prenote (#4), dump the accumulator
15482     \ifx\relax#1\relax\else
15483         \cite{#1}%
15484     \fi
15485     \ifnum\markdownLaTeXCitationsCounter>1\relax
15486         \space % Insert a space before the prenote in later citations
15487     \fi
15488     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
15489     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15490     \relax
15491     \else
15492         \expandafter\expandafter\expandafter
15493         \expandafter\expandafter\expandafter\expandafter
15494         \markdownLaTeXBasicCitations
15495     \fi
15496     \expandafter\expandafter\expandafter{%
15497     \expandafter\expandafter\expandafter}%
15498     \expandafter\expandafter\expandafter{%
15499     \expandafter\expandafter\expandafter}%
15500     \expandafter
15501     \@gobblethree
15502     \fi\markdownLaTeXBasicCitations{#1#2#6},}
15503 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
15504
15505 % Natbib implementation

```

```

15506 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
15507 \advance\markdownLaTeXCitationsCounter by 1\relax
15508 \ifx\relax#3\relax
15509 \ifx\relax#4\relax
15510 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15511 \relax
15512 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
15513 \expandafter\expandafter\expandafter
15514 \expandafter\expandafter\expandafter\expandafter
15515 \@gobbletwo
15516 \fi
15517 \else% Before a postnote (#4), dump the accumulator
15518 \ifx\relax#1\relax\else
15519 \citep{#1}%
15520 \fi
15521 \citep[] [#4]{#5}%
15522 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15523 \relax
15524 \else
15525 \expandafter\expandafter\expandafter
15526 \expandafter\expandafter\expandafter\expandafter
15527 \expandafter\expandafter\expandafter
15528 \expandafter\expandafter\expandafter\expandafter
15529 \markdownLaTeXNatbibCitations
15530 \fi
15531 \expandafter\expandafter\expandafter
15532 \expandafter\expandafter\expandafter\expandafter{%
15533 \expandafter\expandafter\expandafter
15534 \expandafter\expandafter\expandafter\expandafter}%
15535 \expandafter\expandafter\expandafter
15536 \@gobbletwo
15537 \fi
15538 \else% Before a prenote (#3), dump the accumulator
15539 \ifx\relax#1\relax\relax\else
15540 \citep{#1}%
15541 \fi
15542 \citep[#3] [#4]{#5}%
15543 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15544 \relax
15545 \else
15546 \expandafter\expandafter\expandafter
15547 \expandafter\expandafter\expandafter\expandafter
15548 \markdownLaTeXNatbibCitations
15549 \fi
15550 \expandafter\expandafter\expandafter{%
15551 \expandafter\expandafter\expandafter}%
15552 \expandafter

```

```

15553     \@gobbletwo
15554     \fi\markdownLaTeXNatbibCitations{#1,#5}}
15555 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
15556     \advance\markdownLaTeXCitationsCounter by 1\relax
15557     \ifx\relax#3\relax
15558         \ifx\relax#4\relax
15559             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15560                 \relax
15561                 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
15562                 \expandafter\expandafter\expandafter
15563                 \expandafter\expandafter\expandafter\expandafter
15564                 \@gobbletwo
15565             \fi
15566         \else% After a prenote or a postnote, dump the accumulator
15567             \ifx\relax#1\relax\else
15568                 \citet{#1}%
15569             \fi
15570             , \citet[#3][#4]{#5}%
15571             \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15572                 \relax
15573             ,
15574             \else
15575                 \ifnum
15576                     \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15577                 \relax
15578             ,
15579             \fi
15580         \fi
15581         \expandafter\expandafter\expandafter
15582         \expandafter\expandafter\expandafter\expandafter
15583         \markdownLaTeXNatbibTextCitations
15584         \expandafter\expandafter\expandafter
15585         \expandafter\expandafter\expandafter\expandafter{%
15586         \expandafter\expandafter\expandafter
15587         \expandafter\expandafter\expandafter\expandafter}%
15588         \expandafter\expandafter\expandafter
15589         \@gobbletwo
15590     \fi
15591 \else% After a prenote or a postnote, dump the accumulator
15592     \ifx\relax#1\relax\relax\else
15593         \citet{#1}%
15594     \fi
15595     , \citet[#3][#4]{#5}%
15596     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15597         \relax
15598     ,
15599     \else

```



```

15600     \ifnum
15601     \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15602     \relax
15603     ,
15604     \fi
15605     \fi
15606     \expandafter\expandafter\expandafter
15607     \markdownLaTeXNatbibTextCitations
15608     \expandafter\expandafter\expandafter{%
15609     \expandafter\expandafter\expandafter}%
15610     \expandafter
15611     \@gobbletwo
15612     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
15613
15614 % BibLaTeX implementation
15615 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
15616 \advance\markdownLaTeXCitationsCounter by 1\relax
15617 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15618 \relax
15619 \autocites#1[#3][#4]{#5}%
15620 \expandafter\@gobbletwo
15621 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
15622 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
15623 \advance\markdownLaTeXCitationsCounter by 1\relax
15624 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15625 \relax
15626 \textcites#1[#3][#4]{#5}%
15627 \expandafter\@gobbletwo
15628 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
15629
15630 \markdownSetup{rendererPrototypes = {
15631 cite = {%
15632 \markdownLaTeXCitationsCounter=1%
15633 \def\markdownLaTeXCitationsTotal{#1}%
15634 \@ifundefined{autocites}{%
15635 \@ifundefined{citep}{%
15636 \expandafter\expandafter\expandafter
15637 \markdownLaTeXBasicCitations
15638 \expandafter\expandafter\expandafter{%
15639 \expandafter\expandafter\expandafter}%
15640 \expandafter\expandafter\expandafter{%
15641 \expandafter\expandafter\expandafter}%
15642 }{%
15643 \expandafter\expandafter\expandafter
15644 \markdownLaTeXNatbibCitations
15645 \expandafter\expandafter\expandafter{%
15646 \expandafter\expandafter\expandafter}%

```

```

15647     }%
15648   }{%
15649     \expandafter\expandafter\expandafter
15650     \markdownLaTeXBibLaTeXCitations
15651     \expandafter{\expandafter}%
15652   }},
15653   textCite = {%
15654     \markdownLaTeXCitationsCounter=1%
15655     \def\markdownLaTeXCitationsTotal{#1}%
15656     \@ifundefined{autocites}{%
15657       \@ifundefined{citep}{%
15658         \expandafter\expandafter\expandafter
15659         \markdownLaTeXBasicTextCitations
15660         \expandafter\expandafter\expandafter{%
15661         \expandafter\expandafter\expandafter}%
15662         \expandafter\expandafter\expandafter{%
15663         \expandafter\expandafter\expandafter}%
15664       }{%
15665         \expandafter\expandafter\expandafter
15666         \markdownLaTeXNatbibTextCitations
15667         \expandafter\expandafter\expandafter{%
15668         \expandafter\expandafter\expandafter}%
15669       }%
15670     }{%
15671       \expandafter\expandafter\expandafter
15672       \markdownLaTeXBibLaTeXTextCitations
15673       \expandafter{\expandafter}%
15674     }}}

```

### 3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

15675 \RequirePackage{url}
15676 \RequirePackage{expl3}
15677 \ExplSyntaxOn
15678 \cs_gset_protected:Npn
15679   \markdownRendererLinkPrototype
15680   #1#2#3#4
15681   {
15682     \tl_set:Nn \l_tmpa_tl { #1 }
15683     \tl_set:Nn \l_tmpb_tl { #2 }
15684     \bool_set:Nn
15685       \l_tmpa_bool
15686       {
15687         \tl_if_eq_p:NN
15688           \l_tmpa_tl
15689           \l_tmpb_tl

```

```

15690     }
15691     \tl_set:Nn \l_tmpa_tl { #4 }
15692     \bool_set:Nn
15693       \l_tmpb_bool
15694       {
15695         \tl_if_empty_p:N
15696           \l_tmpa_tl
15697       }

```

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.

```

15698     \bool_if:nTF
15699       {
15700         \l_tmpa_bool && \l_tmpb_bool
15701       }
15702       {
15703         \markdownLaTeXRendererAutolink { #2 } { #3 }
15704       }
15705       {
15706         \markdownLaTeXRendererDirectOrIndirectLink
15707           { #1 } { #2 } { #3 } { #4 }
15708       }
15709     }
15710 \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.

```

15711     \tl_set:Nn
15712       \l_tmpa_tl
15713       { #2 }
15714     \tl_trim_spaces:N
15715       \l_tmpa_tl
15716     \tl_set:Nx
15717       \l_tmpb_tl
15718       {
15719         \tl_range:Nnn
15720           \l_tmpa_tl
15721           { 1 }
15722           { 1 }
15723       }
15724     \str_if_eq:NNTF
15725       \l_tmpb_tl
15726       \c_hash_str
15727       {
15728         \tl_set:Nx
15729           \l_tmpb_tl
15730           {

```

```

15731         \tl_range:Nnn
15732         \l_tmpa_tl
15733         { 2 }
15734         { -1 }
15735     }
15736     \exp_args:NV
15737     \ref
15738     \l_tmpb_tl
15739 }
15740 {
15741     \url { #2 }
15742 }
15743 }
15744 \ExplSyntaxOff
15745 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
15746     #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.

```

15747 \newcount\markdownLaTeXRowCount
15748 \newcount\markdownLaTeXRowTotal
15749 \newcount\markdownLaTeXColumnCounter
15750 \newcount\markdownLaTeXColumnTotal
15751 \newtoks\markdownLaTeXTable
15752 \newtoks\markdownLaTeXTableAlignment
15753 \newtoks\markdownLaTeXTableEnd
15754 \AtBeginDocument{%
15755     \@ifpackageloaded{booktabs}{%
15756         \def\markdownLaTeXTopRule{\toprule}%
15757         \def\markdownLaTeXMidRule{\midrule}%
15758         \def\markdownLaTeXBottomRule{\bottomrule}%
15759     }{%
15760         \def\markdownLaTeXTopRule{\hline}%
15761         \def\markdownLaTeXMidRule{\hline}%
15762         \def\markdownLaTeXBottomRule{\hline}%
15763     }%
15764 }
15765 \markdownSetup{rendererPrototypes={
15766     table = {%
15767         \markdownLaTeXTable={}%
15768         \markdownLaTeXTableAlignment={}%
15769         \markdownLaTeXTableEnd={%
15770             \markdownLaTeXBottomRule
15771             \end{tabular}}%
15772         \ifx\empty#1\empty\else

```

```

15773     \addto@hook\markdownLaTeXTable{%
15774         \begin{table}
15775         \centering}%
15776     \addto@hook\markdownLaTeXTableEnd{%
15777         \caption{#1}}%
15778     \fi
15779 }
15780 }}

```

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.

```

15781 \ExplSyntaxOn
15782 \seq_new:N
15783   \l_@@_table_identifiers_seq
15784 \markdownSetup {
15785   rendererPrototypes = {
15786     table += {
15787       \seq_map_inline:Nn
15788         \l_@@_table_identifiers_seq
15789         {
15790           \addto@hook
15791             \markdownLaTeXTableEnd
15792             { \label { ##1 } }
15793         }
15794     },
15795   }
15796 }
15797 \markdownSetup {
15798   rendererPrototypes = {
15799     tableAttributeContextBegin = {
15800       \group_begin:
15801       \markdownSetup {
15802         rendererPrototypes = {
15803           attributeIdentifier = {
15804             \seq_put_right:Nn
15805               \l_@@_table_identifiers_seq
15806               { ##1 }
15807           },
15808         },
15809       }
15810     },
15811     tableAttributeContextEnd = {
15812       \group_end:
15813     },
15814   },
15815 }
15816 \ExplSyntaxOff

```

```

15817 \markdownSetup{rendererPrototypes={
15818   table += {%
15819     \ifx\empty#1\empty\else
15820       \addto@hook\markdownLaTeXTableEnd{%
15821         \end{table}}%
15822     \fi
15823     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
15824     \markdownLaTeXRowCounter=0%
15825     \markdownLaTeXRowTotal=#2%
15826     \markdownLaTeXColumnTotal=#3%
15827     \markdownLaTeXRenderTableRow
15828   }
15829 }}
15830 \def\markdownLaTeXRenderTableRow#1{%
15831   \markdownLaTeXColumnCounter=0%
15832   \ifnum\markdownLaTeXRowCounter=0\relax
15833     \markdownLaTeXReadAlignments#1%
15834     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
15835       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
15836         \the\markdownLaTeXTableAlignment}}%
15837     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
15838   \else
15839     \markdownLaTeXRenderTableCell#1%
15840   \fi
15841   \ifnum\markdownLaTeXRowCounter=1\relax
15842     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
15843   \fi
15844   \advance\markdownLaTeXRowCounter by 1\relax
15845   \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
15846     \the\markdownLaTeXTable
15847     \the\markdownLaTeXTableEnd
15848     \expandafter\@gobble
15849   \fi\markdownLaTeXRenderTableRow}
15850 \def\markdownLaTeXReadAlignments#1{%
15851   \advance\markdownLaTeXColumnCounter by 1\relax
15852   \if#1d%
15853     \addto@hook\markdownLaTeXTableAlignment{1}%
15854   \else
15855     \addto@hook\markdownLaTeXTableAlignment{#1}%
15856   \fi
15857   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
15858     \expandafter\@gobble
15859   \fi\markdownLaTeXReadAlignments}
15860 \def\markdownLaTeXRenderTableCell#1{%
15861   \advance\markdownLaTeXColumnCounter by 1\relax
15862   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
15863     \addto@hook\markdownLaTeXTable{#1&}%

```

```

15864 \else
15865   \addto@hook\markdownLaTeXTable{#1\}%
15866   \expandafter\@gobble
15867 \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.

```

15868
15869 \markdownIfOption{lineBlocks}{%
15870   \RequirePackage{verse}
15871   \markdownSetup{rendererPrototypes={
15872     lineBlockBegin = {%
15873       \begingroup
15874       \def\markdownRendererHardLineBreak{\}%
15875       \begin{verse}%
15876     },
15877     lineBlockEnd = {%
15878       \end{verse}%
15879     \endgroup
15880   },
15881   }}
15882 }{}
15883

```

### 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.

```

15884 \ExplSyntaxOn
15885 \keys_define:nn
15886 { markdown / jekyllData }
15887 {
15888   author .code:n = {
15889     \author
15890     { #1 }
15891   },
15892   date .code:n = {
15893     \date
15894     { #1 }
15895   },
15896   title .code:n = {
15897     \title
15898     { #1 }

```

To complement the default setup of our key-values, we will use the `\maketitle` macro to typeset the title page of a document at the end of YAML metadata. If we are in the preamble, we will wait macro until after the beginning of the document. Otherwise, we will use the `\maketitle` macro straight away.

```

15899     \AddToHook
15900         { begindocument / end }
15901         { \maketitle }
15902     },
15903 }

```

### 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.

```

15904 \@@_if_option:nT
15905 { mark }
15906 {
15907     \sys_if_engine_luatex:TF
15908     {
15909         \RequirePackage
15910         { luacolor }
15911         \RequirePackage
15912         { lua-ul }
15913         \markdownSetup
15914         {
15915             rendererPrototypes = {
15916                 mark = {
15917                     \highLight
15918                     { #1 }
15919                 },
15920             }
15921         }
15922     }
15923     {
15924         \RequirePackage
15925         { xcolor }
15926         \RequirePackage
15927         { soul }
15928         \markdownSetup
15929         {
15930             rendererPrototypes = {
15931                 mark = {
15932                     \hl
15933                     { #1 }
15934                 },
15935             }
15936         }

```



```

15937     }
15938 }

```

### 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.

```

15939 \@@_if_option:nT
15940 { strikeThrough }
15941 {
15942   \sys_if_engine_luatex:TF
15943   {
15944     \RequirePackage
15945     { lua-ul }
15946     \markdownSetup
15947     {
15948       rendererPrototypes = {
15949         strikeThrough = {
15950           \strikeThrough
15951           { #1 }
15952         },
15953       }
15954     }
15955   }
15956   {
15957     \RequirePackage
15958     { soul }
15959     \markdownSetup
15960     {
15961       rendererPrototypes = {
15962         strikeThrough = {
15963           \st
15964           { #1 }
15965         },
15966       }
15967     }
15968   }
15969 }

```

### 3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form  $\langle key \rangle = \langle value \rangle$  set the corresponding keys of the `graphicx` package to the corresponding

values and we will register any identifiers, so that they can be used as L<sup>A</sup>T<sub>E</sub>X labels for referencing figures.

```

15970 \seq_new:N
15971   \l_@@_image_identifiers_seq
15972 \markdownSetup {
15973   rendererPrototypes = {
15974     image = {
15975       \tl_if_empty:nTF
15976         { #4 }
15977         {
15978           \begin { center }
15979             \includegraphics
15980               [ alt = { #1 } ]
15981               { #3 }
15982           \end { center }
15983         }
15984       {
15985         \begin { figure }
15986           \begin { center }
15987             \includegraphics
15988               [ alt = { #1 } ]
15989               { #3 }
15990             \caption { #4 }
15991             \seq_map_inline:Nn
15992               \l_@@_image_identifiers_seq
15993               { \label { ##1 } }
15994           \end { center }
15995         \end { figure }
15996       }
15997     },
15998   }
15999 }
16000 \@@_if_option:nT
16001   { linkAttributes }
16002   {
16003     \RequirePackage { graphicx }
16004   }
16005 \markdownSetup {
16006   rendererPrototypes = {
16007     imageAttributeContextBegin = {
16008       \group_begin:
16009       \markdownSetup {
16010         rendererPrototypes = {
16011           attributeIdentifier = {
16012             \seq_put_right:Nn
16013               \l_@@_image_identifiers_seq
16014               { ##1 }

```

```

16015         },
16016         attributeKeyValue = {
16017             \setkeys
16018             { Gin }
16019             { { ##1 } = { ##2 } }
16020         },
16021     },
16022 }
16023 },
16024 imageAttributeContextEnd = {
16025     \group_end:
16026 },
16027 },
16028 }
16029 \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`.

```

16030 \ExplSyntaxOn
16031 \cs_new:Nn
16032   \@@_luaxml_print_html:n
16033   {
16034     \luabridge_now:n
16035     {
16036       local~input_file = assert(io.open(" #1 ", "r"))
16037       local~input = assert(input_file:read("*a"))
16038       assert(input_file:close())
16039       input = "<body>" .. input .. "</body>"
16040       local~dom = require("luaxml-domobject").html_parse(input)
16041       local~output = require("luaxml-htmltemplates"):process_dom(dom)
16042       print(output)
16043     }
16044   }
16045 \cs_gset_protected:Npn
16046   \markdownRendererInputRawInlinePrototype#1#2
16047   {
16048     \str_case:nnF
16049     { #2 }
16050     {
16051       { latex }
16052       {
16053         \@@_plain_tex_default_input_raw_inline:nn
16054         { #1 }

```

```

16055         { tex }
16056     }
16057     { html }
16058     {

```

If we are using  $\text{\TeX}4\text{ht}$ <sup>37</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16059         \cs_if_exist:NTF
16060         \HCode
16061         {
16062             \if_mode_vertical:
16063             \IgnorePar
16064             \EndP
16065             \fi:
16066             \special
16067             { t4ht* < #1 }
16068         }
16069     {
16070         \@@_luaxml_print_html:n
16071         { #1 }
16072     }
16073 }
16074 }
16075 {
16076     \@@_plain_tex_default_input_raw_inline:nn
16077     { #1 }
16078     { #2 }
16079 }
16080 }
16081 \cs_gset_protected:Npn
16082 \markdownRendererInputRawBlockPrototype#1#2
16083 {
16084     \str_case:nnF
16085     { #2 }
16086     {
16087         { latex }
16088         {
16089             \@@_plain_tex_default_input_raw_block:nn
16090             { #1 }
16091             { tex }
16092         }
16093     { html }
16094     {

```

---

<sup>37</sup>See <https://tug.org/tex4ht/>.

If we are using  $\text{\TeX}4\text{ht}$ <sup>38</sup>, we will pass HTML elements to the output HTML document unchanged.

```

16095         \cs_if_exist:NTF
16096         \HCode
16097         {
16098             \if_mode_vertical:
16099             \IgnorePar
16100             \fi:
16101             \EndP
16102             \special
16103             { t4ht* < #1 }
16104             \par
16105             \ShowPar
16106         }
16107         {
16108             \@@_luaxml_print_html:n
16109             { #1 }
16110         }
16111     }
16112 }
16113 {
16114     \@@_plain_tex_default_input_raw_block:nn
16115     { #1 }
16116     { #2 }
16117 }
16118 }

```

### 3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as  $\text{\LaTeX}$  labels for referencing the last  $\text{\LaTeX}$  counter that has been incremented in e.g. ordered lists.

```

16119 \seq_new:N
16120 \l_@@_bracketed_span_identifiers_seq
16121 \markdownSetup {
16122     rendererPrototypes = {
16123         bracketedSpanAttributeContextBegin = {
16124             \group_begin:
16125             \markdownSetup {
16126                 rendererPrototypes = {
16127                     attributeIdentifier = {
16128                         \seq_put_right:Nn
16129                         \l_@@_bracketed_span_identifiers_seq
16130                         { ##1 }
16131                     },

```

---

<sup>38</sup>See <https://tug.org/tex4ht/>.

```

16132     },
16133   }
16134 },
16135 bracketedSpanAttributeContextEnd = {
16136   \seq_map_inline:Nn
16137   \l_@@_bracketed_span_identifiers_seq
16138   { \label { ##1 } }
16139   \group_end:
16140 },
16141 },
16142 }
16143 \ExplSyntaxOff
16144 \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.

```

16145 \newcommand\markdownMakeOther{%
16146   \count0=128\relax
16147   \loop
16148     \catcode\count0=11\relax
16149     \advance\count0 by 1\relax
16150   \ifnum\count0<256\repeat}%

```

## 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents LATEX` package.

```

16151 \def\markdownMakeOther{%
16152   \count0=128\relax
16153   \loop
16154     \catcode\count0=11\relax
16155     \advance\count0 by 1\relax
16156   \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.

```
16157 \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).

```
16158 \long\def\inputmarkdown{%
16159   \dosingleempty
16160   \doinputmarkdown}%
16161 \long\def\doinputmarkdown[#1]#2{%
16162   \begingroup
16163     \iffirstargument
16164     \setupmarkdown[#1]%
16165     \fi
16166     \markdownInput{#2}%
16167   \endgroup}%
16168 \long\def\inputyaml{%
16169   \dosingleempty
16170   \doinputyaml}%
16171 \long\def\doinputyaml[#1]#2{%
16172   \doinputmarkdown
16173   [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. [18, sec. 31]. According to Eijkhout [19, 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.

```
16174 \startluacode
16175   document.markdown_buffering = false
16176   local function preserve_trailing_spaces(line)
16177     if document.markdown_buffering then
16178       line = line:gsub("[ \t][ \t]$", "\t\t")
16179     end
16180     return line
16181   end
16182   resolvers.installinputlinehandler(preserve_trailing_spaces)
16183 \stopluacode
16184 \begingroup
16185   \catcode`\|=0%
16186   \catcode`\|=12%
```

```

16187 |gdef|startmarkdown{%
16188   |ctxlua{document.markdown_buffering = true}%
16189   |markdownReadAndConvert{\stopmarkdown}%
16190                               {|stopmarkdown}}%
16191 |gdef|stopmarkdown{%
16192   |ctxlua{document.markdown_buffering = false}%
16193   |markdownEnd}%
16194 |gdef|startyaml{%
16195   |begingroup
16196   |ctxlua{document.markdown_buffering = true}%
16197   |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
16198   |markdownReadAndConvert{\stopyaml}%
16199                               {|stopyaml}}%
16200 |gdef|stopyaml{%
16201   |ctxlua{document.markdown_buffering = false}%
16202   |yamlEnd}%
16203 |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  $\text{Con}\TeX$ t themes provided with the Markdown package.

```

16204 \ExplSyntaxOn
16205 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
16206 \prop_new:N \g_@@_context_loaded_themes_versions_prop
16207 \cs_gset:Nn
16208   \@@_load_theme:nnn
16209   {

```

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.

```

16210   \bool_if:nTF
16211     {
16212       \bool_lazy_or_p:nn
16213         {
16214           \prop_if_in_p:Nn
16215             \g_@@_context_built_in_themes_prop
16216             { #1 }
16217         }
16218         {
16219           \file_if_exist_p:n
16220             { t - markdown theme #3.tex }
16221         }

```



```

16222     }
16223     {
16224         \prop_get:NnNTF
16225         \g_@@_context_loaded_themes_linenos_prop
16226         { #1 }
16227         \l_tmpa_tl
16228         {
16229             \prop_get:NnN
16230             \g_@@_context_loaded_themes_versions_prop
16231             { #1 }
16232             \l_tmpb_tl
16233             \str_if_eq:nVTF
16234             { #2 }
16235             \l_tmpb_tl
16236             {
16237                 \msg_warning:nnnVn
16238                 { markdown }
16239                 { repeatedly-loaded-context-theme }
16240                 { #1 }
16241                 \l_tmpa_tl
16242                 { #2 }
16243             }
16244             {
16245                 \msg_error:nnnnVV
16246                 { markdown }
16247                 { different-versions-of-context-theme }
16248                 { #1 }
16249                 { #2 }
16250                 \l_tmpb_tl
16251                 \l_tmpa_tl
16252             }
16253         }
16254     }
16255     \prop_gput:Nnx
16256     \g_@@_context_loaded_themes_linenos_prop
16257     { #1 }
16258     { \tex_the:D \tex_inputlineno:D } % noqa: W200
16259     \prop_gput:Nnn
16260     \g_@@_context_loaded_themes_versions_prop
16261     { #1 }
16262     { #2 }

```

Load built-in plain  $\text{\TeX}$  themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

16263     \prop_if_in:NnTF
16264     \g_@@_context_built_in_themes_prop
16265     { #1 }

```

```

16266         {
16267             \msg_info:nnnn
16268             { markdown }
16269             { loading-built-in-context-theme }
16270             { #1 }
16271             { #2 }
16272             \prop_item:Nn
16273             \g_@@_context_built_in_themes_prop
16274             { #1 }
16275         }
16276         {
16277             \msg_info:nnnn
16278             { markdown }
16279             { loading-context-theme }
16280             { #1 }
16281             { #2 }
16282             \usemodule
16283             [ t ]
16284             [ markdown theme #3 ]
16285         }
16286     }
16287 }
16288 {
16289     \@@_plain_tex_load_theme:nnn
16290     { #1 }
16291     { #2 }
16292     { #3 }
16293 }
16294 }
16295 \msg_new:nnn
16296 { markdown }
16297 { loading-built-in-context-theme }
16298 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
16299 \msg_new:nnn
16300 { markdown }
16301 { loading-context-theme }
16302 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
16303 \msg_new:nnn
16304 { markdown }
16305 { repeatedly-loaded-context-theme }
16306 {
16307     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
16308     loaded~on~line~#2,~not~loading~it~again
16309 }
16310 \msg_new:nnn
16311 { markdown }
16312 { different-versions-of-context-theme }

```

```

16313 {
16314   Tried-to-load-version-#2-of-ConTeXt-Markdown-theme-#1~
16315   but-version-#3-has-already-been-loaded-on-line-#4
16316 }
16317 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```
16318 \markdownLoadPlainTeXTheme
```

Next, the ConTeXt theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

16319 \markdownIfOption{plain}{\iffalse}{\iftrue}
16320 \def\markdownRendererHardLineBreakPrototype{\blank}%
16321 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
16322 \def\markdownRendererRightBracePrototype{\textbraceright}%
16323 \def\markdownRendererDollarSignPrototype{\textdollar}%
16324 \def\markdownRendererPercentSignPrototype{\percent}%
16325 \def\markdownRendererUnderscorePrototype{\textunderscore}%
16326 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
16327 \def\markdownRendererBackslashPrototype{\textbackslash}%
16328 \def\markdownRendererTildePrototype{\textasciitilde}%
16329 \def\markdownRendererPipePrototype{\char`|}%
16330 \def\markdownRendererLinkPrototype#1#2#3#4{%
16331   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
16332   \fi\tt<\hyphenatedurl{#3}>}}%
16333 \usemodule[database]
16334 \defineseparatedlist
16335   [MarkdownConTeXtCSV]
16336   [separator={,},
16337   before=\bTABLE,after=\eTABLE,
16338   first=\bTR,last=\eTR,
16339   left=\bTD,right=\eTD]
16340 \def\markdownConTeXtCSV{csv}
16341 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
16342   \def\markdownConTeXtCSV@arg{#1}%
16343   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
16344     \placetable[] [tab:#1]{#4}{%
16345       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
16346   \else
16347     \markdownInput{#3}%

```

```

16348 \fi}%
16349 \def\markdownRendererImagePrototype#1#2#3#4{%
16350 \placefigure [] []{#4}{\externalfigure[#3]}}%
16351 \def\markdownRendererUlBeginPrototype{\startitemize}%
16352 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
16353 \def\markdownRendererUlItemPrototype{\item}%
16354 \def\markdownRendererUlEndPrototype{\stopitemize}%
16355 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
16356 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
16357 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
16358 \def\markdownRendererOlItemPrototype{\item}%
16359 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
16360 \def\markdownRendererOlEndPrototype{\stopitemize}%
16361 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
16362 \definedescription
16363 [MarkdownConTeXtDlItemPrototype]
16364 [location=hanging,
16365 margin=standard,
16366 headstyle=bold]%
16367 \definestartstop
16368 [MarkdownConTeXtDlPrototype]
16369 [before=\blank,
16370 after=\blank]%
16371 \definestartstop
16372 [MarkdownConTeXtDlTightPrototype]
16373 [before=\blank\startpacked,
16374 after=\stoppacked\blank]%
16375 \def\markdownRendererDlBeginPrototype{%
16376 \startMarkdownConTeXtDlPrototype}%
16377 \def\markdownRendererDlBeginTightPrototype{%
16378 \startMarkdownConTeXtDlTightPrototype}%
16379 \def\markdownRendererDlItemPrototype#1{%
16380 \startMarkdownConTeXtDlItemPrototype{#1}}%
16381 \def\markdownRendererDlItemEndPrototype{%
16382 \stopMarkdownConTeXtDlItemPrototype}%
16383 \def\markdownRendererDlEndPrototype{%
16384 \stopMarkdownConTeXtDlPrototype}%
16385 \def\markdownRendererDlEndTightPrototype{%
16386 \stopMarkdownConTeXtDlTightPrototype}%
16387 \def\markdownRendererEmphasisPrototype#1{{\em#1}}%
16388 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
16389 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
16390 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
16391 \def\markdownRendererLineBlockBeginPrototype{%
16392 \begingroup
16393 \def\markdownRendererHardLineBreak{
16394 }%

```

```

16395     \startlines
16396 }%
16397 \def\markdownRendererLineBlockEndPrototype{%
16398     \stoptlines
16399     \endgroup
16400 }%
16401 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

16402 \ExplSyntaxOn
16403 \cs_gset:Npn
16404   \markdownRendererInputFencedCodePrototype#1#2#3
16405   {
16406     \tl_if_empty:nTF
16407       { #2 }
16408       { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\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

```

```

16409   {
16410     \regex_extract_once:nnN
16411       { \w* }
16412       { #2 }
16413       \l_tmpa_seq
16414     \seq_pop_left:NN
16415       \l_tmpa_seq

```

```

16416         \l_tmpa_tl
16417         \typefile[ \l_tmpa_tl ][] {#1}
16418     }
16419 }
16420 \ExplSyntaxOff
16421 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
16422 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
16423 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
16424 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
16425 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
16426 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
16427 \def\markdownRendererThematicBreakPrototype{%
16428     \blackrule[height=1pt, width=\hsize]}%
16429 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
16430 \def\markdownRendererTickedBoxPrototype{\$ \boxtimes $}
16431 \def\markdownRendererHalfTickedBoxPrototype{\$ \boxdot $}
16432 \def\markdownRendererUntickedBoxPrototype{\$ \square $}
16433 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
16434 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
16435 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
16436 \def\markdownRendererDisplayMathPrototype#1{%
16437     \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

16438 \newcount\markdownConTeXtRowCounter
16439 \newcount\markdownConTeXtRowTotal
16440 \newcount\markdownConTeXtColumnCounter
16441 \newcount\markdownConTeXtColumnTotal
16442 \newtoks\markdownConTeXtTable
16443 \newtoks\markdownConTeXtTableFloat
16444 \def\markdownRendererTablePrototype#1#2#3{%
16445     \markdownConTeXtTable={}%
16446     \ifx\empty#1\empty
16447         \markdownConTeXtTableFloat={%
16448             \the\markdownConTeXtTable}%
16449     \else
16450         \markdownConTeXtTableFloat={%
16451             \placetable{#1}{\the\markdownConTeXtTable}}%
16452     \fi
16453     \begingroup
16454     \setupTABLE[r][each][topframe=off, bottomframe=off,
16455         leftframe=off, rightframe=off]
16456     \setupTABLE[c][each][topframe=off, bottomframe=off,
16457         leftframe=off, rightframe=off]
16458     \setupTABLE[r][1][topframe=on, bottomframe=on]

```

```

16459 \setupTABLE[r] [#1] [bottomframe=on]
16460 \markdownConTeXtRowCounter=0%
16461 \markdownConTeXtRowTotal=#2%
16462 \markdownConTeXtColumnTotal=#3%
16463 \markdownConTeXtRenderTableRow}
16464 \def\markdownConTeXtRenderTableRow#1{%
16465 \markdownConTeXtColumnCounter=0%
16466 \ifnum\markdownConTeXtRowCounter=0\relax
16467 \markdownConTeXtReadAlignments#1%
16468 \markdownConTeXtTable={\bTABLE}%
16469 \else
16470 \markdownConTeXtTable=\expandafter{%
16471 \the\markdownConTeXtTable\bTR}%
16472 \markdownConTeXtRenderTableCell#1%
16473 \markdownConTeXtTable=\expandafter{%
16474 \the\markdownConTeXtTable\eTR}%
16475 \fi
16476 \advance\markdownConTeXtRowCounter by 1\relax
16477 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
16478 \markdownConTeXtTable=\expandafter{%
16479 \the\markdownConTeXtTable\eTABLE}%
16480 \the\markdownConTeXtTableFloat
16481 \endgroup
16482 \expandafter\gobbleoneargument
16483 \fi\markdownConTeXtRenderTableRow}
16484 \def\markdownConTeXtReadAlignments#1{%
16485 \advance\markdownConTeXtColumnCounter by 1\relax
16486 \if#1d%
16487 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=right]
16488 \fi\if#1l%
16489 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=right]
16490 \fi\if#1c%
16491 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=middle]
16492 \fi\if#1r%
16493 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=left]
16494 \fi
16495 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16496 \else
16497 \expandafter\gobbleoneargument
16498 \fi\markdownConTeXtReadAlignments}
16499 \def\markdownConTeXtRenderTableCell#1{%
16500 \advance\markdownConTeXtColumnCounter by 1\relax
16501 \markdownConTeXtTable=\expandafter{%
16502 \the\markdownConTeXtTable\bTD#1\eTD}%
16503 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16504 \else
16505 \expandafter\gobbleoneargument

```

```
16506 \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`.

```
16507 \ExplSyntaxOn
16508 \cs_gset:Npn
16509   \markdownRendererInputRawInlinePrototype#1#2
16510   {
16511     \str_case:nnF
16512       { #2 }
16513       {
16514         { latex }
16515         {
16516           \@@_plain_tex_default_input_raw_inline:nn
16517             { #1 }
16518             { context }
16519         }
16520       }
16521     {
16522       \@@_plain_tex_default_input_raw_inline:nn
16523         { #1 }
16524         { #2 }
16525     }
16526   }
16527 \cs_gset:Npn
16528   \markdownRendererInputRawBlockPrototype#1#2
16529   {
16530     \str_case:nnF
16531       { #2 }
16532       {
16533         { context }
16534         {
16535           \@@_plain_tex_default_input_raw_block:nn
16536             { #1 }
16537             { tex }
16538         }
16539       }
16540     {
16541       \@@_plain_tex_default_input_raw_block:nn
16542         { #1 }
16543         { #2 }
16544     }
16545   }
16546 \cs_gset_eq:NN
16547   \markdownRendererInputRawBlockPrototype
```



```

16548 \markdownRendererInputRawInlinePrototype
16549 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}`
16550 \ExplSyntaxOff
16551 \stopmodule
16552 \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).

```

16553 \ExplSyntaxOn
16554 \str_if_eq:VVT
16555 \c_@@_top_layer_tl
16556 \c_@@_option_layer_context_tl
16557 {
16558   \use:c
16559     { ExplSyntaxOff }
16560   \@@_if_option:nF
16561     { noDefaults }
16562     {
16563       \@@_if_option:nTF
16564         { experimental }
16565         {
16566           \@@_setup:n
16567             { theme = witiko/markdown/defaults@experimental }
16568         }
16569         {
16570           \@@_setup:n
16571             { theme = witiko/markdown/defaults }
16572         }
16573     }
16574   \use:c
16575     { ExplSyntaxOn }
16576 }
16577 \ExplSyntaxOff
16578 \stopmodule
16579 \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] L<sup>A</sup>T<sub>E</sub>X Project. *l3kernel. L<sup>A</sup>T<sub>E</sub>X3 programming conventions*. Dec. 25, 2024. URL: <https://ctan.org/pkg/l3kernel> (visited on 01/06/2025).

- [3] Frank Mittelbach, Ulrike Fischer, and L<sup>A</sup>T<sub>E</sub>X Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).
- [4] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [5] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [6] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [7] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [8] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [9] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [10] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [11] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [12] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [13] Vít Starý Novotný. *Routing YAML metadata to expl3 key-values*. Markdown Enhancement Proposal. Oct. 14, 2024. URL: <https://github.com/witiko/markdown/discussions/517> (visited on 01/06/2025).
- [14] Frank Mittelbach. *L<sup>A</sup>T<sub>E</sub>X’s hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [15] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).

- [16] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [17] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [18] 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.
- [19] 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, 85, 101                        |
| blankBeforeBlockquote           | 21                                     |
| blankBeforeCodeFence            | 22                                     |
| blankBeforeDivFence             | 22                                     |
| blankBeforeHeading              | 22                                     |
| blankBeforeList                 | 22                                     |
| bracketedSpans                  | 23, 87, 461                            |
| breakableBlockquotes            | 23                                     |
| cacheDir                        | 4, 17, 19, 59, 159, 172, 373, 393, 413 |
| citationNbsps                   | 23                                     |
| citations                       | 24, 90, 91                             |
| codeSpans                       | 24                                     |
| contentBlocks                   | 20, 25, 34                             |
| contentBlocksLanguageMap        | 20                                     |
| contentLevel                    | 25                                     |
| debugExtensions                 | 9, 20, 26, 317                         |
| debugExtensionsFileName         | 20, 26                                 |
| defaultOptions                  | 10, 51, 372, 373                       |
| definitionLists                 | 26, 95                                 |
| eagerCache                      | 17, 371                                |
| ensureJekyllData                | 27                                     |
| entities.char_entity            | 218                                    |
| entities.dec_entity             | 218                                    |
| entities.hex_entity             | 218                                    |
| entities.hex_entity_with_x_char | 218                                    |
| escape_minimal                  | 222                                    |

|                                       |                                   |
|---------------------------------------|-----------------------------------|
| escape_programmatic_text              | 222                               |
| escape_typographic_text               | 222                               |
| expandtabs                            | 277                               |
| expectJekyllData                      | 27, 27                            |
| experimental                          | 5, 17, 431                        |
| extensions                            | 28, 167, 322                      |
| extensions.bracketed_spans            | 323                               |
| extensions.citations                  | 324                               |
| extensions.content_blocks             | 328                               |
| extensions.definition_lists           | 331                               |
| extensions.fancy_lists                | 333                               |
| extensions.fenced_code                | 339                               |
| extensions.fenced_divs                | 345                               |
| extensions.header_attributes          | 349                               |
| extensions.inline_code_attributes     | 351                               |
| extensions.jekyll_data                | 367                               |
| extensions.line_blocks                | 351                               |
| extensions.link_attributes            | 353                               |
| extensions.mark                       | 352                               |
| extensions.notes                      | 354                               |
| extensions.pipe_table                 | 357                               |
| extensions.raw_inline                 | 361                               |
| extensions.strike_through             | 362                               |
| extensions.subscripts                 | 362                               |
| extensions.superscripts               | 363                               |
| extensions.tex_math                   | 364                               |
| <br>                                  |                                   |
| fancyLists                            | 30, 112–117, 431                  |
| fencedCode                            | 30, 39, 92, 99, 118, 385, 387     |
| fencedCodeAttributes                  | 31, 85, 99, 387                   |
| fencedDiv                             | 100                               |
| fencedDivs                            | 31, 41                            |
| finalizeCache                         | 17, 20, 32, 32, 59, 158, 371, 372 |
| frozenCache                           | 20, 32, 59, 75, 158, 385, 393     |
| frozenCacheCounter                    | 32, 372, 420, 421                 |
| frozenCacheFileName                   | 20, 32, 59, 372                   |
| <br>                                  |                                   |
| \g_markdown_diagrams_infostrings_prop | 389                               |
| gfmAutoIdentifiers                    | 21, 32, 85, 101                   |
| <br>                                  |                                   |
| hashEnumerators                       | 33                                |
| headerAttributes                      | 33, 41, 85, 101                   |
| html                                  | 34, 104, 444                      |

|                                  |                                                     |
|----------------------------------|-----------------------------------------------------|
| hybrid                           | 34, 34, 40, 46, 48, 62, 78, 119, 159, 223, 278, 420 |
| inlineCodeAttributes             | 35, 85, 93                                          |
| inlineNotes                      | 36                                                  |
| \input                           | 55, 56                                              |
| \inputmarkdown                   | 162, 163, 165, 463                                  |
| inputTempFileName                | 60, 62, 414, 415, 417, 418                          |
| \inputyaml                       | 162, 164, 463                                       |
| iterlines                        | 277                                                 |
| jekyllData                       | 3, 27, 28, 36, 128–131, 134                         |
| \l_file_search_path_seq          | 419                                                 |
| languages_json                   | 328, 328                                            |
| lineBlocks                       | 37, 107                                             |
| linkAttributes                   | 37, 85, 105, 109, 298, 457                          |
| mark                             | 38, 110, 456                                        |
| \markdown                        | 154, 155, 424                                       |
| markdown                         | 153, 153, 154, 423                                  |
| markdown*                        | 153, 153, 154, 158, 423                             |
| \markdownBegin                   | 53, 53–56, 151–153, 155, 162, 163                   |
| \markdownCleanup                 | 413                                                 |
| \markdownConvert                 | 413                                                 |
| \markdownEnd                     | 53, 53–56, 152–155, 162, 163                        |
| \markdownError                   | 151, 151                                            |
| \markdownEscape                  | 53, 56, 421                                         |
| \markdownIfOption                | 58                                                  |
| \markdownIfSnippetExists         | 80                                                  |
| \markdownInfo                    | 151, 151                                            |
| \markdownInput                   | 53, 55, 56, 153, 156, 158, 163, 419, 422            |
| \markdownInputFilename           | 412                                                 |
| \markdownInputFileStream         | 413                                                 |
| \markdownInputPlainTeX           | 422                                                 |
| \markdownLoadPlainTeXTheme       | 160, 166, 383                                       |
| \markdownLuaExecute              | 416, 419                                            |
| \markdownLuaOptions              | 409, 413                                            |
| \markdownMakeOther               | 151, 462                                            |
| \markdownOptionFinalizeCache     | 59                                                  |
| \markdownOptionFrozenCache       | 59                                                  |
| \markdownOptionHybrid            | 62                                                  |
| \markdownOptionInputTempFileName | 59                                                  |
| \markdownOptionNoDefaults        | 61                                                  |

|                                                                  |                         |
|------------------------------------------------------------------|-------------------------|
| <code>\markdownOptionOutputDir</code>                            | 60, 60, 63              |
| <code>\markdownOptionPlain</code>                                | 60                      |
| <code>\markdownOptionStripPercentSigns</code>                    | 61                      |
| <code>\markdownOutputFileStream</code>                           | 413                     |
| <code>\markdownPrepare</code>                                    | 412                     |
| <code>\markdownPrepareInputFilename</code>                       | 412                     |
| <code>\markdownPrepareLuaOptions</code>                          | 409                     |
| <code>\markdownReadAndConvert</code>                             | 151, 414, 423, 424, 463 |
| <code>\markdownReadAndConvertProcessLine</code>                  | 415, 416                |
| <code>\markdownReadAndConvertStripPercentSigns</code>            | 414                     |
| <code>\markdownReadAndConvertTab</code>                          | 413                     |
| <code>\markdownRendererAttributeName</code>                      | 85                      |
| <code>\markdownRendererAttributeIdentifier</code>                | 85                      |
| <code>\markdownRendererAttributeKeyValue</code>                  | 85                      |
| <code>\markdownRendererBlockQuoteBegin</code>                    | 86                      |
| <code>\markdownRendererBlockQuoteEnd</code>                      | 87                      |
| <code>\markdownRendererBracketedSpanAttributeContextBegin</code> | 87                      |
| <code>\markdownRendererBracketedSpanAttributeContextEnd</code>   | 87                      |
| <code>\markdownRendererCite</code>                               | 90, 91                  |
| <code>\markdownRendererCodeSpan</code>                           | 92                      |
| <code>\markdownRendererCodeSpanAttributeContextBegin</code>      | 93                      |
| <code>\markdownRendererCodeSpanAttributeContextEnd</code>        | 93                      |
| <code>\markdownRendererContentBlock</code>                       | 93, 94                  |
| <code>\markdownRendererContentBlockCode</code>                   | 94                      |
| <code>\markdownRendererContentBlockOnlineImage</code>            | 94                      |
| <code>\markdownRendererDisplayMath</code>                        | 125                     |
| <code>\markdownRendererDlBegin</code>                            | 95                      |
| <code>\markdownRendererDlBeginTight</code>                       | 95                      |
| <code>\markdownRendererDlDefinitionBegin</code>                  | 96                      |
| <code>\markdownRendererDlDefinitionEnd</code>                    | 97                      |
| <code>\markdownRendererDlEnd</code>                              | 97                      |
| <code>\markdownRendererDlEndTight</code>                         | 98                      |
| <code>\markdownRendererDlItem</code>                             | 96                      |
| <code>\markdownRendererDlItemEnd</code>                          | 96                      |
| <code>\markdownRendererDocumentBegin</code>                      | 110                     |
| <code>\markdownRendererDocumentEnd</code>                        | 110                     |
| <code>\markdownRendererEllipsis</code>                           | 41, 98                  |
| <code>\markdownRendererEmphasis</code>                           | 98, 138                 |
| <code>\markdownRendererError</code>                              | 127                     |
| <code>\markdownRendererFancyOlBegin</code>                       | 112, 113                |
| <code>\markdownRendererFancyOlBeginTight</code>                  | 113                     |
| <code>\markdownRendererFancyOlEnd</code>                         | 117                     |

|                                                               |               |
|---------------------------------------------------------------|---------------|
| <code>\markdownRendererFancyOlEndTight</code>                 | 117           |
| <code>\markdownRendererFancyOlItem</code>                     | 115           |
| <code>\markdownRendererFancyOlItemEnd</code>                  | 115           |
| <code>\markdownRendererFancyOlItemWithNumber</code>           | 115           |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code> | 99            |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code>   | 99            |
| <code>\markdownRendererFencedDivAttributeContextBegin</code>  | 100           |
| <code>\markdownRendererFencedDivAttributeContextEnd</code>    | 100           |
| <code>\markdownRendererHalfTickedBox</code>                   | 126           |
| <code>\markdownRendererHardLineBreak</code>                   | 108           |
| <code>\markdownRendererHeaderAttributeContextBegin</code>     | 101           |
| <code>\markdownRendererHeaderAttributeContextEnd</code>       | 101           |
| <code>\markdownRendererHeadingFive</code>                     | 103           |
| <code>\markdownRendererHeadingFour</code>                     | 103           |
| <code>\markdownRendererHeadingOne</code>                      | 101           |
| <code>\markdownRendererHeadingSix</code>                      | 103           |
| <code>\markdownRendererHeadingThree</code>                    | 102           |
| <code>\markdownRendererHeadingTwo</code>                      | 102           |
| <code>\markdownRendererImage</code>                           | 105           |
| <code>\markdownRendererImageAttributeContextBegin</code>      | 105           |
| <code>\markdownRendererImageAttributeContextEnd</code>        | 105           |
| <code>\markdownRendererInlineHtmlComment</code>               | 104           |
| <code>\markdownRendererInlineHtmlTag</code>                   | 104           |
| <code>\markdownRendererInlineMath</code>                      | 125           |
| <code>\markdownRendererInputBlockHtmlElement</code>           | 104           |
| <code>\markdownRendererInputFencedCode</code>                 | 92            |
| <code>\markdownRendererInputRawBlock</code>                   | 118           |
| <code>\markdownRendererInputRawInline</code>                  | 117           |
| <code>\markdownRendererInputVerbatim</code>                   | 91            |
| <code>\markdownRendererInterblockSeparator</code>             | 106           |
| <code>\markdownRendererJekyllDataBegin</code>                 | 128           |
| <code>\markdownRendererJekyllDataBoolean</code>               | 130           |
| <code>\markdownRendererJekyllDataEmpty</code>                 | 134           |
| <code>\markdownRendererJekyllDataEnd</code>                   | 128           |
| <code>\markdownRendererJekyllDataMappingBegin</code>          | 129           |
| <code>\markdownRendererJekyllDataMappingEnd</code>            | 129           |
| <code>\markdownRendererJekyllDataNumber</code>                | 131           |
| <code>\markdownRendererJekyllDataProgrammaticString</code>    | 131, 131, 132 |
| <code>\markdownRendererJekyllDataSequenceBegin</code>         | 130           |
| <code>\markdownRendererJekyllDataSequenceEnd</code>           | 130           |
| <code>\markdownRendererJekyllDataString</code>                | 132, 136      |
| <code>\markdownRendererJekyllDataStringPrototype</code>       | 147           |

|                                                           |                                     |
|-----------------------------------------------------------|-------------------------------------|
| <code>\markdownRendererJekyllDataTypographicString</code> | 131, 131, 132, 368                  |
| <code>\markdownRendererLineBlockBegin</code>              | 107                                 |
| <code>\markdownRendererLineBlockEnd</code>                | 107                                 |
| <code>\markdownRendererLink</code>                        | 108, 138                            |
| <code>\markdownRendererLinkAttributeContextBegin</code>   | 109                                 |
| <code>\markdownRendererLinkAttributeContextEnd</code>     | 109                                 |
| <code>\markdownRendererMark</code>                        | 110                                 |
| <code>\markdownRendererNbsp</code>                        | 111                                 |
| <code>\markdownRendererNote</code>                        | 111                                 |
| <code>\markdownRendererOlBegin</code>                     | 112                                 |
| <code>\markdownRendererOlBeginTight</code>                | 112                                 |
| <code>\markdownRendererOlEnd</code>                       | 116                                 |
| <code>\markdownRendererOlEndTight</code>                  | 116                                 |
| <code>\markdownRendererOlItem</code>                      | 42, 113                             |
| <code>\markdownRendererOlItemEnd</code>                   | 114                                 |
| <code>\markdownRendererOlItemWithNumber</code>            | 42, 114                             |
| <code>\markdownRendererParagraphSeparator</code>          | 107                                 |
| <code>\markdownRendererReplacementCharacter</code>        | 119                                 |
| <code>\markdownRendererSectionBegin</code>                | 118                                 |
| <code>\markdownRendererSectionEnd</code>                  | 118                                 |
| <code>\markdownRendererSoftLineBreak</code>               | 108                                 |
| <code>\markdownRendererStrikeThrough</code>               | 122                                 |
| <code>\markdownRendererStrongEmphasis</code>              | 99                                  |
| <code>\markdownRendererSubscript</code>                   | 123                                 |
| <code>\markdownRendererSuperscript</code>                 | 123                                 |
| <code>\markdownRendererTable</code>                       | 124                                 |
| <code>\markdownRendererTableAttributeContextBegin</code>  | 124                                 |
| <code>\markdownRendererTableAttributeContextEnd</code>    | 124                                 |
| <code>\markdownRendererTextCite</code>                    | 91                                  |
| <code>\markdownRendererThematicBreak</code>               | 126                                 |
| <code>\markdownRendererTickedBox</code>                   | 126                                 |
| <code>\markdownRendererUlBegin</code>                     | 88                                  |
| <code>\markdownRendererUlBeginTight</code>                | 88                                  |
| <code>\markdownRendererUlEnd</code>                       | 90                                  |
| <code>\markdownRendererUlEndTight</code>                  | 90                                  |
| <code>\markdownRendererUlItem</code>                      | 89                                  |
| <code>\markdownRendererUlItemEnd</code>                   | 89                                  |
| <code>\markdownRendererUntickedBox</code>                 | 126                                 |
| <code>\markdownRendererWarning</code>                     | 127                                 |
| <code>\markdownSetup</code>                               | 57, 58, 62, 157, 158, 165, 424, 430 |
| <code>\markdownSetupSnippet</code>                        | 79, 79                              |
| <code>\markdownThemeVersion</code>                        | 69, 69                              |



|                                                |                            |
|------------------------------------------------|----------------------------|
| <code>\markdownWarning</code>                  | 151, 151                   |
| <code>\markinline</code>                       | 53, 55, 153, 156, 417, 422 |
| <code>\markinlinePlainTeX</code>               | 422                        |
| <code>new</code>                               | 7, 18, 371, 373            |
| <code>notes</code>                             | 38, 111                    |
| <code>parsers</code>                           | 238, 276, 277              |
| <code>parsers.commented_line</code>            | 257                        |
| <code>parsers.punctuation</code>               | 239                        |
| <code>pipeTables</code>                        | 7, 39, 45, 124             |
| <code>preserveTabs</code>                      | 39, 43, 277                |
| <code>rawAttribute</code>                      | 35, 39, 40, 117, 118       |
| <code>reader</code>                            | 8, 29, 167, 238, 276, 323  |
| <code>reader-&gt;add_special_character</code>  | 8, 9, 29, 317              |
| <code>reader-&gt;auto_link_email</code>        | 306                        |
| <code>reader-&gt;auto_link_url</code>          | 306                        |
| <code>reader-&gt;create_parser</code>          | 277                        |
| <code>reader-&gt;finalize_grammar</code>       | 312, 378                   |
| <code>reader-&gt;initialize_named_group</code> | 317                        |
| <code>reader-&gt;insert_pattern</code>         | 8, 9, 29, 313, 319         |
| <code>reader-&gt;lookup_note_reference</code>  | 291                        |
| <code>reader-&gt;lookup_reference</code>       | 291                        |
| <code>reader-&gt;normalize_tag</code>          | 277                        |
| <code>reader-&gt;options</code>                | 276                        |
| <code>reader-&gt;parser_functions</code>       | 277                        |
| <code>reader-&gt;parser_functions.name</code>  | 277                        |
| <code>reader-&gt;parsers</code>                | 276, 276, 277              |
| <code>reader-&gt;register_link</code>          | 291                        |
| <code>reader-&gt;update_rule</code>            | 313, 316, 319              |
| <code>reader-&gt;writer</code>                 | 276                        |
| <code>reader.new</code>                        | 276, 276, 378              |
| <code>relativeReferences</code>                | 40                         |
| <code>\setupmarkdown</code>                    | 165, 165                   |
| <code>\setupyaml</code>                        | 165                        |
| <code>shiftHeadings</code>                     | 7, 41                      |
| <code>singletonCache</code>                    | 18                         |
| <code>slice</code>                             | 7, 41, 219, 231, 232       |
| <code>smartEllipses</code>                     | 41, 98, 159                |
| <code>\startmarkdown</code>                    | 162, 162, 463              |
| <code>startNumber</code>                       | 42, 113–115                |

|                                               |                                             |
|-----------------------------------------------|---------------------------------------------|
| <code>\startyaml</code>                       | 162, 162, 163, 463                          |
| <code>\stopmarkdown</code>                    | 162, 162, 463                               |
| <code>\stopyaml</code>                        | 162, 162, 163, 463                          |
| <code>strikeThrough</code>                    | 42, 122, 457                                |
| <code>stripIndent</code>                      | 43, 278                                     |
| <code>stripPercentSigns</code>                | 414                                         |
| <code>subscripts</code>                       | 43, 123                                     |
| <code>superscripts</code>                     | 44, 123                                     |
| <code>syntax</code>                           | 314, 318                                    |
| <br>                                          |                                             |
| <code>tableAttributes</code>                  | 44, 124, 453                                |
| <code>tableCaptions</code>                    | 7, 44, 45, 124                              |
| <code>taskLists</code>                        | 45, 126, 444                                |
| <code>texComments</code>                      | 46, 278                                     |
| <code>texMathDollars</code>                   | 35, 46, 125                                 |
| <code>texMathDoubleBackslash</code>           | 35, 47, 125                                 |
| <code>texMathSingleBackslash</code>           | 35, 47, 125                                 |
| <code>tightLists</code>                       | 47, 88, 90, 95, 98, 112, 113, 116, 117, 431 |
| <br>                                          |                                             |
| <code>underscores</code>                      | 48                                          |
| <code>unicodeNormalization</code>             | 18, 19                                      |
| <code>unicodeNormalizationForm</code>         | 18, 19                                      |
| <code>util.cache</code>                       | 167, 168                                    |
| <code>util.cache_verbatim</code>              | 168                                         |
| <code>util.encode_json_string</code>          | 168                                         |
| <code>util.err</code>                         | 167                                         |
| <code>util.escaper</code>                     | 170                                         |
| <code>util.expand_tabs_in_line</code>         | 168                                         |
| <code>util.flatten</code>                     | 169                                         |
| <code>util.intersperse</code>                 | 170                                         |
| <code>util.map</code>                         | 170                                         |
| <code>util.pathname</code>                    | 171                                         |
| <code>util.rope_last</code>                   | 170                                         |
| <code>util.rope_to_string</code>              | 169                                         |
| <code>util.salt</code>                        | 172                                         |
| <code>util.table_copy</code>                  | 168                                         |
| <code>util.walk</code>                        | 168, 169, 170                               |
| <code>util.warning</code>                     | 172                                         |
| <br>                                          |                                             |
| <code>walkable_syntax</code>                  | 8, 20, 26, 312, 313, 316–318                |
| <code>writer</code>                           | 167, 167, 219, 323                          |
| <code>writer-&gt;active_attributes</code>     | 230, 230–232                                |
| <code>writer-&gt;attribute_type_levels</code> | 230                                         |

|                                 |          |
|---------------------------------|----------|
| writer->attributes              | 228      |
| writer->block_html_element      | 226      |
| writer->blockquote              | 227      |
| writer->bulletitem              | 225      |
| writer->bulletlist              | 225      |
| writer->citations               | 324      |
| writer->code                    | 223      |
| writer->contentblock            | 329      |
| writer->defer_call              | 238, 238 |
| writer->definitionlist          | 331      |
| writer->display_math            | 364      |
| writer->div_begin               | 345      |
| writer->div_end                 | 345      |
| writer->document                | 227      |
| writer->ellipsis                | 221      |
| writer->emphasis                | 227      |
| writer->error                   | 223      |
| writer->escape                  | 222      |
| writer->escaped_chars           | 222, 222 |
| writer->escaped_minimal_strings | 221, 222 |
| writer->escaped_strings         | 222      |
| writer->escaped_uri_chars       | 221, 222 |
| writer->fancyitem               | 335      |
| writer->fancylist               | 334      |
| writer->fencedCode              | 340      |
| writer->flatten_inlines         | 219, 219 |
| writer->get_state               | 237      |
| writer->hard_line_break         | 221      |
| writer->heading                 | 235      |
| writer->identifier              | 223      |
| writer->image                   | 224      |
| writer->infostring              | 223      |
| writer->inline_html_comment     | 226      |
| writer->inline_html_tag         | 226      |
| writer->inline_math             | 364      |
| writer->interblocksep           | 220      |
| writer->is_writing              | 219, 219 |
| writer->jekyllData              | 368      |
| writer->lineblock               | 351      |
| writer->link                    | 224      |
| writer->mark                    | 352      |
| writer->math                    | 223      |

|                         |                            |
|-------------------------|----------------------------|
| writer->nbsp            | 220                        |
| writer->note            | 355                        |
| writer->options         | 219                        |
| writer->ordereditem     | 226                        |
| writer->orderedlist     | 225                        |
| writer->paragraph       | 220                        |
| writer->paragraphsep    | 220                        |
| writer->plain           | 220                        |
| writer->pop_attributes  | 231, 231, 232              |
| writer->push_attributes | 231, 231, 232              |
| writer->rawBlock        | 340                        |
| writer->rawInline       | 361                        |
| writer->set_state       | 237                        |
| writer->slice_begin     | 219                        |
| writer->slice_end       | 219                        |
| writer->soft_line_break | 221                        |
| writer->space           | 220                        |
| writer->span            | 323                        |
| writer->strike_through  | 362                        |
| writer->string          | 223                        |
| writer->strong          | 227                        |
| writer->subscript       | 362                        |
| writer->superscript     | 363                        |
| writer->table           | 358                        |
| writer->thematic_break  | 221                        |
| writer->checkbox        | 227                        |
| writer->undosep         | 221, 321                   |
| writer->uri             | 223                        |
| writer->verbatim        | 227                        |
| writer->warning         | 172, 223                   |
| writer.new              | 219, 219, 378              |
|                         |                            |
| \yaml                   | 155                        |
| yaml                    | 153, 155, 423              |
| \yamlBegin              | 53, 54, 151, 152, 155, 162 |
| \yamlEnd                | 53, 54, 152, 155, 162      |
| \yamlInput              | 53, 56, 153, 156, 164, 422 |
| \yamlSetup              | 58                         |