

A Markdown Interpreter for T_EX

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

Version 3.9.0-0-g4f1abe21
2024-11-21

Contents

1	Introduction	1	3	Implementation	163
1.1	Requirements	2	3.1	Lua Implementation	163
1.2	Feedback	6	3.2	Plain T _E X Implementation	376
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation	406
2	Interfaces	7	3.4	ConT _E Xt Implementation	446
2.1	Lua Interface	7			
2.2	Plain T _E X Interface	53			
2.3	L ^A T _E X Interface	149			
2.4	ConT _E Xt Interface	158			
				References	457
				Index	459

List of Figures

1	A block diagram of the Markdown package	8
2	A sequence diagram of typesetting a document using the T _E X interface	49
3	A sequence diagram of typesetting a document using the Lua CLI	50
4	Various formats of mathematical formulae	74
5	The banner of the Markdown package	75

1 Introduction

The Markdown package¹ converts CommonMark² markup to T_EX commands. The functionality is provided both as a Lua module and as plain T_EX, L^AT_EX, and ConT_EXt macro packages that can be used to directly typeset T_EX documents containing markdown markup. Unlike other converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

This document is a technical documentation for the Markdown package. It consists of three sections. This section introduces the package and outlines its prerequisites. Section 2 describes the interfaces exposed by the package. Section 3 describes the implementation of the package. The technical documentation contains only a limited

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

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

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

1.1 Requirements

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

1.1.1 Lua Requirements

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
25 hard lua-uni-algos
26 local uni_algos = require("lua-uni-algos")
```

api7/lua-tinyyaml A library that provides a regex-based recursive descent YAML (subset) parser that is used to read YAML metadata when the `jeekyllData` option is enabled.

```
27 hard lua-tinyyaml
```

1.1.2 Plain T_EX Requirements

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

expl3 A package that enables the expl3 language from the L^AT_EX3 kernel in T_EX 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 \ifx\ExplSyntaxOn\undefined
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^AT_EX Requirements

The L^AT_EX part of the package requires that the L^AT_EX 2_ε format is loaded, a TeX engine that extends ϵ -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^AT_EX themes (see Section 2.3.4) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
36 soft url
```

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

```
37 soft graphics
```

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

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

```
38 soft enumitem
```

```
39 soft paralist
```

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

```
40 soft fancyvrb
```

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

```
41 soft csvsimple
```

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

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

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

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/dot` and `witiko/graphicx/http` plain \TeX 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 GitHub⁴ to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the \TeX - \LaTeX Stack Exchange.⁵ community question answering web site under the `markdown` tag.

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

⁵See <https://tex.stackexchange.com>.

1.3 Acknowledgements

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

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

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

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

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

2 Interfaces

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

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

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

2.1 Lua Interface

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

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

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

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

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

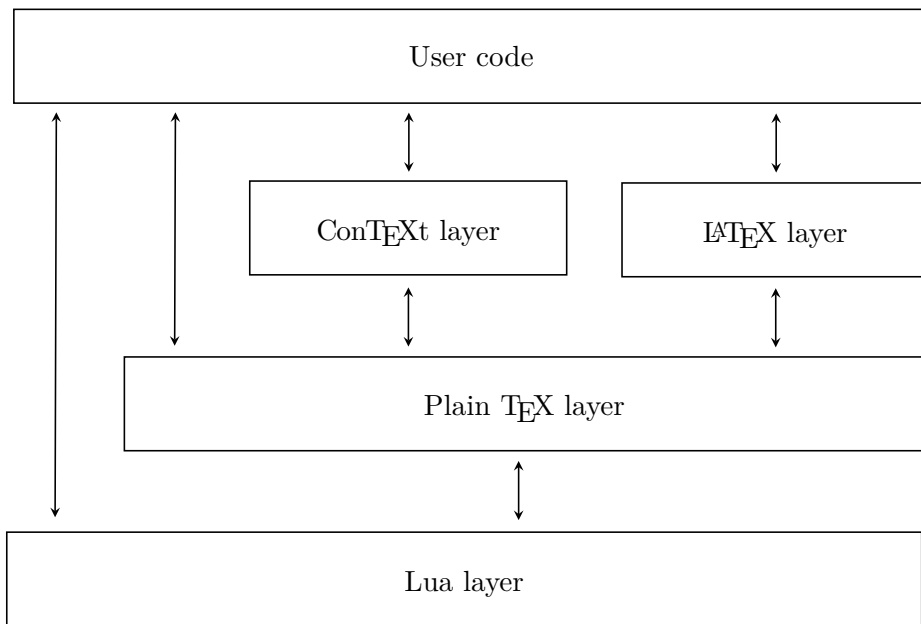


Figure 1: A block diagram of the Markdown package

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

The following example Lua code converts the markdown string `Hello *world*!` to a 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: }
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)⁶. 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 forms⁷ before conversion. The Unicode normalization norm used is determined by option `unicodeNormalizationForm`.

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

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

⁷See <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_EX implementations. The Lua implementation expects that the entire path already exists.

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

```

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_EX document that contains markdown documents without invoking Lua using the `frozenCache` plain T_EX option. As a result, the plain T_EX document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```

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 extension⁸:

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

```

⁸See 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 extension⁹:

`[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
```

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

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension¹⁰:

```
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
```

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

`contentBlocks=true, false`

default: `false`

`true`

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

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

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

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

505 defaultOptions.contentBlocks = false
```

`contentLevel=block, inline`

default: `block`

`block` Treat content as a sequence of blocks.

```
- this is a list
- it contains two items
```

`inline` Treat all content as inline content.

```
- this is a text
- not a list
```

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

510 defaultOptions.contentLevel = "block"
```

`debugExtensions=true, false`

default: `false`

- `true` Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.7) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.
- `false` Do not produce a JSON file with the PEG grammar of markdown.

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

515 defaultOptions.debugExtensions = false
```

`definitionLists=true, false`

default: `false`

- `true` Enable the pandoc definition list syntax extension:

```
Term 1

: Definition 1

Term 2 with inline markup

: Definition 2

    { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

- `false` Disable the pandoc definition list syntax extension.

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

520 defaultOptions.definitionLists = false
```

`ensureJekyllData=true, false`

default: false

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

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

`expectJekyllData=true, false`

default: false

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

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

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

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

```
526 \@@_add_lua_option:nnn
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 extension¹¹:

```
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


```

  <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
```

¹¹See <https://pandoc.org/MANUAL.html#org-fancy-lists>.

`fencedCodeAttributes=true, false`

default: false

true Enable the Pandoc fenced code attribute syntax extension¹²:

```
~~~~ {#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 extension¹³:

```
::::: {#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
```

¹²See https://pandoc.org/MANUAL.html#extension-fenced_code_attributes.

¹³See 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 extension¹⁴:

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

# 123 Sesame Street
```

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

¹⁴See 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:nnn
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 extension¹⁵:

```

`<$>`{.haskell}

```

¹⁵See 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 extension¹⁶:

```
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 extension¹⁷ 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.
---
```

¹⁶See https://pandoc.org/MANUAL.html#extension-inline_notes.

¹⁷See 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 extension¹⁸:

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 extension¹⁹:

```
| 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
```

¹⁸See https://pandoc.org/MANUAL.html#extension-link_attributes.

¹⁹See https://pandoc.org/MANUAL.html#extension-line_blocks.

`mark=true, false` default: false

`true` Enable the Pandoc mark syntax extension²⁰:

```
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 extension²¹:

```
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
```

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

²¹See <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 extension²²:

```

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

```

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

```

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

```

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

```
 c & d
 \end{dcases}
\]

```

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

`false`      Disable the Pandoc raw attribute syntax extension.

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

`relativeReferences=true, false`

default: `false`

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

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

Conclusion {#conclusion}
=====

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

`false`      Disable relative references in autolinks.

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

---

<sup>23</sup>See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.



`shiftHeadings`=*<shift amount>* default: 0

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

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

655 defaultOptions.shiftHeadings = 0
```

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

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

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

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

```
656 \@@_add_lua_option:nnn
657 { slice }
658 { slice }
659 { ^-$ }

660 defaultOptions.slice = "^ $"
```

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

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

```

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

665 defaultOptions.smartEllipses = false

```

`startNumber=true, false`

default: true

- true**      Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererO1ItemWithNumber` T<sub>E</sub>X macro.
- false**     Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererO1Item` T<sub>E</sub>X macro.

```

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

670 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

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

This ~~is deleted text.~~

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

```

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

675 defaultOptions.strikeThrough = false

```

---

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

`stripIndent=true, false`

default: `false`

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

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

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

```
676 \@@_add_lua_option: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 extension²⁸:

```
- [ ] 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
```

²⁷See https://pandoc.org/MANUAL.html#extension-table_captions.

²⁸See https://pandoc.org/MANUAL.html#extension-task_lists.

`texComments=true, false`

default: false

true Strip T_EX-style comments.

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

Always enabled when `hybrid` is enabled.

false Do not strip T_EX-style comments.

```
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 extension²⁹:

```
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
```

²⁹See https://pandoc.org/MANUAL.html#extension-tex_math_dollars.

`texMathDoubleBackslash=true, false` default: false

true Enable the Pandoc double backslash math syntax extension³⁰:

```
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 extension³¹:

```
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:

³⁰See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.

³¹See 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 $\text{T}_{\text{E}}\text{X}$ layer hands markdown documents to the Lua layer. Lua converts the documents to $\text{T}_{\text{E}}\text{X}$, and hands the converted documents back to plain $\text{T}_{\text{E}}\text{X}$ layer for typesetting, see Figure 2.

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

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

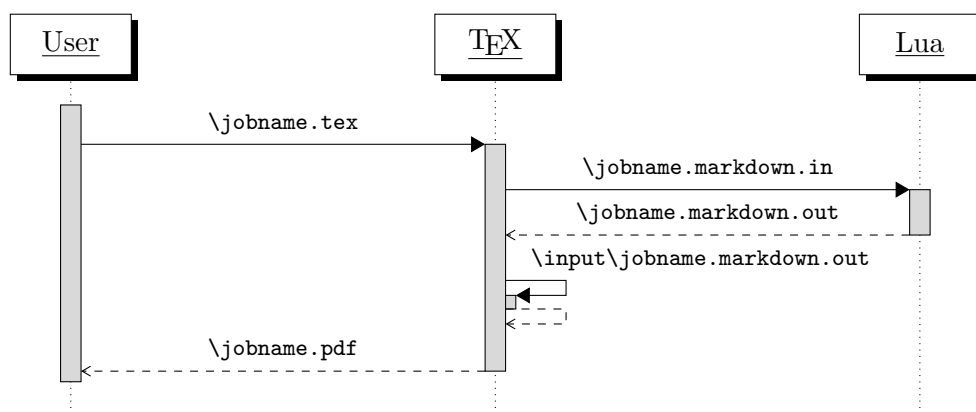


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the $\text{T}_{\text{E}}\text{X}$ interface

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

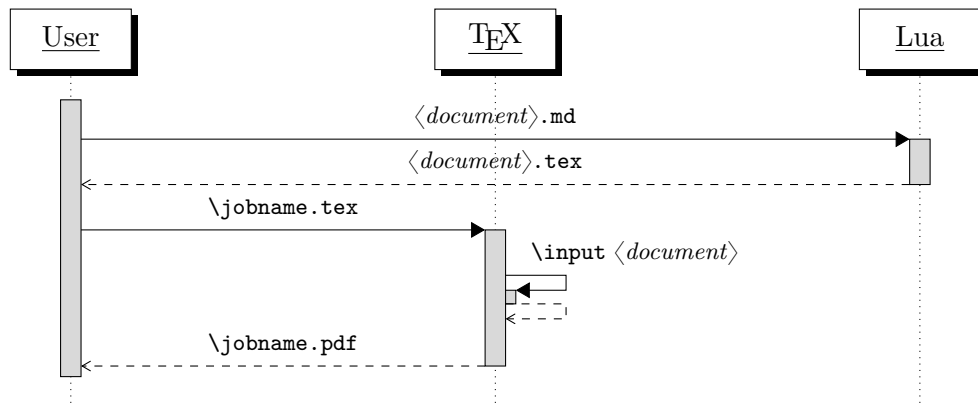


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

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

To make it easier to copy-and-paste options from Pandoc [5] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-Case variants of options. As a bonus, studies [6] also show that `snake_case` is faster to read than camelCase.

```

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

```

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

```

After the optional `--` argument has been specified, the remaining arguments are assumed to be input and output filenames. This argument is optional, but encouraged, because it helps resolve ambiguities when deciding whether an option or a filename has been specified.

```

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

```

Unless the `--` argument has been specified before, an argument containing the equals sign (`=`) is assumed to be an option specification in a `<key>=<value>` format. The available options are listed in Section 2.1.3.

```

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

```

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

```

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

```

```

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

```

Unless the `--` argument has been specified before, an argument `--help`, or `-h` causes a brief documentation for how to invoke the program to be printed to the standard output.

```

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

```

Unless the `--` argument has been specified before, an argument `--version`, or `-v` causes the program to print information about its name, version, origin and legal status, all on standard output.

```

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

```

The first argument that matches none of the above patterns is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a \TeX 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 \TeX 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_EX document `hello.tex`. After the Markdown package for our T_EX format has been loaded, the converted document can be typeset as follows:

```
\input hello
```

2.2 Plain T_EX Interface

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

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

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

```
\input markdown
```

It is expected that the special plain T_EX characters have the expected category codes, when `\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_EX [7, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```
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_EX 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_EX code showcases the usage of the `\markinline` macro:

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

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

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

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_EX primitive to include T_EX 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_EX.

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

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

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

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

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

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

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

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

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

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

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

2.2.1.3 Typesetting TeX from inside Markdown and YAML documents

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

```
858 \let\markdownEscape\relax
```


2.2.2 Options

The plain \TeX options are represented by \TeX commands. Some of them map directly to the options recognized by the Lua interface (see Section 2.1.3), while some of them are specific to the plain \TeX interface.

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

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

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

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

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

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

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

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

```

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

```

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

```

896 \cs_gset_eq:NN
897   \yamlSetup
898   \markdownSetup

```

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

```

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

```

```

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

```

2.2.2.1 Finalizing and Freezing the Cache

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

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

```

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

```

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

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

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

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

```

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

```

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

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

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

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

2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level \TeX formats such as \LaTeX and \ConTeXt . Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

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

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

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

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

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

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

The `\markdownOptionNoDefaults` macro specifies whether we should prevent the loading of default definitions or not. This is useful in contexts, where we want to have total control over how all elements are rendered.

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

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

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

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

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

```
949 \@@_add_plain_tex_option:nnn
950   { noDefaults }
951   { boolean }
952   { false }
```

2.2.2.4 Miscellaneous Options

The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see sections 3.2.5 and 3.2.6) or not. Notably, this enables the use of markdown when writing \TeX package documentation using the Doc \LaTeX package [8] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

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

2.2.2.5 Generating Plain T_EX Option Macros and Key-Values

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

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

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

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

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

To make it easier to copy-and-paste options from Pandoc [5] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-Case variants of options. As a bonus, studies [6] also show that `snake_case` is faster to read than `camelCase`.

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

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

```

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

```

Do not override options defined before loading the package.

```

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

```

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

```

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

```

```

1027         \sys_if_shell_unrestricted_p:
1028     )
1029     }
1030     {

```

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

```

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

```



```

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

```

```
1117         {
```

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

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

For options of type `clist`, we assume that $\langle key \rangle$ is a regular English noun in plural (such as `extensions`) and we also define the $\langle singular\ key \rangle = \langle value \rangle$ interface, where $\langle singular\ key \rangle$ is $\langle key \rangle$ after stripping the trailing -s (such as `extension`). Rather than setting the option to $\langle value \rangle$, this interface appends $\langle value \rangle$ to the current value as the rightmost item in the list.

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

```

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

```

```

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

```

If plain \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.

```

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

```

2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a \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 \LaTeX package, which provides similar functionality with its `\usetheme` macro [9, Section 15.1].

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

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

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

```

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

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

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

```

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

```

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

```

```

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

```

First, we validate the theme name.

```

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

```

Next, we extract the theme version.

```

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

```

```

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

```

Next, we munge the theme name.

```

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

```

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

```

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

```

```

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

```

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

```

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

```

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

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

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

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

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

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

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

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

```

\end{markdown}
\end{document}

```

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

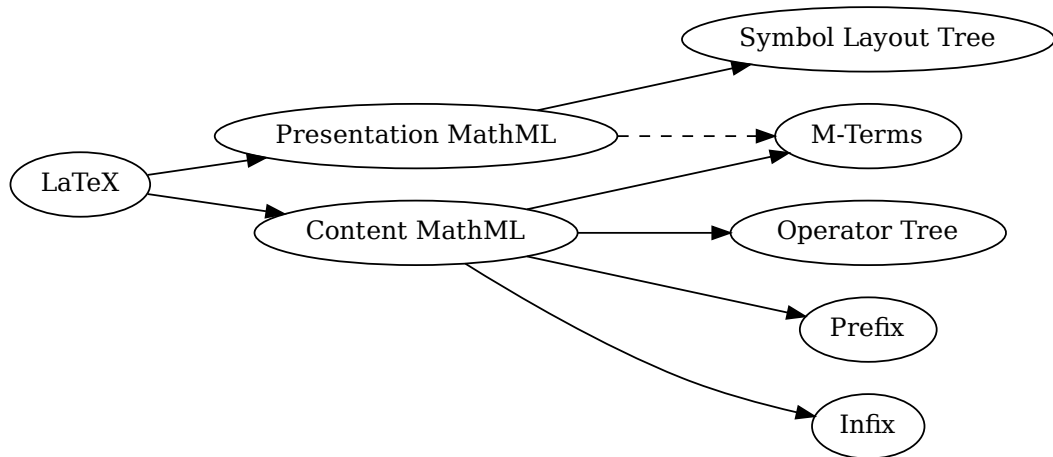


Figure 4: Various formats of mathematical formulae

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

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

```

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

```

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

```

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

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

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

```



Chapter 1

Introduction

1.1 Section
1.1.1 Subsection
Hello *Markdown*!

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

Table 1.1: Table

Figure 5: The banner of the Markdown package

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_EX theme with the default definitions of token renderer prototypes for plain T_EX. 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_EX themes.

2.2.4 Snippets

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

```
1381 \prop_new:N
1382   \g_@@_snippets_prop
1383 \cs_new:Nn
1384   \@@_setup_snippet:nn
1385   {
1386     \tl_if_empty:nT
1387       { #1 }
1388     {
1389       \msg_error:nnn
1390         { markdown }
1391         { empty-snippet-name }
1392         { #1 }
1393     }
1394     \tl_set:NV
1395       \l_tmpa_tl
1396       \g_@@_current_theme_tl
1397     \tl_put_right:Nn
1398       \l_tmpa_tl
1399       { #1 }
1400     \@@_if_snippet_exists:nT
1401       { #1 }
1402     {
1403       \msg_warning:nnV
1404         { markdown }
1405         { redefined-snippet }
1406         \l_tmpa_tl
1407     }
1408     \keys_precompile:nnN
1409       { markdown/options }
1410       { #2 }
1411     \l_tmpb_tl
1412     \prop_gput:NVV
1413       \g_@@_snippets_prop
1414       \l_tmpa_tl
1415       \l_tmpb_tl
1416   }
1417 \cs_gset_eq:NN
1418   \markdownSetupSnippet
1419   \@@_setup_snippet:nn
1420 \msg_new:nnnn
1421   { markdown }
1422   { empty-snippet-name }
```

```

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

```

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

```

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

```

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

```

1450 \keys_define:nn
1451   { markdown/options }
1452   {
1453     snippet .code:n = {
1454       \tl_set:NV
1455         \l_tmpa_tl
1456         \g_@@_current_theme_tl
1457       \tl_put_right:Nn
1458         \l_tmpa_tl
1459         { #1 }
1460       \@@_if_snippet_exists:nTF
1461         { #1 }
1462         {
1463           \prop_get:NVN
1464             \g_@@_snippets_prop
1465             \l_tmpa_tl

```

```

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

```

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

```

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

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}

```

If the `romanNumerals` snippet were defined in the `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` \LaTeX 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^AT_EX option:

```

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

```

```

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

```

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

```

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

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```

1491     \tl_set_eq:NN
1492     \l_@@_import_current_theme_tl
1493     \l_keys_key_str

```



```

1494     \tl_replace_all:NVn
1495     \l_@@_import_current_theme_tl
1496     \c_backslash_str
1497     { / }

```

Here, we import the snippets.

```

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

```

Here, we load the theme.

```

1534     \@@_set_theme:V
1535     \l_@@_import_current_theme_tl
1536 },
1537 }

```

```

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

```

2.2.5 Token Renderers

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

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

```

1547 \ExplSyntaxOn
1548 \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.

```

1549 \prop_new:N \g_@@_renderer_arities_prop
1550 \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.

```

1551 \ExplSyntaxOn
1552 \cs_gset_protected:Npn
1553   \markdownRendererAttributeIdentifier
1554   {
1555     \markdownRendererAttributeIdentifierPrototype
1556   }
1557 \seq_gput_right:Nn
1558   \g_@@_renderers_seq
1559   { attributeIdentifier }
1560 \prop_gput:Nnn
1561   \g_@@_renderer_arities_prop
1562   { attributeIdentifier }
1563   { 1 }
1564 \cs_gset_protected:Npn
1565   \markdownRendererAttributeClassName
1566   {
1567     \markdownRendererAttributeClassNamePrototype
1568   }
1569 \seq_gput_right:Nn
1570   \g_@@_renderers_seq
1571   { attributeClassName }
1572 \prop_gput:Nnn
1573   \g_@@_renderer_arities_prop
1574   { attributeClassName }
1575   { 1 }
1576 \cs_gset_protected:Npn
1577   \markdownRendererAttributeKeyValue
1578   {
1579     \markdownRendererAttributeKeyValuePrototype
1580   }
1581 \seq_gput_right:Nn
1582   \g_@@_renderers_seq
1583   { attributeKeyValue }
1584 \prop_gput:Nnn
1585   \g_@@_renderer_arities_prop
1586   { attributeKeyValue }
1587   { 2 }
1588 \ExplSyntaxOff

```

2.2.5.2 Block Quote Renderers

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

```

1589 \ExplSyntaxOn
1590 \cs_gset_protected:Npn
1591   \markdownRendererBlockQuoteBegin
1592   {
1593     \markdownRendererBlockQuoteBeginPrototype
1594   }
1595 \seq_gput_right:Nn
1596   \g_@@_renderers_seq
1597   { blockQuoteBegin }
1598 \prop_gput:Nnn
1599   \g_@@_renderer_arities_prop
1600   { blockQuoteBegin }
1601   { 0 }
1602 \ExplSyntaxOff

```

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

```

1603 \ExplSyntaxOn
1604 \cs_gset_protected:Npn
1605   \markdownRendererBlockQuoteEnd
1606   {
1607     \markdownRendererBlockQuoteEndPrototype
1608   }
1609 \seq_gput_right:Nn
1610   \g_@@_renderers_seq
1611   { blockQuoteEnd }
1612 \prop_gput:Nnn
1613   \g_@@_renderer_arities_prop
1614   { blockQuoteEnd }
1615   { 0 }
1616 \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.

```

1617 \ExplSyntaxOn
1618 \cs_gset_protected:Npn
1619   \markdownRendererBracketedSpanAttributeContextBegin
1620   {
1621     \markdownRendererBracketedSpanAttributeContextBeginPrototype
1622   }
1623 \seq_gput_right:Nn

```

```

1624 \g_@@_renderers_seq
1625 { bracketedSpanAttributeContextBegin }
1626 \prop_gput:Nnn
1627 \g_@@_renderer_arities_prop
1628 { bracketedSpanAttributeContextBegin }
1629 { 0 }
1630 \cs_gset_protected:Npn
1631 \markdownRendererBracketedSpanAttributeContextEnd
1632 {
1633   \markdownRendererBracketedSpanAttributeContextEndPrototype
1634 }
1635 \seq_gput_right:Nn
1636 \g_@@_renderers_seq
1637 { bracketedSpanAttributeContextEnd }
1638 \prop_gput:Nnn
1639 \g_@@_renderer_arities_prop
1640 { bracketedSpanAttributeContextEnd }
1641 { 0 }
1642 \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.

```

1643 \ExplSyntaxOn
1644 \cs_gset_protected:Npn
1645 \markdownRendererUlBegin
1646 {
1647   \markdownRendererUlBeginPrototype
1648 }
1649 \seq_gput_right:Nn
1650 \g_@@_renderers_seq
1651 { ulBegin }
1652 \prop_gput:Nnn
1653 \g_@@_renderer_arities_prop
1654 { ulBegin }
1655 { 0 }
1656 \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.

```

1657 \ExplSyntaxOn
1658 \cs_gset_protected:Npn

```

```

1659 \markdownRendererUlBeginTight
1660 {
1661   \markdownRendererUlBeginTightPrototype
1662 }
1663 \seq_gput_right:Nn
1664 \g_@@_renderers_seq
1665 { ulBeginTight }
1666 \prop_gput:Nnn
1667 \g_@@_renderer_arities_prop
1668 { ulBeginTight }
1669 { 0 }
1670 \ExplSyntaxOff

```

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

```

1671 \ExplSyntaxOn
1672 \cs_gset_protected:Npn
1673   \markdownRendererUlItem
1674   {
1675     \markdownRendererUlItemPrototype
1676   }
1677 \seq_gput_right:Nn
1678 \g_@@_renderers_seq
1679 { ulItem }
1680 \prop_gput:Nnn
1681 \g_@@_renderer_arities_prop
1682 { ulItem }
1683 { 0 }
1684 \ExplSyntaxOff

```

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

```

1685 \ExplSyntaxOn
1686 \cs_gset_protected:Npn
1687   \markdownRendererUlItemEnd
1688   {
1689     \markdownRendererUlItemEndPrototype
1690   }
1691 \seq_gput_right:Nn
1692 \g_@@_renderers_seq
1693 { ulItemEnd }
1694 \prop_gput:Nnn
1695 \g_@@_renderer_arities_prop
1696 { ulItemEnd }
1697 { 0 }
1698 \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.

```

1699 \ExplSyntaxOn
1700 \cs_gset_protected:Npn
1701   \markdownRendererUPEnd
1702   {
1703     \markdownRendererUPEndPrototype
1704   }
1705 \seq_gput_right:Nn
1706   \g_@@_renderers_seq
1707   { ulEnd }
1708 \prop_gput:Nnn
1709   \g_@@_renderer_arities_prop
1710   { ulEnd }
1711   { 0 }
1712 \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.

```

1713 \ExplSyntaxOn
1714 \cs_gset_protected:Npn
1715   \markdownRendererUPEndTight
1716   {
1717     \markdownRendererUPEndTightPrototype
1718   }
1719 \seq_gput_right:Nn
1720   \g_@@_renderers_seq
1721   { ulEndTight }
1722 \prop_gput:Nnn
1723   \g_@@_renderer_arities_prop
1724   { ulEndTight }
1725   { 0 }
1726 \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.

```

1727 \ExplSyntaxOn
1728 \cs_gset_protected:Npn
1729   \markdownRendererCite
1730   {
1731     \markdownRendererCitePrototype
1732   }
1733 \seq_gput_right:Nn
1734   \g_@@_renderers_seq
1735   { cite }
1736 \prop_gput:Nnn
1737   \g_@@_renderer_arities_prop
1738   { cite }
1739   { 1 }
1740 \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.

```

1741 \ExplSyntaxOn
1742 \cs_gset_protected:Npn
1743   \markdownRendererTextCite
1744   {
1745     \markdownRendererTextCitePrototype
1746   }
1747 \seq_gput_right:Nn
1748   \g_@@_renderers_seq
1749   { textCite }
1750 \prop_gput:Nnn
1751   \g_@@_renderer_arities_prop
1752   { textCite }
1753   { 1 }
1754 \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.

```

1755 \ExplSyntaxOn
1756 \cs_gset_protected:Npn
1757   \markdownRendererInputVerbatim
1758   {
1759     \markdownRendererInputVerbatimPrototype
1760   }
1761 \seq_gput_right:Nn

```



```

1762 \g_@@_renderers_seq
1763 { inputVerbatim }
1764 \prop_gput:Nnn
1765 \g_@@_renderer_arities_prop
1766 { inputVerbatim }
1767 { 1 }
1768 \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.

```

1769 \ExplSyntaxOn
1770 \cs_gset_protected:Npn
1771 \markdownRendererInputFencedCode
1772 {
1773   \markdownRendererInputFencedCodePrototype
1774 }
1775 \seq_gput_right:Nn
1776 \g_@@_renderers_seq
1777 { inputFencedCode }
1778 \prop_gput:Nnn
1779 \g_@@_renderer_arities_prop
1780 { inputFencedCode }
1781 { 3 }
1782 \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.

```

1783 \ExplSyntaxOn
1784 \cs_gset_protected:Npn
1785 \markdownRendererCodeSpan
1786 {
1787   \markdownRendererCodeSpanPrototype
1788 }
1789 \seq_gput_right:Nn
1790 \g_@@_renderers_seq
1791 { codeSpan }
1792 \prop_gput:Nnn
1793 \g_@@_renderer_arities_prop
1794 { codeSpan }
1795 { 1 }
1796 \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.

```
1797 \ExplSyntaxOn
1798 \cs_gset_protected:Npn
1799   \markdownRendererCodeSpanAttributeContextBegin
1800   {
1801     \markdownRendererCodeSpanAttributeContextBeginPrototype
1802   }
1803 \seq_gput_right:Nn
1804   \g_@@_renderers_seq
1805   { codeSpanAttributeContextBegin }
1806 \prop_gput:Nnn
1807   \g_@@_renderer_arities_prop
1808   { codeSpanAttributeContextBegin }
1809   { 0 }
1810 \cs_gset_protected:Npn
1811   \markdownRendererCodeSpanAttributeContextEnd
1812   {
1813     \markdownRendererCodeSpanAttributeContextEndPrototype
1814   }
1815 \seq_gput_right:Nn
1816   \g_@@_renderers_seq
1817   { codeSpanAttributeContextEnd }
1818 \prop_gput:Nnn
1819   \g_@@_renderer_arities_prop
1820   { codeSpanAttributeContextEnd }
1821   { 0 }
1822 \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.

```
1823 \ExplSyntaxOn
1824 \cs_gset_protected:Npn
1825   \markdownRendererContentBlock
1826   {
1827     \markdownRendererContentBlockPrototype
1828   }
1829 \seq_gput_right:Nn
```

```

1830 \g_@@_renderers_seq
1831 { contentBlock }
1832 \prop_gput:Nnn
1833 \g_@@_renderer_arities_prop
1834 { contentBlock }
1835 { 4 }
1836 \ExplSyntaxOff

```

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

```

1837 \ExplSyntaxOn
1838 \cs_gset_protected:Npn
1839 \markdownRendererContentBlockOnlineImage
1840 {
1841 \markdownRendererContentBlockOnlineImagePrototype
1842 }
1843 \seq_gput_right:Nn
1844 \g_@@_renderers_seq
1845 { contentBlockOnlineImage }
1846 \prop_gput:Nnn
1847 \g_@@_renderer_arities_prop
1848 { contentBlockOnlineImage }
1849 { 4 }
1850 \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`³² 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 T_EX 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 [4] is a good starting point.

```

1851 \ExplSyntaxOn
1852 \cs_gset_protected:Npn

```

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

```

1853 \markdownRendererContentBlockCode
1854 {
1855   \markdownRendererContentBlockCodePrototype
1856 }
1857 \seq_gput_right:Nn
1858 \g_@@_renderers_seq
1859 { contentBlockCode }
1860 \prop_gput:Nnn
1861 \g_@@_renderer_arities_prop
1862 { contentBlockCode }
1863 { 5 }
1864 \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.

```

1865 \ExplSyntaxOn
1866 \cs_gset_protected:Npn
1867 \markdownRendererDlBegin
1868 {
1869   \markdownRendererDlBeginPrototype
1870 }
1871 \seq_gput_right:Nn
1872 \g_@@_renderers_seq
1873 { dlBegin }
1874 \prop_gput:Nnn
1875 \g_@@_renderer_arities_prop
1876 { dlBegin }
1877 { 0 }
1878 \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.

```

1879 \ExplSyntaxOn
1880 \cs_gset_protected:Npn
1881 \markdownRendererDlBeginTight
1882 {
1883   \markdownRendererDlBeginTightPrototype
1884 }
1885 \seq_gput_right:Nn

```

```

1886 \g_@@_renderers_seq
1887 { dlBeginTight }
1888 \prop_gput:Nnn
1889 \g_@@_renderer_arities_prop
1890 { dlBeginTight }
1891 { 0 }
1892 \ExplSyntaxOff

```

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

```

1893 \ExplSyntaxOn
1894 \cs_gset_protected:Npn
1895 \markdownRendererDlItem
1896 {
1897 \markdownRendererDlItemPrototype
1898 }
1899 \seq_gput_right:Nn
1900 \g_@@_renderers_seq
1901 { dlItem }
1902 \prop_gput:Nnn
1903 \g_@@_renderer_arities_prop
1904 { dlItem }
1905 { 1 }
1906 \ExplSyntaxOff

```

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

```

1907 \ExplSyntaxOn
1908 \cs_gset_protected:Npn
1909 \markdownRendererDlItemEnd
1910 {
1911 \markdownRendererDlItemEndPrototype
1912 }
1913 \seq_gput_right:Nn
1914 \g_@@_renderers_seq
1915 { dlItemEnd }
1916 \prop_gput:Nnn
1917 \g_@@_renderer_arities_prop
1918 { dlItemEnd }
1919 { 0 }
1920 \ExplSyntaxOff

```

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

```

1921 \ExplSyntaxOn
1922 \cs_gset_protected:Npn
1923 \markdownRendererDlDefinitionBegin

```

```

1924 {
1925   \markdownRendererDlDefinitionBeginPrototype
1926 }
1927 \seq_gput_right:Nn
1928 \g_@@_renderers_seq
1929 { dlDefinitionBegin }
1930 \prop_gput:Nnn
1931 \g_@@_renderer_arities_prop
1932 { dlDefinitionBegin }
1933 { 0 }
1934 \ExplSyntaxOff

```

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

```

1935 \ExplSyntaxOn
1936 \cs_gset_protected:Npn
1937   \markdownRendererDlDefinitionEnd
1938   {
1939     \markdownRendererDlDefinitionEndPrototype
1940   }
1941 \seq_gput_right:Nn
1942 \g_@@_renderers_seq
1943 { dlDefinitionEnd }
1944 \prop_gput:Nnn
1945 \g_@@_renderer_arities_prop
1946 { dlDefinitionEnd }
1947 { 0 }
1948 \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.

```

1949 \ExplSyntaxOn
1950 \cs_gset_protected:Npn
1951   \markdownRendererDlEnd
1952   {
1953     \markdownRendererDlEndPrototype
1954   }
1955 \seq_gput_right:Nn
1956 \g_@@_renderers_seq
1957 { dlEnd }
1958 \prop_gput:Nnn
1959 \g_@@_renderer_arities_prop
1960 { dlEnd }
1961 { 0 }
1962 \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.

```

1963 \ExplSyntaxOn
1964 \cs_gset_protected:Npn
1965   \markdownRendererDlEndTight
1966   {
1967     \markdownRendererDlEndTightPrototype
1968   }
1969 \seq_gput_right:Nn
1970   \g_@@_renderers_seq
1971   { dlEndTight }
1972 \prop_gput:Nnn
1973   \g_@@_renderer_arities_prop
1974   { dlEndTight }
1975   { 0 }
1976 \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.

```

1977 \ExplSyntaxOn
1978 \cs_gset_protected:Npn
1979   \markdownRendererEllipsis
1980   {
1981     \markdownRendererEllipsisPrototype
1982   }
1983 \seq_gput_right:Nn
1984   \g_@@_renderers_seq
1985   { ellipsis }
1986 \prop_gput:Nnn
1987   \g_@@_renderer_arities_prop
1988   { ellipsis }
1989   { 0 }
1990 \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.

```

1991 \ExplSyntaxOn
1992 \cs_gset_protected:Npn

```

```

1993 \markdownRendererEmphasis
1994 {
1995   \markdownRendererEmphasisPrototype
1996 }
1997 \seq_gput_right:Nn
1998 \g_@@_renderers_seq
1999 { emphasis }
2000 \prop_gput:Nnn
2001 \g_@@_renderer_arities_prop
2002 { emphasis }
2003 { 1 }
2004 \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.

```

2005 \ExplSyntaxOn
2006 \cs_gset_protected:Npn
2007   \markdownRendererStrongEmphasis
2008   {
2009     \markdownRendererStrongEmphasisPrototype
2010   }
2011 \seq_gput_right:Nn
2012 \g_@@_renderers_seq
2013 { strongEmphasis }
2014 \prop_gput:Nnn
2015 \g_@@_renderer_arities_prop
2016 { strongEmphasis }
2017 { 1 }
2018 \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.

```

2019 \ExplSyntaxOn
2020 \cs_gset_protected:Npn
2021   \markdownRendererFencedCodeAttributeContextBegin
2022   {
2023     \markdownRendererFencedCodeAttributeContextBeginPrototype
2024   }
2025 \seq_gput_right:Nn
2026 \g_@@_renderers_seq

```



```

2027 { fencedCodeAttributeContextBegin }
2028 \prop_gput:Nnn
2029 \g_@@_renderer_arities_prop
2030 { fencedCodeAttributeContextBegin }
2031 { 0 }
2032 \cs_gset_protected:Npn
2033 \markdownRendererFencedCodeAttributeContextEnd
2034 {
2035   \markdownRendererFencedCodeAttributeContextEndPrototype
2036 }
2037 \seq_gput_right:Nn
2038 \g_@@_renderers_seq
2039 { fencedCodeAttributeContextEnd }
2040 \prop_gput:Nnn
2041 \g_@@_renderer_arities_prop
2042 { fencedCodeAttributeContextEnd }
2043 { 0 }
2044 \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.

```

2045 \ExplSyntaxOn
2046 \cs_gset_protected:Npn
2047 \markdownRendererFencedDivAttributeContextBegin
2048 {
2049   \markdownRendererFencedDivAttributeContextBeginPrototype
2050 }
2051 \seq_gput_right:Nn
2052 \g_@@_renderers_seq
2053 { fencedDivAttributeContextBegin }
2054 \prop_gput:Nnn
2055 \g_@@_renderer_arities_prop
2056 { fencedDivAttributeContextBegin }
2057 { 0 }
2058 \cs_gset_protected:Npn
2059 \markdownRendererFencedDivAttributeContextEnd
2060 {
2061   \markdownRendererFencedDivAttributeContextEndPrototype
2062 }
2063 \seq_gput_right:Nn
2064 \g_@@_renderers_seq
2065 { fencedDivAttributeContextEnd }
2066 \prop_gput:Nnn

```

```

2067 \g_@@_renderer_arities_prop
2068 { fencedDivAttributeContextEnd }
2069 { 0 }
2070 \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.

```

2071 \ExplSyntaxOn
2072 \cs_gset_protected:Npn
2073 \markdownRendererHeaderAttributeContextBegin
2074 {
2075   \markdownRendererHeaderAttributeContextBeginPrototype
2076 }
2077 \seq_gput_right:Nn
2078 \g_@@_renderers_seq
2079 { headerAttributeContextBegin }
2080 \prop_gput:Nnn
2081 \g_@@_renderer_arities_prop
2082 { headerAttributeContextBegin }
2083 { 0 }
2084 \cs_gset_protected:Npn
2085 \markdownRendererHeaderAttributeContextEnd
2086 {
2087   \markdownRendererHeaderAttributeContextEndPrototype
2088 }
2089 \seq_gput_right:Nn
2090 \g_@@_renderers_seq
2091 { headerAttributeContextEnd }
2092 \prop_gput:Nnn
2093 \g_@@_renderer_arities_prop
2094 { headerAttributeContextEnd }
2095 { 0 }
2096 \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.

```

2097 \ExplSyntaxOn
2098 \cs_gset_protected:Npn
2099 \markdownRendererHeadingOne

```

```

2100 {
2101   \markdownRendererHeadingOnePrototype
2102 }
2103 \seq_gput_right:Nn
2104 \g_@@_renderers_seq
2105 { headingOne }
2106 \prop_gput:Nnn
2107 \g_@@_renderer_arities_prop
2108 { headingOne }
2109 { 1 }
2110 \ExplSyntaxOff

```

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

```

2111 \ExplSyntaxOn
2112 \cs_gset_protected:Npn
2113   \markdownRendererHeadingTwo
2114   {
2115     \markdownRendererHeadingTwoPrototype
2116   }
2117 \seq_gput_right:Nn
2118 \g_@@_renderers_seq
2119 { headingTwo }
2120 \prop_gput:Nnn
2121 \g_@@_renderer_arities_prop
2122 { headingTwo }
2123 { 1 }
2124 \ExplSyntaxOff

```

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

```

2125 \ExplSyntaxOn
2126 \cs_gset_protected:Npn
2127   \markdownRendererHeadingThree
2128   {
2129     \markdownRendererHeadingThreePrototype
2130   }
2131 \seq_gput_right:Nn
2132 \g_@@_renderers_seq
2133 { headingThree }
2134 \prop_gput:Nnn
2135 \g_@@_renderer_arities_prop
2136 { headingThree }
2137 { 1 }
2138 \ExplSyntaxOff

```

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

```
2139 \ExplSyntaxOn
2140 \cs_gset_protected:Npn
2141   \markdownRendererHeadingFour
2142   {
2143     \markdownRendererHeadingFourPrototype
2144   }
2145 \seq_gput_right:Nn
2146   \g_@@_renderers_seq
2147   { headingFour }
2148 \prop_gput:Nnn
2149   \g_@@_renderer_arities_prop
2150   { headingFour }
2151   { 1 }
2152 \ExplSyntaxOff
```

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

```
2153 \ExplSyntaxOn
2154 \cs_gset_protected:Npn
2155   \markdownRendererHeadingFive
2156   {
2157     \markdownRendererHeadingFivePrototype
2158   }
2159 \seq_gput_right:Nn
2160   \g_@@_renderers_seq
2161   { headingFive }
2162 \prop_gput:Nnn
2163   \g_@@_renderer_arities_prop
2164   { headingFive }
2165   { 1 }
2166 \ExplSyntaxOff
```

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

```
2167 \ExplSyntaxOn
2168 \cs_gset_protected:Npn
2169   \markdownRendererHeadingSix
2170   {
2171     \markdownRendererHeadingSixPrototype
2172   }
2173 \seq_gput_right:Nn
2174   \g_@@_renderers_seq
2175   { headingSix }
2176 \prop_gput:Nnn
```

```

2177 \g_@@_renderer_arities_prop
2178 { headingSix }
2179 { 1 }
2180 \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.

```

2181 \ExplSyntaxOn
2182 \cs_gset_protected:Npn
2183 \markdownRendererInlineHtmlComment
2184 {
2185   \markdownRendererInlineHtmlCommentPrototype
2186 }
2187 \seq_gput_right:Nn
2188 \g_@@_renderers_seq
2189 { inlineHtmlComment }
2190 \prop_gput:Nnn
2191 \g_@@_renderer_arities_prop
2192 { inlineHtmlComment }
2193 { 1 }
2194 \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.

```

2195 \ExplSyntaxOn
2196 \cs_gset_protected:Npn
2197 \markdownRendererInlineHtmlTag
2198 {
2199   \markdownRendererInlineHtmlTagPrototype
2200 }
2201 \seq_gput_right:Nn
2202 \g_@@_renderers_seq
2203 { inlineHtmlTag }
2204 \prop_gput:Nnn

```

```

2205 \g_@@_renderer_arities_prop
2206 { inlineHtmlTag }
2207 { 1 }
2208 \cs_gset_protected:Npn
2209 \markdownRendererInputBlockHtmlElement
2210 {
2211   \markdownRendererInputBlockHtmlElementPrototype
2212 }
2213 \seq_gput_right:Nn
2214 \g_@@_renderers_seq
2215 { inputBlockHtmlElement }
2216 \prop_gput:Nnn
2217 \g_@@_renderer_arities_prop
2218 { inputBlockHtmlElement }
2219 { 1 }
2220 \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.

```

2221 \ExplSyntaxOn
2222 \cs_gset_protected:Npn
2223 \markdownRendererImage
2224 {
2225   \markdownRendererImagePrototype
2226 }
2227 \seq_gput_right:Nn
2228 \g_@@_renderers_seq
2229 { image }
2230 \prop_gput:Nnn
2231 \g_@@_renderer_arities_prop
2232 { image }
2233 { 4 }
2234 \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.

```

2235 \ExplSyntaxOn
2236 \cs_gset_protected:Npn

```

```

2237 \markdownRendererImageAttributeContextBegin
2238 {
2239   \markdownRendererImageAttributeContextBeginPrototype
2240 }
2241 \seq_gput_right:Nn
2242 \g_@@_renderers_seq
2243 { imageAttributeContextBegin }
2244 \prop_gput:Nnn
2245 \g_@@_renderer_arities_prop
2246 { imageAttributeContextBegin }
2247 { 0 }
2248 \cs_gset_protected:Npn
2249 \markdownRendererImageAttributeContextEnd
2250 {
2251   \markdownRendererImageAttributeContextEndPrototype
2252 }
2253 \seq_gput_right:Nn
2254 \g_@@_renderers_seq
2255 { imageAttributeContextEnd }
2256 \prop_gput:Nnn
2257 \g_@@_renderer_arities_prop
2258 { imageAttributeContextEnd }
2259 { 0 }
2260 \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.

```

2261 \ExplSyntaxOn
2262 \cs_gset_protected:Npn
2263 \markdownRendererInterblockSeparator
2264 {
2265   \markdownRendererInterblockSeparatorPrototype
2266 }
2267 \seq_gput_right:Nn
2268 \g_@@_renderers_seq
2269 { interblockSeparator }
2270 \prop_gput:Nnn
2271 \g_@@_renderer_arities_prop
2272 { interblockSeparator }
2273 { 0 }
2274 \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.

```
2275 \ExplSyntaxOn
2276 \cs_gset_protected:Npn
2277   \markdownRendererParagraphSeparator
2278   {
2279     \markdownRendererParagraphSeparatorPrototype
2280   }
2281 \seq_gput_right:Nn
2282   \g_@@_renderers_seq
2283   { paragraphSeparator }
2284 \prop_gput:Nnn
2285   \g_@@_renderer_arities_prop
2286   { paragraphSeparator }
2287   { 0 }
2288 \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.

```
2289 \ExplSyntaxOn
2290 \cs_gset_protected:Npn
2291   \markdownRendererLineBlockBegin
2292   {
2293     \markdownRendererLineBlockBeginPrototype
2294   }
2295 \seq_gput_right:Nn
2296   \g_@@_renderers_seq
2297   { lineBlockBegin }
2298 \prop_gput:Nnn
2299   \g_@@_renderer_arities_prop
2300   { lineBlockBegin }
2301   { 0 }
2302 \cs_gset_protected:Npn
2303   \markdownRendererLineBlockEnd
2304   {
2305     \markdownRendererLineBlockEndPrototype
2306   }
2307 \seq_gput_right:Nn
2308   \g_@@_renderers_seq
2309   { lineBlockEnd }
```



```

2310 \prop_gput:Nnn
2311   \g_@@_renderer_arities_prop
2312   { lineBlockEnd }
2313   { 0 }
2314 \ExplSyntaxOff

```

2.2.5.23 Line Break Renderers

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

```

2315 \ExplSyntaxOn
2316 \cs_gset_protected:Npn
2317   \markdownRendererSoftLineBreak
2318   {
2319     \markdownRendererSoftLineBreakPrototype
2320   }
2321 \seq_gput_right:Nn
2322   \g_@@_renderers_seq
2323   { softLineBreak }
2324 \prop_gput:Nnn
2325   \g_@@_renderer_arities_prop
2326   { softLineBreak }
2327   { 0 }
2328 \ExplSyntaxOff

```

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

```

2329 \ExplSyntaxOn
2330 \cs_gset_protected:Npn
2331   \markdownRendererHardLineBreak
2332   {
2333     \markdownRendererHardLineBreakPrototype
2334   }
2335 \seq_gput_right:Nn
2336   \g_@@_renderers_seq
2337   { hardLineBreak }
2338 \prop_gput:Nnn
2339   \g_@@_renderer_arities_prop
2340   { hardLineBreak }
2341   { 0 }
2342 \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.

```

2343 \ExplSyntaxOn
2344 \cs_gset_protected:Npn
2345   \markdownRendererLink
2346   {
2347     \markdownRendererLinkPrototype
2348   }
2349 \seq_gput_right:Nn
2350   \g_@@_renderers_seq
2351   { link }
2352 \prop_gput:Nnn
2353   \g_@@_renderer_arities_prop
2354   { link }
2355   { 4 }
2356 \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.

```

2357 \ExplSyntaxOn
2358 \cs_gset_protected:Npn
2359   \markdownRendererLinkAttributeContextBegin
2360   {
2361     \markdownRendererLinkAttributeContextBeginPrototype
2362   }
2363 \seq_gput_right:Nn
2364   \g_@@_renderers_seq
2365   { linkAttributeContextBegin }
2366 \prop_gput:Nnn
2367   \g_@@_renderer_arities_prop
2368   { linkAttributeContextBegin }
2369   { 0 }
2370 \cs_gset_protected:Npn
2371   \markdownRendererLinkAttributeContextEnd
2372   {
2373     \markdownRendererLinkAttributeContextEndPrototype
2374   }
2375 \seq_gput_right:Nn
2376   \g_@@_renderers_seq
2377   { linkAttributeContextEnd }
2378 \prop_gput:Nnn
2379   \g_@@_renderer_arities_prop
2380   { linkAttributeContextEnd }
2381   { 0 }

```

```
2382 \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.

```
2383 \ExplSyntaxOn
2384 \cs_gset_protected:Npn
2385   \markdownRendererMark
2386   {
2387     \markdownRendererMarkPrototype
2388   }
2389 \seq_gput_right:Nn
2390   \g_@@_renderers_seq
2391   { mark }
2392 \prop_gput:Nnn
2393   \g_@@_renderer_arities_prop
2394   { mark }
2395   { 1 }
2396 \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.

```
2397 \ExplSyntaxOn
2398 \cs_gset_protected:Npn
2399   \markdownRendererDocumentBegin
2400   {
2401     \markdownRendererDocumentBeginPrototype
2402   }
2403 \seq_gput_right:Nn
2404   \g_@@_renderers_seq
2405   { documentBegin }
2406 \prop_gput:Nnn
2407   \g_@@_renderer_arities_prop
2408   { documentBegin }
2409   { 0 }
2410 \cs_gset_protected:Npn
2411   \markdownRendererDocumentEnd
```

```

2412 {
2413   \markdownRendererDocumentEndPrototype
2414 }
2415 \seq_gput_right:Nn
2416   \g_@@_renderers_seq
2417   { documentEnd }
2418 \prop_gput:Nnn
2419   \g_@@_renderer_arities_prop
2420   { documentEnd }
2421   { 0 }
2422 \ExplSyntaxOff

```

2.2.5.28 Non-Breaking Space Renderer

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

```

2423 \ExplSyntaxOn
2424 \cs_gset_protected:Npn
2425   \markdownRendererNbsp
2426   {
2427     \markdownRendererNbspPrototype
2428   }
2429 \seq_gput_right:Nn
2430   \g_@@_renderers_seq
2431   { nbsp }
2432 \prop_gput:Nnn
2433   \g_@@_renderer_arities_prop
2434   { nbsp }
2435   { 0 }
2436 \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.

```

2437 \def\markdownRendererNote{%
2438   \markdownRendererNotePrototype}%
2439 \ExplSyntaxOn
2440 \seq_gput_right:Nn
2441   \g_@@_renderers_seq
2442   { note }
2443 \prop_gput:Nnn
2444   \g_@@_renderer_arities_prop
2445   { note }
2446   { 1 }
2447 \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.

```
2448 \ExplSyntaxOn
2449 \cs_gset_protected:Npn
2450   \markdownRendererOlBegin
2451   {
2452     \markdownRendererOlBeginPrototype
2453   }
2454 \seq_gput_right:Nn
2455   \g_@@_renderers_seq
2456   { olBegin }
2457 \prop_gput:Nnn
2458   \g_@@_renderer_arities_prop
2459   { olBegin }
2460   { 0 }
2461 \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.

```
2462 \ExplSyntaxOn
2463 \cs_gset_protected:Npn
2464   \markdownRendererOlBeginTight
2465   {
2466     \markdownRendererOlBeginTightPrototype
2467   }
2468 \seq_gput_right:Nn
2469   \g_@@_renderers_seq
2470   { olBeginTight }
2471 \prop_gput:Nnn
2472   \g_@@_renderer_arities_prop
2473   { olBeginTight }
2474   { 0 }
2475 \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`).

```

2476 \ExplSyntaxOn
2477 \cs_gset_protected:Npn
2478   \markdownRendererFancyOlBegin
2479   {
2480     \markdownRendererFancyOlBeginPrototype
2481   }
2482 \seq_gput_right:Nn
2483   \g_@@_renderers_seq
2484   { fancyOlBegin }
2485 \prop_gput:Nnn
2486   \g_@@_renderer_arities_prop
2487   { fancyOlBegin }
2488   { 2 }
2489 \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.

```

2490 \ExplSyntaxOn
2491 \cs_gset_protected:Npn
2492   \markdownRendererFancyOlBeginTight
2493   {
2494     \markdownRendererFancyOlBeginTightPrototype
2495   }
2496 \seq_gput_right:Nn
2497   \g_@@_renderers_seq
2498   { fancyOlBeginTight }
2499 \prop_gput:Nnn
2500   \g_@@_renderer_arities_prop
2501   { fancyOlBeginTight }
2502   { 2 }
2503 \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.

```

2504 \ExplSyntaxOn
2505 \cs_gset_protected:Npn
2506   \markdownRendererOlItem
2507   {
2508     \markdownRendererOlItemPrototype
2509   }
2510 \seq_gput_right:Nn

```

```

2511 \g_@@_renderers_seq
2512 { olItem }
2513 \prop_gput:Nnn
2514 \g_@@_renderer_arities_prop
2515 { olItem }
2516 { 0 }
2517 \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.

```

2518 \ExplSyntaxOn
2519 \cs_gset_protected:Npn
2520 \markdownRendererOlItemEnd
2521 {
2522   \markdownRendererOlItemEndPrototype
2523 }
2524 \seq_gput_right:Nn
2525 \g_@@_renderers_seq
2526 { olItemEnd }
2527 \prop_gput:Nnn
2528 \g_@@_renderer_arities_prop
2529 { olItemEnd }
2530 { 0 }
2531 \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.

```

2532 \ExplSyntaxOn
2533 \cs_gset_protected:Npn
2534 \markdownRendererOlItemWithNumber
2535 {
2536   \markdownRendererOlItemWithNumberPrototype
2537 }
2538 \seq_gput_right:Nn
2539 \g_@@_renderers_seq
2540 { olItemWithNumber }
2541 \prop_gput:Nnn
2542 \g_@@_renderer_arities_prop
2543 { olItemWithNumber }
2544 { 1 }
2545 \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.

```

2546 \ExplSyntaxOn
2547 \cs_gset_protected:Npn
2548   \markdownRendererFancy01Item
2549   {
2550     \markdownRendererFancy01ItemPrototype
2551   }
2552 \seq_gput_right:Nn
2553   \g_@@_renderers_seq
2554   { fancy01Item }
2555 \prop_gput:Nnn
2556   \g_@@_renderer_arities_prop
2557   { fancy01Item }
2558   { 0 }
2559 \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.

```

2560 \ExplSyntaxOn
2561 \cs_gset_protected:Npn
2562   \markdownRendererFancy01ItemEnd
2563   {
2564     \markdownRendererFancy01ItemEndPrototype
2565   }
2566 \seq_gput_right:Nn
2567   \g_@@_renderers_seq
2568   { fancy01ItemEnd }
2569 \prop_gput:Nnn
2570   \g_@@_renderer_arities_prop
2571   { fancy01ItemEnd }
2572   { 0 }
2573 \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.

```

2574 \ExplSyntaxOn
2575 \cs_gset_protected:Npn
2576   \markdownRendererFancy01ItemWithNumber
2577   {
2578     \markdownRendererFancy01ItemWithNumberPrototype
2579   }

```



```

2580 \seq_gput_right:Nn
2581   \g_@@_renderers_seq
2582   { fancyO1ItemWithNumber }
2583 \prop_gput:Nnn
2584   \g_@@_renderer_arities_prop
2585   { fancyO1ItemWithNumber }
2586   { 1 }
2587 \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.

```

2588 \ExplSyntaxOn
2589 \cs_gset_protected:Npn
2590   \markdownRendererO1End
2591   {
2592     \markdownRendererO1EndPrototype
2593   }
2594 \seq_gput_right:Nn
2595   \g_@@_renderers_seq
2596   { olEnd }
2597 \prop_gput:Nnn
2598   \g_@@_renderer_arities_prop
2599   { olEnd }
2600   { 0 }
2601 \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.

```

2602 \ExplSyntaxOn
2603 \cs_gset_protected:Npn
2604   \markdownRendererO1EndTight
2605   {
2606     \markdownRendererO1EndTightPrototype
2607   }
2608 \seq_gput_right:Nn
2609   \g_@@_renderers_seq
2610   { olEndTight }
2611 \prop_gput:Nnn
2612   \g_@@_renderer_arities_prop
2613   { olEndTight }
2614   { 0 }
2615 \ExplSyntaxOff

```

The `\markdownRendererFancy01End` macro represents the end of a fancy ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is enabled. The macro receives no arguments.

```

2616 \ExplSyntaxOn
2617 \cs_gset_protected:Npn
2618   \markdownRendererFancy01End
2619   {
2620     \markdownRendererFancy01EndPrototype
2621   }
2622 \seq_gput_right:Nn
2623   \g_@@_renderers_seq
2624   { fancy01End }
2625 \prop_gput:Nnn
2626   \g_@@_renderer_arities_prop
2627   { fancy01End }
2628   { 0 }
2629 \ExplSyntaxOff

```

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

```

2630 \ExplSyntaxOn
2631 \cs_gset_protected:Npn
2632   \markdownRendererFancy01EndTight
2633   {
2634     \markdownRendererFancy01EndTightPrototype
2635   }
2636 \seq_gput_right:Nn
2637   \g_@@_renderers_seq
2638   { fancy01EndTight }
2639 \prop_gput:Nnn
2640   \g_@@_renderer_arities_prop
2641   { fancy01EndTight }
2642   { 0 }
2643 \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.

```

2644 \ExplSyntaxOn

```

```

2645 \cs_gset_protected:Npn
2646   \markdownRendererInputRawInline
2647   {
2648     \markdownRendererInputRawInlinePrototype
2649   }
2650 \seq_gput_right:Nn
2651   \g_@@_renderers_seq
2652   { inputRawInline }
2653 \prop_gput:Nnn
2654   \g_@@_renderer_arities_prop
2655   { inputRawInline }
2656   { 2 }
2657 \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.

```

2658 \ExplSyntaxOn
2659 \cs_gset_protected:Npn
2660   \markdownRendererInputRawBlock
2661   {
2662     \markdownRendererInputRawBlockPrototype
2663   }
2664 \seq_gput_right:Nn
2665   \g_@@_renderers_seq
2666   { inputRawBlock }
2667 \prop_gput:Nnn
2668   \g_@@_renderer_arities_prop
2669   { inputRawBlock }
2670   { 2 }
2671 \ExplSyntaxOff

```

2.2.5.32 Section Renderers

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

```

2672 \ExplSyntaxOn
2673 \cs_gset_protected:Npn
2674   \markdownRendererSectionBegin
2675   {
2676     \markdownRendererSectionBeginPrototype
2677   }
2678 \seq_gput_right:Nn
2679   \g_@@_renderers_seq
2680   { sectionBegin }
2681 \prop_gput:Nnn

```

```

2682 \g_@@_renderer_arities_prop
2683 { sectionBegin }
2684 { 0 }
2685 \cs_gset_protected:Npn
2686 \markdownRendererSectionEnd
2687 {
2688   \markdownRendererSectionEndPrototype
2689 }
2690 \seq_gput_right:Nn
2691 \g_@@_renderers_seq
2692 { sectionEnd }
2693 \prop_gput:Nnn
2694 \g_@@_renderer_arities_prop
2695 { sectionEnd }
2696 { 0 }
2697 \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.

```

2698 \ExplSyntaxOn
2699 \cs_gset_protected:Npn
2700 \markdownRendererReplacementCharacter
2701 {
2702   \markdownRendererReplacementCharacterPrototype
2703 }
2704 \seq_gput_right:Nn
2705 \g_@@_renderers_seq
2706 { replacementCharacter }
2707 \prop_gput:Nnn
2708 \g_@@_renderer_arities_prop
2709 { replacementCharacter }
2710 { 0 }
2711 \ExplSyntaxOff

```

2.2.5.34 Special Character Renderers

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

```

2712 \ExplSyntaxOn
2713 \cs_gset_protected:Npn
2714 \markdownRendererLeftBrace
2715 {
2716   \markdownRendererLeftBracePrototype
2717 }

```

```

2718 \seq_gput_right:Nn
2719   \g_@@_renderers_seq
2720   { leftBrace }
2721 \prop_gput:Nnn
2722   \g_@@_renderer_arities_prop
2723   { leftBrace }
2724   { 0 }
2725 \cs_gset_protected:Npn
2726   \markdownRendererRightBrace
2727   {
2728     \markdownRendererRightBracePrototype
2729   }
2730 \seq_gput_right:Nn
2731   \g_@@_renderers_seq
2732   { rightBrace }
2733 \prop_gput:Nnn
2734   \g_@@_renderer_arities_prop
2735   { rightBrace }
2736   { 0 }
2737 \cs_gset_protected:Npn
2738   \markdownRendererDollarSign
2739   {
2740     \markdownRendererDollarSignPrototype
2741   }
2742 \seq_gput_right:Nn
2743   \g_@@_renderers_seq
2744   { dollarSign }
2745 \prop_gput:Nnn
2746   \g_@@_renderer_arities_prop
2747   { dollarSign }
2748   { 0 }
2749 \cs_gset_protected:Npn
2750   \markdownRendererPercentSign
2751   {
2752     \markdownRendererPercentSignPrototype
2753   }
2754 \seq_gput_right:Nn
2755   \g_@@_renderers_seq
2756   { percentSign }
2757 \prop_gput:Nnn
2758   \g_@@_renderer_arities_prop
2759   { percentSign }
2760   { 0 }
2761 \cs_gset_protected:Npn
2762   \markdownRendererAmpersand
2763   {
2764     \markdownRendererAmpersandPrototype

```

```

2765 }
2766 \seq_gput_right:Nn
2767 \g_@@_renderers_seq
2768 { ampersand }
2769 \prop_gput:Nnn
2770 \g_@@_renderer_arities_prop
2771 { ampersand }
2772 { 0 }
2773 \cs_gset_protected:Npn
2774 \markdownRendererUnderscore
2775 {
2776   \markdownRendererUnderscorePrototype
2777 }
2778 \seq_gput_right:Nn
2779 \g_@@_renderers_seq
2780 { underscore }
2781 \prop_gput:Nnn
2782 \g_@@_renderer_arities_prop
2783 { underscore }
2784 { 0 }
2785 \cs_gset_protected:Npn
2786 \markdownRendererHash
2787 {
2788   \markdownRendererHashPrototype
2789 }
2790 \seq_gput_right:Nn
2791 \g_@@_renderers_seq
2792 { hash }
2793 \prop_gput:Nnn
2794 \g_@@_renderer_arities_prop
2795 { hash }
2796 { 0 }
2797 \cs_gset_protected:Npn
2798 \markdownRendererCircumflex
2799 {
2800   \markdownRendererCircumflexPrototype
2801 }
2802 \seq_gput_right:Nn
2803 \g_@@_renderers_seq
2804 { circumflex }
2805 \prop_gput:Nnn
2806 \g_@@_renderer_arities_prop
2807 { circumflex }
2808 { 0 }
2809 \cs_gset_protected:Npn
2810 \markdownRendererBackslash
2811 {

```

```

2812     \markdownRendererBackslashPrototype
2813   }
2814 \seq_gput_right:Nn
2815   \g_@@_renderers_seq
2816   { backslash }
2817 \prop_gput:Nnn
2818   \g_@@_renderer_arities_prop
2819   { backslash }
2820   { 0 }
2821 \cs_gset_protected:Npn
2822   \markdownRendererTilde
2823   {
2824     \markdownRendererTildePrototype
2825   }
2826 \seq_gput_right:Nn
2827   \g_@@_renderers_seq
2828   { tilde }
2829 \prop_gput:Nnn
2830   \g_@@_renderer_arities_prop
2831   { tilde }
2832   { 0 }
2833 \cs_gset_protected:Npn
2834   \markdownRendererPipe
2835   {
2836     \markdownRendererPipePrototype
2837   }
2838 \seq_gput_right:Nn
2839   \g_@@_renderers_seq
2840   { pipe }
2841 \prop_gput:Nnn
2842   \g_@@_renderer_arities_prop
2843   { pipe }
2844   { 0 }
2845 \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.

```

2846 \ExplSyntaxOn
2847 \cs_gset_protected:Npn
2848   \markdownRendererStrikeThrough
2849   {
2850     \markdownRendererStrikeThroughPrototype
2851   }

```

```

2852 \seq_gput_right:Nn
2853   \g_@@_renderers_seq
2854   { strikeThrough }
2855 \prop_gput:Nnn
2856   \g_@@_renderer_arities_prop
2857   { strikeThrough }
2858   { 1 }
2859 \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.

```

2860 \ExplSyntaxOn
2861 \cs_gset_protected:Npn
2862   \markdownRendererSubscript
2863   {
2864     \markdownRendererSubscriptPrototype
2865   }
2866 \seq_gput_right:Nn
2867   \g_@@_renderers_seq
2868   { subscript }
2869 \prop_gput:Nnn
2870   \g_@@_renderer_arities_prop
2871   { subscript }
2872   { 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.

```

2873 \ExplSyntaxOn
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_EX Math Renderers

The `\markdownRendererInlineMath` and `\markdownRendererDisplayMath` macros represent inline and display T_EX math. Both macros receive a single argument that corresponds to the T_EX 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_EX.

```

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_EX Token Renderer Macros and Key-Values

We define the command `\@@_define_renderers:` that defines plain T_EX 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` [11]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```

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

By default, the renderer prototypes for YAML metadata provide a high-level interface that can be programmed using the `markdown/jekyllData` key-values from the `l3keys` module of the \LaTeX 3 kernel.

```

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

```

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

```

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

```

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

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

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

```

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 L^AT_EX Interface

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

To determine whether L^AT_EX is the top layer or if there are other layers above L^AT_EX, we take a look on whether the `\c_@@_top_layer_tl` token list has already been defined. If not, we will assume that L^AT_EX 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^AT_EX interface is implemented by the `markdown.sty` file, which can be loaded from the L^AT_EX document preamble as follows:

```
\usepackage[⟨options⟩]{markdown}
```

where `⟨options⟩` are the L^AT_EX 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^AT_EX 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^AT_EX 2_ε parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown`, `markdown*`, and `yaml` L^AT_EX environments, and redefines the `\markinline`, `\markdownInput`, and `\yamlInput` commands.

2.3.1.1 Typesetting Markdown and YAML directly

The `markdown` and `markdown*` L^AT_EX environments are aliases for the macros `\markdownBegin` and `\markdownEnd` exposed by the plain T_EX 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^AT_EX 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^AT_EX code showcases the usage of the `markdown` and `markdown*` environments:

```

\documentclass{article}
\usepackage{markdown}

```

```

\documentclass{article}
\usepackage{markdown}

```

<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^AT_EX 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_EX'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^AT_EX 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^AT_EX environment is an alias for the macros `\yamlBegin` and `\yamlEnd` exposed by the plain T_EX interface.

```
3852 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts \LaTeX interface options (see Section 2.3.3) as the only optional argument.

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

The following example \LaTeX code showcases the usage of the `yaml` environment:

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

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

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

You can't directly extend the `yaml` \LaTeX 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 $\text{T}_{\text{E}}\text{X}$. Unlike the `\markdownInput` macro provided by the plain $\text{T}_{\text{E}}\text{X}$ interface, this macro also accepts $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown document.

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

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

The `\yamlInput` macro accepts a single mandatory parameter containing the filename of a YAML document and expands to the result of the conversion of the input YAML document to plain $\text{T}_{\text{E}}\text{X}$. Unlike the `\yamlInput` macro provided by the plain $\text{T}_{\text{E}}\text{X}$ interface, this macro also accepts $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ interface options (see Section 2.3.3) as its optional argument. These options will only influence this YAML document.

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

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

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

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
\markdownInput[
  jekyllData,
  expectJekyllData,
  ensureJekyllData,
  smartEllipses,
]{hello.yml}
\end{document}
```

2.3.2 Using L^AT_EX hooks with the Markdown package

L^AT_EX provides an intricate hook management system that allows users to insert extra material before and after certain T_EX macros and L^AT_EX environments, among other things. [12, Section 3.1.2]

The Markdown package is compatible with hooks and allows the use of hooks to insert extra material before T_EX commands and before/after L^AT_EX environments without restriction:

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

Processing the above example with L^AT_EX will produce the text “[markdownfoo emphasis: _bar_ baz!/markdown](#)”, as expected.

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

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

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

Processing the above example with L^AT_EX will produce the text “[markdownfoo emphasis_bar_/emphasis baz!/markdown](#)”, as expected.

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

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

2.3.3 Options

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

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

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

2.3.3.1 Finalizing and Freezing the Cache

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

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

We hope that other packages will support the `finalizcache` and `frozenscache` package options in the future, so that they can become a standard interface for preparing \LaTeX document sources for distribution.

```
3853 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3854 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

2.3.3.2 Generating Plain \TeX Option, Token Renderer, and Token Renderer Prototype Macros and Key-Values

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

```

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 \LaTeX code showcases a possible configuration of plain \TeX interface options `hybrid`, `smartEllipses`, and `cacheDir`.

```

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

```

2.3.4 Themes

In Section 2.2.3, we described the concept of themes. In \LaTeX , we expand on the concept of themes by allowing a theme to be a full-blown \LaTeX package. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load a \LaTeX package named `markdowntheme<munged theme name>.sty` 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 \LaTeX -specific `.sty 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 `.sty theme file` using the `\markdownLoadPlainTeXTheme` macro.

If the \LaTeX option with keys `theme` or `import` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown \LaTeX package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named

`witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` L^AT_EX package, and finally the `markdownthemewitiko_dot.sty` L^AT_EX package:

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

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

Due to limitations of L^AT_EX, themes may not be loaded after the beginning of a L^AT_EX 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^AT_EX themes provided with the Markdown package include:

witiko/markdown/defaults A L^AT_EX theme with the default definitions of token renderer prototypes for plain T_EX. 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^AT_EX module, we load the `witiko/markdown/defaults` L^AT_EX 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     \ExplSyntaxOff
3878     \AtEndOfPackage
3879     {
3880       \@@_if_option:nF
3881         { noDefaults }
3882         {
3883           \@@_if_option:nTF
```

```

3884         { experimental }
3885     {
3886         \@@_setup:n
3887         { theme = witiko/markdown/defaults@experimental }
3888     }
3889     {
3890         \@@_setup:n
3891         { theme = witiko/markdown/defaults }
3892     }
3893 }
3894 }
3895 \ExplSyntaxOn
3896 }
3897 \ExplSyntaxOff

```

Please, see Section 3.3.2 for implementation details of the built-in L^AT_EX themes.

2.4 ConT_EXt Interface

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

```

3898 \ExplSyntaxOn
3899 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3900 \cs_generate_variant:Nn
3901   \tl_const:Nn
3902   { NV }
3903 \tl_if_exist:NF
3904   \c_@@_top_layer_tl
3905   {
3906     \tl_const:NV
3907     \c_@@_top_layer_tl
3908     \c_@@_option_layer_context_tl
3909   }
3910 \ExplSyntaxOff

```

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

```

3911 \writestatus{loading}{ConTEXt User Module / markdown}%
3912 \startmodule[markdown]
3913 \def\dospecials{\do\ \do\\\do{\\do\}\do\$\do\&%
3914   \do#\do\^\do\_do\%do\~}%
3915 \input markdown/markdown

```

The ConT_EXt interface is implemented by the `t-markdown.tex` ConT_EXt 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 \input ting the file.

2.4.1 Typesetting Markdown and YAML

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

2.4.1.1 Typesetting Markdown and YAML directly

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

```
3916 \let\startmarkdown\relax
3917 \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 Con \TeX t 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.

```
3918 \let\startyaml\relax
3919 \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 Con \TeX t 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.

```
3920 \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.

```
3921 \let\inputyaml\relax
```

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

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

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

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

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

2.4.2 Options

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

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

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

```
3922 \ExplSyntaxOn
3923 \cs_new:Npn
3924   \setupmarkdown
3925   [ #1 ]
3926   {
3927     \@@_setup:n
3928     { #1 }
3929   }
```

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

```

3930 \cs_gset_eq:NN
3931   \setupyaml
3932   \setupmarkdown

```

2.4.2.1 Generating Plain T_EX Option Macros and Key-Values

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

```

3933 \cs_new:Nn \@@_caseless:N
3934   {
3935     \regex_replace_all:nnN
3936       { ([a-z])([A-Z]) }
3937       { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3938       #1
3939     \tl_set:Nx
3940       #1
3941       { #1 }
3942   }
3943 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3944 \str_if_eq:VVT
3945   \c_@@_top_layer_tl
3946   \c_@@_option_layer_context_tl
3947   {
3948     \@@_define_option_commands_and_keyvals:
3949     \@@_define_renderers:
3950     \@@_define_renderer_prototypes:
3951   }

```

2.4.3 Themes

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

Having the Markdown package automatically load either the generic `.tex theme file` or the ConT_EXt-specific `t-*.tex` theme file allows developers to have a single *theme file*, when the theme is small or the difference between T_EX formats is unimportant, and scale up to separate theme files native to different T_EX formats for large multi-format themes, where different code is needed for different T_EX 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.

```
3952 \prop_new:N
3953   \g_@@_context_built_in_themes_prop
3954 \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.

```
3955 \startmodule[markdownthemewitiko_markdown_defaults]
3956 \unprotect
```

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

3 Implementation

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

Figure 1 shows the high-level structure of the Markdown package: The translation from markdown to TeX *token renderers* is performed by the Lua layer. The plain TeX layer provides default definitions for the token renderers. The L^AT_EX and ConTeXt layers correct idiosyncrasies of the respective TeX formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain 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 T_EX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3957 local upper, format, length =
3958   string.upper, string.format, string.len
3959 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3960   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3961   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

3.1.1 Utility Functions

This section documents the utility functions used by the plain T_EX writer and the markdown reader. These functions are encapsulated in the `util` object. The functions were originally located in the `lunamark/util.lua` file in the Lunamark Lua module.

```
3962 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.

```
3963 function util.err(msg, exit_code)
3964   io.stderr:write("markdown.lua: " .. msg .. "\n")
3965   os.exit(exit_code or 1)
3966 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.

```
3967 function util.cache(dir, string, salt, transform, suffix)
3968   local digest = md5.sumhexa(string .. (salt or ""))
3969   local name = util.pathname(dir, digest .. suffix)
3970   local file = io.open(name, "r")
3971   if file == nil then -- If no cache entry exists, create a new one.
3972     file = assert(io.open(name, "w"),
3973       [[Could not open file ]] .. name .. [[ for writing]])
3974     local result = string
3975     if transform ~= nil then
3976       result = transform(result)
3977     end
3978     assert(file:write(result))
3979     assert(file:close())
3980   end
3981   return name
3982 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.

```

3983 function util.cache_verbatim(dir, string)
3984   local name = util.cache(dir, string, nil, nil, ".verbatim")
3985   return name
3986 end

```

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

```

3987 function util.table_copy(t)
3988   local u = { }
3989   for k, v in pairs(t) do u[k] = v end
3990   return setmetatable(u, getmetatable(t))
3991 end

```

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

```

3992 function util.encode_json_string(s)
3993   s = s:gsub([[\\]], [[\\]])
3994   s = s:gsub([[\"]], [[\"]])
3995   return [[\"]] .. s .. [[\"]]
3996 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 Ierusalimsky [14, Chapter 21].

```

3997 function util.expand_tabs_in_line(s, tabstop)
3998   local tab = tabstop or 4
3999   local corr = 0
4000   return (s:gsub(")\t", function(p)
4001     local sp = tab - (p - 1 + corr) % tab
4002     corr = corr - 1 + sp
4003     return string.rep(" ", sp)
4004   end))
4005 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.

```

4006 function util.walk(t, f)
4007   local typ = type(t)
4008   if typ == "string" then
4009     f(t)
4010   elseif typ == "table" then
4011     local i = 1
4012     local n
4013     n = t[i]
4014     while n do
4015       util.walk(n, f)
4016       i = i + 1

```

```

4017     n = t[i]
4018     end
4019 elseif typ == "function" then
4020     local ok, val = pcall(t)
4021     if ok then
4022         util.walk(val,f)
4023     end
4024 else
4025     f(tostring(t))
4026 end
4027 end

```

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

```

4028 function util.flatten(ary)
4029     local new = {}
4030     for _,v in ipairs(ary) do
4031         if type(v) == "table" then
4032             for _,w in ipairs(util.flatten(v)) do
4033                 new[#new + 1] = w
4034             end
4035         else
4036             new[#new + 1] = v
4037         end
4038     end
4039     return new
4040 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.

```

4041 function util.rope_to_string(rope)
4042     local buffer = {}
4043     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
4044     return table.concat(buffer)
4045 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.

```

4046 function util.rope_last(rope)
4047     if #rope == 0 then
4048         return nil
4049     else
4050         local l = rope[#rope]
4051         if type(l) == "table" then
4052             return util.rope_last(l)
4053         else
4054             return l
4055         end
4056     end

```

```

4056 end
4057 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 $1 \leq i \leq \#ary$.

```

4058 function util.intersperse(ary, x)
4059   local new = {}
4060   local l = #ary
4061   for i,v in ipairs(ary) do
4062     local n = #new
4063     new[n + 1] = v
4064     if i ~= l then
4065       new[n + 2] = x
4066     end
4067   end
4068   return new
4069 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 $1 \leq i \leq \#ary$.

```

4070 function util.map(ary, f)
4071   local new = {}
4072   for i,v in ipairs(ary) do
4073     new[i] = f(v)
4074   end
4075   return new
4076 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.

```

4077 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

4078   local char_escapes_list = ""
4079   for i,_ in pairs(char_escapes) do
4080     char_escapes_list = char_escapes_list .. i
4081   end

```

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

```

4082   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.

```
4083  if string_escapes then
4084      for k,v in pairs(string_escapes) do
4085          escapable = P(k) / v + escapable
4086      end
4087  end
```

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

```
4088  local escape_string = Cs((escapable + any)^0)
```

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

```
4089  return function(s)
4090      return lpeg.match(escape_string, s)
4091  end
4092 end
```

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

```
4093 function util.pathname(dir, file)
4094  if #dir == 0 then
4095      return file
4096  else
4097      return dir .. "/" .. file
4098  end
4099 end
```

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

```
4100 function util.salt(options)
4101  local opt_string = {}
4102  for k, _ in pairs(defaultOptions) do
4103      local v = options[k]
4104      if type(v) == "table" then
4105          for _, i in ipairs(v) do
4106              opt_string[#opt_string+1] = k .. "=" .. tostring(i)
4107          end
4108      end
4109  end
```

The `cacheDir` option is disregarded.

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



```

4114         .. "," .. metadata.version
4115     return salt
4116 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.

```

4117 function util.warning(s)
4118     io.stderr:write("Warning: " .. s .. "\n")
4119 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.

```

4120 local entities = {}
4121
4122 local character_entities = {
4123     ["Tab"] = 9,
4124     ["NewLine"] = 10,
4125     ["excl"] = 33,
4126     ["QUOT"] = 34,
4127     ["quot"] = 34,
4128     ["num"] = 35,
4129     ["dollar"] = 36,
4130     ["percent"] = 37,
4131     ["AMP"] = 38,
4132     ["amp"] = 38,
4133     ["apos"] = 39,
4134     ["lpar"] = 40,
4135     ["rpar"] = 41,
4136     ["ast"] = 42,
4137     ["midast"] = 42,
4138     ["plus"] = 43,
4139     ["comma"] = 44,
4140     ["period"] = 46,
4141     ["sol"] = 47,
4142     ["colon"] = 58,
4143     ["semi"] = 59,
4144     ["LT"] = 60,
4145     ["lt"] = 60,
4146     ["nvlT"] = {60, 8402},
4147     ["bne"] = {61, 8421},
4148     ["equals"] = 61,
4149     ["GT"] = 62,
4150     ["gt"] = 62,

```

4151 ["nvgt"] = {62, 8402},
4152 ["quest"] = 63,
4153 ["commat"] = 64,
4154 ["lbrack"] = 91,
4155 ["lsqb"] = 91,
4156 ["bsol"] = 92,
4157 ["rbrack"] = 93,
4158 ["rsqb"] = 93,
4159 ["Hat"] = 94,
4160 ["UnderBar"] = 95,
4161 ["lowbar"] = 95,
4162 ["DiacriticalGrave"] = 96,
4163 ["grave"] = 96,
4164 ["fjlig"] = {102, 106},
4165 ["lbrace"] = 123,
4166 ["lcub"] = 123,
4167 ["VerticalLine"] = 124,
4168 ["verbar"] = 124,
4169 ["vert"] = 124,
4170 ["rbrace"] = 125,
4171 ["rcub"] = 125,
4172 ["NonBreakingSpace"] = 160,
4173 ["nbsp"] = 160,
4174 ["iexcl"] = 161,
4175 ["cent"] = 162,
4176 ["pound"] = 163,
4177 ["curren"] = 164,
4178 ["yen"] = 165,
4179 ["brvbar"] = 166,
4180 ["sect"] = 167,
4181 ["Dot"] = 168,
4182 ["DoubleDot"] = 168,
4183 ["die"] = 168,
4184 ["uml"] = 168,
4185 ["COPY"] = 169,
4186 ["copy"] = 169,
4187 ["ordf"] = 170,
4188 ["laquo"] = 171,
4189 ["not"] = 172,
4190 ["shy"] = 173,
4191 ["REG"] = 174,
4192 ["circledR"] = 174,
4193 ["reg"] = 174,
4194 ["macr"] = 175,
4195 ["strns"] = 175,
4196 ["deg"] = 176,
4197 ["PlusMinus"] = 177,

4198 ["plusmn"] = 177,
4199 ["pm"] = 177,
4200 ["sup2"] = 178,
4201 ["sup3"] = 179,
4202 ["DiacriticalAcute"] = 180,
4203 ["acute"] = 180,
4204 ["micro"] = 181,
4205 ["para"] = 182,
4206 ["CenterDot"] = 183,
4207 ["centerdot"] = 183,
4208 ["middot"] = 183,
4209 ["Cedilla"] = 184,
4210 ["cedil"] = 184,
4211 ["sup1"] = 185,
4212 ["ordm"] = 186,
4213 ["raquo"] = 187,
4214 ["frac14"] = 188,
4215 ["frac12"] = 189,
4216 ["half"] = 189,
4217 ["frac34"] = 190,
4218 ["iquest"] = 191,
4219 ["Agrave"] = 192,
4220 ["Aacute"] = 193,
4221 ["Acirc"] = 194,
4222 ["Atilde"] = 195,
4223 ["Auml"] = 196,
4224 ["Aring"] = 197,
4225 ["angst"] = 197,
4226 ["AElig"] = 198,
4227 ["Ccedil"] = 199,
4228 ["Egrave"] = 200,
4229 ["Eacute"] = 201,
4230 ["Ecirc"] = 202,
4231 ["Euml"] = 203,
4232 ["Igrave"] = 204,
4233 ["Iacute"] = 205,
4234 ["Icirc"] = 206,
4235 ["Iuml"] = 207,
4236 ["ETH"] = 208,
4237 ["Ntilde"] = 209,
4238 ["Ograve"] = 210,
4239 ["Oacute"] = 211,
4240 ["Ocirc"] = 212,
4241 ["Otilde"] = 213,
4242 ["Ouml"] = 214,
4243 ["times"] = 215,
4244 ["Oslash"] = 216,

4245 ["Ugrave"] = 217,
4246 ["Uacute"] = 218,
4247 ["Ucirc"] = 219,
4248 ["Uuml"] = 220,
4249 ["Yacute"] = 221,
4250 ["THORN"] = 222,
4251 ["szlig"] = 223,
4252 ["agrave"] = 224,
4253 ["aacute"] = 225,
4254 ["acirc"] = 226,
4255 ["atilde"] = 227,
4256 ["auml"] = 228,
4257 ["aring"] = 229,
4258 ["aelig"] = 230,
4259 ["ccedil"] = 231,
4260 ["egrave"] = 232,
4261 ["eacute"] = 233,
4262 ["ecirc"] = 234,
4263 ["euml"] = 235,
4264 ["igrave"] = 236,
4265 ["iacute"] = 237,
4266 ["icirc"] = 238,
4267 ["iuml"] = 239,
4268 ["eth"] = 240,
4269 ["ntilde"] = 241,
4270 ["ograve"] = 242,
4271 ["oacute"] = 243,
4272 ["ocirc"] = 244,
4273 ["otilde"] = 245,
4274 ["ouml"] = 246,
4275 ["div"] = 247,
4276 ["divide"] = 247,
4277 ["oslash"] = 248,
4278 ["ugrave"] = 249,
4279 ["uacute"] = 250,
4280 ["ucirc"] = 251,
4281 ["uuml"] = 252,
4282 ["yacute"] = 253,
4283 ["thorn"] = 254,
4284 ["yuml"] = 255,
4285 ["Amacr"] = 256,
4286 ["amacr"] = 257,
4287 ["Abreve"] = 258,
4288 ["abreve"] = 259,
4289 ["Aogon"] = 260,
4290 ["aogon"] = 261,
4291 ["Cacute"] = 262,

4292 ["cacute"] = 263,
4293 ["Ccirc"] = 264,
4294 ["ccirc"] = 265,
4295 ["Cdot"] = 266,
4296 ["cdot"] = 267,
4297 ["Ccaron"] = 268,
4298 ["ccaron"] = 269,
4299 ["Dcaron"] = 270,
4300 ["dcaron"] = 271,
4301 ["Dstrok"] = 272,
4302 ["dstrok"] = 273,
4303 ["Emacr"] = 274,
4304 ["emacr"] = 275,
4305 ["Edot"] = 278,
4306 ["edot"] = 279,
4307 ["Eogon"] = 280,
4308 ["eogon"] = 281,
4309 ["Ecaron"] = 282,
4310 ["ecaron"] = 283,
4311 ["Gcirc"] = 284,
4312 ["gcirc"] = 285,
4313 ["Gbreve"] = 286,
4314 ["gbreve"] = 287,
4315 ["Gdot"] = 288,
4316 ["gdot"] = 289,
4317 ["Gcedil"] = 290,
4318 ["Hcirc"] = 292,
4319 ["hcirc"] = 293,
4320 ["Hstrok"] = 294,
4321 ["hstrok"] = 295,
4322 ["Itilde"] = 296,
4323 ["itilde"] = 297,
4324 ["Imacr"] = 298,
4325 ["imacr"] = 299,
4326 ["Iogon"] = 302,
4327 ["iogon"] = 303,
4328 ["Idot"] = 304,
4329 ["imath"] = 305,
4330 ["inodot"] = 305,
4331 ["IJlig"] = 306,
4332 ["ijlig"] = 307,
4333 ["Jcirc"] = 308,
4334 ["jcirc"] = 309,
4335 ["Kcedil"] = 310,
4336 ["kcedil"] = 311,
4337 ["kgreen"] = 312,
4338 ["Lacute"] = 313,

4339 ["lacute"] = 314,
4340 ["Lcedil"] = 315,
4341 ["lcedil"] = 316,
4342 ["Lcaron"] = 317,
4343 ["lcaron"] = 318,
4344 ["Lmidot"] = 319,
4345 ["lmidot"] = 320,
4346 ["Lstrok"] = 321,
4347 ["lstrok"] = 322,
4348 ["Nacute"] = 323,
4349 ["nacute"] = 324,
4350 ["Ncedil"] = 325,
4351 ["ncedil"] = 326,
4352 ["Ncaron"] = 327,
4353 ["ncaron"] = 328,
4354 ["napos"] = 329,
4355 ["ENG"] = 330,
4356 ["eng"] = 331,
4357 ["Omacr"] = 332,
4358 ["omacr"] = 333,
4359 ["Odblac"] = 336,
4360 ["odblac"] = 337,
4361 ["OElig"] = 338,
4362 ["oelig"] = 339,
4363 ["Racute"] = 340,
4364 ["racute"] = 341,
4365 ["Rcedil"] = 342,
4366 ["rcedil"] = 343,
4367 ["Rcaron"] = 344,
4368 ["rcaron"] = 345,
4369 ["Sacute"] = 346,
4370 ["sacute"] = 347,
4371 ["Scirc"] = 348,
4372 ["scirc"] = 349,
4373 ["Scedil"] = 350,
4374 ["scedil"] = 351,
4375 ["Scaron"] = 352,
4376 ["scaron"] = 353,
4377 ["Tcedil"] = 354,
4378 ["tcedil"] = 355,
4379 ["Tcaron"] = 356,
4380 ["tcaron"] = 357,
4381 ["Tstrok"] = 358,
4382 ["tstrok"] = 359,
4383 ["Utilde"] = 360,
4384 ["utilde"] = 361,
4385 ["Umacr"] = 362,

4386 ["umacr"] = 363,
4387 ["Ubreve"] = 364,
4388 ["ubreve"] = 365,
4389 ["Uring"] = 366,
4390 ["uring"] = 367,
4391 ["Udblac"] = 368,
4392 ["udblac"] = 369,
4393 ["Uogon"] = 370,
4394 ["uogon"] = 371,
4395 ["Wcirc"] = 372,
4396 ["wcirc"] = 373,
4397 ["Ycirc"] = 374,
4398 ["ycirc"] = 375,
4399 ["Yuml"] = 376,
4400 ["Zacute"] = 377,
4401 ["zacute"] = 378,
4402 ["Zdot"] = 379,
4403 ["zdot"] = 380,
4404 ["Zcaron"] = 381,
4405 ["zcaron"] = 382,
4406 ["fnof"] = 402,
4407 ["imped"] = 437,
4408 ["gacute"] = 501,
4409 ["jmath"] = 567,
4410 ["circ"] = 710,
4411 ["Hacek"] = 711,
4412 ["caron"] = 711,
4413 ["Breve"] = 728,
4414 ["breve"] = 728,
4415 ["DiacriticalDot"] = 729,
4416 ["dot"] = 729,
4417 ["ring"] = 730,
4418 ["ogon"] = 731,
4419 ["DiacriticalTilde"] = 732,
4420 ["tilde"] = 732,
4421 ["DiacriticalDoubleAcute"] = 733,
4422 ["dblac"] = 733,
4423 ["DownBreve"] = 785,
4424 ["Alpha"] = 913,
4425 ["Beta"] = 914,
4426 ["Gamma"] = 915,
4427 ["Delta"] = 916,
4428 ["Epsilon"] = 917,
4429 ["Zeta"] = 918,
4430 ["Eta"] = 919,
4431 ["Theta"] = 920,
4432 ["Iota"] = 921,

4433 ["Kappa"] = 922,
4434 ["Lambda"] = 923,
4435 ["Mu"] = 924,
4436 ["Nu"] = 925,
4437 ["Xi"] = 926,
4438 ["Omicron"] = 927,
4439 ["Pi"] = 928,
4440 ["Rho"] = 929,
4441 ["Sigma"] = 931,
4442 ["Tau"] = 932,
4443 ["Upsilon"] = 933,
4444 ["Phi"] = 934,
4445 ["Chi"] = 935,
4446 ["Psi"] = 936,
4447 ["Omega"] = 937,
4448 ["ohm"] = 937,
4449 ["alpha"] = 945,
4450 ["beta"] = 946,
4451 ["gamma"] = 947,
4452 ["delta"] = 948,
4453 ["epsi"] = 949,
4454 ["epsilon"] = 949,
4455 ["zeta"] = 950,
4456 ["eta"] = 951,
4457 ["theta"] = 952,
4458 ["iota"] = 953,
4459 ["kappa"] = 954,
4460 ["lambda"] = 955,
4461 ["mu"] = 956,
4462 ["nu"] = 957,
4463 ["xi"] = 958,
4464 ["omicron"] = 959,
4465 ["pi"] = 960,
4466 ["rho"] = 961,
4467 ["sigmaf"] = 962,
4468 ["sigmav"] = 962,
4469 ["varsigma"] = 962,
4470 ["sigma"] = 963,
4471 ["tau"] = 964,
4472 ["upsi"] = 965,
4473 ["upsilon"] = 965,
4474 ["phi"] = 966,
4475 ["chi"] = 967,
4476 ["psi"] = 968,
4477 ["omega"] = 969,
4478 ["thetasym"] = 977,
4479 ["thetav"] = 977,

4480 ["vartheta"] = 977,
4481 ["Upsi"] = 978,
4482 ["upsih"] = 978,
4483 ["phiv"] = 981,
4484 ["straightphi"] = 981,
4485 ["varphi"] = 981,
4486 ["piv"] = 982,
4487 ["varpi"] = 982,
4488 ["Gammad"] = 988,
4489 ["digamma"] = 989,
4490 ["gammad"] = 989,
4491 ["kappav"] = 1008,
4492 ["varkappa"] = 1008,
4493 ["rhov"] = 1009,
4494 ["varrho"] = 1009,
4495 ["epsiv"] = 1013,
4496 ["straightepsilon"] = 1013,
4497 ["varepsilon"] = 1013,
4498 ["backepsilon"] = 1014,
4499 ["bepsi"] = 1014,
4500 ["IOcy"] = 1025,
4501 ["DJcy"] = 1026,
4502 ["GJcy"] = 1027,
4503 ["Jukcy"] = 1028,
4504 ["DScy"] = 1029,
4505 ["Iukcy"] = 1030,
4506 ["YIcy"] = 1031,
4507 ["Jsercy"] = 1032,
4508 ["LJcy"] = 1033,
4509 ["NJcy"] = 1034,
4510 ["TSHcy"] = 1035,
4511 ["KJcy"] = 1036,
4512 ["Ubrcy"] = 1038,
4513 ["DZcy"] = 1039,
4514 ["Acy"] = 1040,
4515 ["Bcy"] = 1041,
4516 ["Vcy"] = 1042,
4517 ["Gcy"] = 1043,
4518 ["Dcy"] = 1044,
4519 ["IEcy"] = 1045,
4520 ["ZHcy"] = 1046,
4521 ["Zcy"] = 1047,
4522 ["Icy"] = 1048,
4523 ["Jcy"] = 1049,
4524 ["Kcy"] = 1050,
4525 ["Lcy"] = 1051,
4526 ["Mcy"] = 1052,

4527 ["Ncy"] = 1053,
4528 ["Ocy"] = 1054,
4529 ["Pcy"] = 1055,
4530 ["Rcy"] = 1056,
4531 ["Scy"] = 1057,
4532 ["Tcy"] = 1058,
4533 ["Ucy"] = 1059,
4534 ["Fcy"] = 1060,
4535 ["KHcy"] = 1061,
4536 ["TScy"] = 1062,
4537 ["CHcy"] = 1063,
4538 ["SHcy"] = 1064,
4539 ["SHCHcy"] = 1065,
4540 ["HARDcy"] = 1066,
4541 ["Ycy"] = 1067,
4542 ["SOFTcy"] = 1068,
4543 ["Ecy"] = 1069,
4544 ["YUcy"] = 1070,
4545 ["YAcy"] = 1071,
4546 ["acy"] = 1072,
4547 ["bcy"] = 1073,
4548 ["vcy"] = 1074,
4549 ["gcy"] = 1075,
4550 ["dcy"] = 1076,
4551 ["iecy"] = 1077,
4552 ["zhcy"] = 1078,
4553 ["zcy"] = 1079,
4554 ["icy"] = 1080,
4555 ["jcy"] = 1081,
4556 ["kcy"] = 1082,
4557 ["lcy"] = 1083,
4558 ["mcy"] = 1084,
4559 ["ncy"] = 1085,
4560 ["ocy"] = 1086,
4561 ["pcy"] = 1087,
4562 ["rcy"] = 1088,
4563 ["scy"] = 1089,
4564 ["tcy"] = 1090,
4565 ["ucy"] = 1091,
4566 ["fcy"] = 1092,
4567 ["khcy"] = 1093,
4568 ["tscy"] = 1094,
4569 ["chcy"] = 1095,
4570 ["shcy"] = 1096,
4571 ["shchcy"] = 1097,
4572 ["hardcy"] = 1098,
4573 ["ycy"] = 1099,

4574 ["softcy"] = 1100,
4575 ["ecy"] = 1101,
4576 ["yucy"] = 1102,
4577 ["yacy"] = 1103,
4578 ["iocy"] = 1105,
4579 ["djcy"] = 1106,
4580 ["gjcy"] = 1107,
4581 ["jukcy"] = 1108,
4582 ["dscy"] = 1109,
4583 ["iukcy"] = 1110,
4584 ["yicy"] = 1111,
4585 ["jsercy"] = 1112,
4586 ["ljcy"] = 1113,
4587 ["njcy"] = 1114,
4588 ["tshcy"] = 1115,
4589 ["kjcy"] = 1116,
4590 ["ubrcty"] = 1118,
4591 ["dzcy"] = 1119,
4592 ["ensp"] = 8194,
4593 ["emsp"] = 8195,
4594 ["emsp13"] = 8196,
4595 ["emsp14"] = 8197,
4596 ["numsp"] = 8199,
4597 ["puncsp"] = 8200,
4598 ["ThinSpace"] = 8201,
4599 ["thinsp"] = 8201,
4600 ["VeryThinSpace"] = 8202,
4601 ["hairsp"] = 8202,
4602 ["NegativeMediumSpace"] = 8203,
4603 ["NegativeThickSpace"] = 8203,
4604 ["NegativeThinSpace"] = 8203,
4605 ["NegativeVeryThinSpace"] = 8203,
4606 ["ZeroWidthSpace"] = 8203,
4607 ["zwnj"] = 8204,
4608 ["zwj"] = 8205,
4609 ["lrm"] = 8206,
4610 ["rlm"] = 8207,
4611 ["dash"] = 8208,
4612 ["hyphen"] = 8208,
4613 ["ndash"] = 8211,
4614 ["mdash"] = 8212,
4615 ["horbar"] = 8213,
4616 ["Verbar"] = 8214,
4617 ["Vert"] = 8214,
4618 ["OpenCurlyQuote"] = 8216,
4619 ["lsquo"] = 8216,
4620 ["CloseCurlyQuote"] = 8217,

4621 ["rsquo"] = 8217,
 4622 ["rsquor"] = 8217,
 4623 ["lsquor"] = 8218,
 4624 ["sbquo"] = 8218,
 4625 ["OpenCurlyDoubleQuote"] = 8220,
 4626 ["ldquo"] = 8220,
 4627 ["CloseCurlyDoubleQuote"] = 8221,
 4628 ["rdquo"] = 8221,
 4629 ["rdquor"] = 8221,
 4630 ["bdquo"] = 8222,
 4631 ["ldquor"] = 8222,
 4632 ["dagger"] = 8224,
 4633 ["Dagger"] = 8225,
 4634 ["ddagger"] = 8225,
 4635 ["bull"] = 8226,
 4636 ["bullet"] = 8226,
 4637 ["nldr"] = 8229,
 4638 ["hellip"] = 8230,
 4639 ["mldr"] = 8230,
 4640 ["permil"] = 8240,
 4641 ["pertenk"] = 8241,
 4642 ["prime"] = 8242,
 4643 ["Prime"] = 8243,
 4644 ["tprime"] = 8244,
 4645 ["backprime"] = 8245,
 4646 ["bprime"] = 8245,
 4647 ["lsaquo"] = 8249,
 4648 ["rsaquo"] = 8250,
 4649 ["OverBar"] = 8254,
 4650 ["oline"] = 8254,
 4651 ["caret"] = 8257,
 4652 ["hybull"] = 8259,
 4653 ["frasl"] = 8260,
 4654 ["bsemi"] = 8271,
 4655 ["qprime"] = 8279,
 4656 ["MediumSpace"] = 8287,
 4657 ["ThickSpace"] = {8287, 8202},
 4658 ["NoBreak"] = 8288,
 4659 ["ApplyFunction"] = 8289,
 4660 ["af"] = 8289,
 4661 ["InvisibleTimes"] = 8290,
 4662 ["it"] = 8290,
 4663 ["InvisibleComma"] = 8291,
 4664 ["ic"] = 8291,
 4665 ["euro"] = 8364,
 4666 ["TripleDot"] = 8411,
 4667 ["tdot"] = 8411,

4668 ["DotDot"] = 8412,
4669 ["Copf"] = 8450,
4670 ["complexes"] = 8450,
4671 ["incare"] = 8453,
4672 ["gscr"] = 8458,
4673 ["HilbertSpace"] = 8459,
4674 ["Hscr"] = 8459,
4675 ["hamilt"] = 8459,
4676 ["Hfr"] = 8460,
4677 ["Poincareplane"] = 8460,
4678 ["Hopf"] = 8461,
4679 ["quaternions"] = 8461,
4680 ["planckh"] = 8462,
4681 ["hbar"] = 8463,
4682 ["hslash"] = 8463,
4683 ["planck"] = 8463,
4684 ["plankv"] = 8463,
4685 ["Iscr"] = 8464,
4686 ["imagline"] = 8464,
4687 ["Ifr"] = 8465,
4688 ["Im"] = 8465,
4689 ["image"] = 8465,
4690 ["imagpart"] = 8465,
4691 ["Laplacetrnf"] = 8466,
4692 ["Lscr"] = 8466,
4693 ["lagran"] = 8466,
4694 ["ell"] = 8467,
4695 ["Nopf"] = 8469,
4696 ["naturals"] = 8469,
4697 ["numero"] = 8470,
4698 ["copysr"] = 8471,
4699 ["weierp"] = 8472,
4700 ["wp"] = 8472,
4701 ["Popf"] = 8473,
4702 ["primes"] = 8473,
4703 ["Qopf"] = 8474,
4704 ["rationals"] = 8474,
4705 ["Rscr"] = 8475,
4706 ["realine"] = 8475,
4707 ["Re"] = 8476,
4708 ["Rfr"] = 8476,
4709 ["real"] = 8476,
4710 ["realpart"] = 8476,
4711 ["Ropf"] = 8477,
4712 ["reals"] = 8477,
4713 ["rx"] = 8478,
4714 ["TRADE"] = 8482,

4715 ["trade"] = 8482,
4716 ["Zopf"] = 8484,
4717 ["integers"] = 8484,
4718 ["mho"] = 8487,
4719 ["Zfr"] = 8488,
4720 ["zeetrf"] = 8488,
4721 ["iiota"] = 8489,
4722 ["Bernoullis"] = 8492,
4723 ["Bscr"] = 8492,
4724 ["bernou"] = 8492,
4725 ["Cayleys"] = 8493,
4726 ["Cfr"] = 8493,
4727 ["escr"] = 8495,
4728 ["Escr"] = 8496,
4729 ["expectation"] = 8496,
4730 ["Fouriertrf"] = 8497,
4731 ["Fscr"] = 8497,
4732 ["Mellintrf"] = 8499,
4733 ["Mscr"] = 8499,
4734 ["phmmat"] = 8499,
4735 ["order"] = 8500,
4736 ["orderof"] = 8500,
4737 ["oscr"] = 8500,
4738 ["alefsym"] = 8501,
4739 ["aleph"] = 8501,
4740 ["beth"] = 8502,
4741 ["gimel"] = 8503,
4742 ["daleth"] = 8504,
4743 ["CapitalDifferentialD"] = 8517,
4744 ["DD"] = 8517,
4745 ["DifferentialD"] = 8518,
4746 ["dd"] = 8518,
4747 ["ExponentialE"] = 8519,
4748 ["ee"] = 8519,
4749 ["exponentiale"] = 8519,
4750 ["ImaginaryI"] = 8520,
4751 ["ii"] = 8520,
4752 ["frac13"] = 8531,
4753 ["frac23"] = 8532,
4754 ["frac15"] = 8533,
4755 ["frac25"] = 8534,
4756 ["frac35"] = 8535,
4757 ["frac45"] = 8536,
4758 ["frac16"] = 8537,
4759 ["frac56"] = 8538,
4760 ["frac18"] = 8539,
4761 ["frac38"] = 8540,

4762 ["frac58"] = 8541,
 4763 ["frac78"] = 8542,
 4764 ["LeftArrow"] = 8592,
 4765 ["ShortLeftArrow"] = 8592,
 4766 ["larr"] = 8592,
 4767 ["leftarrow"] = 8592,
 4768 ["slarr"] = 8592,
 4769 ["ShortUpArrow"] = 8593,
 4770 ["UpArrow"] = 8593,
 4771 ["uarr"] = 8593,
 4772 ["uparrow"] = 8593,
 4773 ["RightArrow"] = 8594,
 4774 ["ShortRightArrow"] = 8594,
 4775 ["rarr"] = 8594,
 4776 ["rightarrow"] = 8594,
 4777 ["srarr"] = 8594,
 4778 ["DownArrow"] = 8595,
 4779 ["ShortDownArrow"] = 8595,
 4780 ["darr"] = 8595,
 4781 ["downarrow"] = 8595,
 4782 ["LeftRightArrow"] = 8596,
 4783 ["harr"] = 8596,
 4784 ["leftrightarrow"] = 8596,
 4785 ["UpDownArrow"] = 8597,
 4786 ["updownarrow"] = 8597,
 4787 ["varr"] = 8597,
 4788 ["UpperLeftArrow"] = 8598,
 4789 ["nwarr"] = 8598,
 4790 ["nwarrow"] = 8598,
 4791 ["UpperRightArrow"] = 8599,
 4792 ["nearr"] = 8599,
 4793 ["nearrow"] = 8599,
 4794 ["LowerRightArrow"] = 8600,
 4795 ["searr"] = 8600,
 4796 ["searrow"] = 8600,
 4797 ["LowerLeftArrow"] = 8601,
 4798 ["swarr"] = 8601,
 4799 ["swarrow"] = 8601,
 4800 ["nlarr"] = 8602,
 4801 ["nleftarrow"] = 8602,
 4802 ["nrarr"] = 8603,
 4803 ["nrightarrow"] = 8603,
 4804 ["nrarrw"] = {8605, 824},
 4805 ["rarrw"] = 8605,
 4806 ["rightsquigarrow"] = 8605,
 4807 ["Larr"] = 8606,
 4808 ["twoheadleftarrow"] = 8606,

```

4809 ["Uarr"] = 8607,
4810 ["Rarr"] = 8608,
4811 ["twoheadrightarrow"] = 8608,
4812 ["Darr"] = 8609,
4813 ["larrtl"] = 8610,
4814 ["leftarrowtail"] = 8610,
4815 ["rarrtl"] = 8611,
4816 ["rightarrowtail"] = 8611,
4817 ["LeftTeeArrow"] = 8612,
4818 ["mapstoleft"] = 8612,
4819 ["UpTeeArrow"] = 8613,
4820 ["mapstoup"] = 8613,
4821 ["RightTeeArrow"] = 8614,
4822 ["map"] = 8614,
4823 ["mapsto"] = 8614,
4824 ["DownTeeArrow"] = 8615,
4825 ["mapstodown"] = 8615,
4826 ["hookleftarrow"] = 8617,
4827 ["larrhk"] = 8617,
4828 ["hookrightarrow"] = 8618,
4829 ["rarrhk"] = 8618,
4830 ["larrlp"] = 8619,
4831 ["looparrowleft"] = 8619,
4832 ["looparrowright"] = 8620,
4833 ["rarrlp"] = 8620,
4834 ["harrw"] = 8621,
4835 ["leftrightsquigarrow"] = 8621,
4836 ["nharr"] = 8622,
4837 ["nletrightarrow"] = 8622,
4838 ["Lsh"] = 8624,
4839 ["lsh"] = 8624,
4840 ["Rsh"] = 8625,
4841 ["rsh"] = 8625,
4842 ["ldsh"] = 8626,
4843 ["rdsh"] = 8627,
4844 ["crarr"] = 8629,
4845 ["cularr"] = 8630,
4846 ["curvearrowleft"] = 8630,
4847 ["curarr"] = 8631,
4848 ["curvearrowright"] = 8631,
4849 ["circlearrowleft"] = 8634,
4850 ["olarr"] = 8634,
4851 ["circlearrowright"] = 8635,
4852 ["orarr"] = 8635,
4853 ["LeftVector"] = 8636,
4854 ["lefttharpoonup"] = 8636,
4855 ["lharu"] = 8636,

```


4856 ["DownLeftVector"] = 8637,
 4857 ["leftharpoondown"] = 8637,
 4858 ["lhard"] = 8637,
 4859 ["RightUpVector"] = 8638,
 4860 ["uharr"] = 8638,
 4861 ["upharpoonright"] = 8638,
 4862 ["LeftUpVector"] = 8639,
 4863 ["uharl"] = 8639,
 4864 ["upharpoonleft"] = 8639,
 4865 ["RightVector"] = 8640,
 4866 ["rharu"] = 8640,
 4867 ["rightharpoonup"] = 8640,
 4868 ["DownRightVector"] = 8641,
 4869 ["rhard"] = 8641,
 4870 ["rightharpoondown"] = 8641,
 4871 ["RightDownVector"] = 8642,
 4872 ["dharr"] = 8642,
 4873 ["downharpoonright"] = 8642,
 4874 ["LeftDownVector"] = 8643,
 4875 ["dharl"] = 8643,
 4876 ["downharpoonleft"] = 8643,
 4877 ["RightArrowLeftArrow"] = 8644,
 4878 ["rightleftarrows"] = 8644,
 4879 ["rlarr"] = 8644,
 4880 ["UpArrowDownArrow"] = 8645,
 4881 ["udarr"] = 8645,
 4882 ["LeftArrowRightArrow"] = 8646,
 4883 ["leftrightarrows"] = 8646,
 4884 ["lrarr"] = 8646,
 4885 ["leftleftarrows"] = 8647,
 4886 ["llarr"] = 8647,
 4887 ["upuparrows"] = 8648,
 4888 ["uuarr"] = 8648,
 4889 ["rightrightarrows"] = 8649,
 4890 ["rrarr"] = 8649,
 4891 ["ddarr"] = 8650,
 4892 ["downdownarrows"] = 8650,
 4893 ["ReverseEquilibrium"] = 8651,
 4894 ["leftrightharpoons"] = 8651,
 4895 ["lrhar"] = 8651,
 4896 ["Equilibrium"] = 8652,
 4897 ["rightleftharpoons"] = 8652,
 4898 ["rlhar"] = 8652,
 4899 ["nLeftarrow"] = 8653,
 4900 ["nlArr"] = 8653,
 4901 ["nLeftrightarrow"] = 8654,
 4902 ["nhArr"] = 8654,

```

4903 ["nRightarrow"] = 8655,
4904 ["nrArr"] = 8655,
4905 ["DoubleLeftArrow"] = 8656,
4906 ["Leftarrow"] = 8656,
4907 ["lArr"] = 8656,
4908 ["DoubleUpArrow"] = 8657,
4909 ["Uparrow"] = 8657,
4910 ["uArr"] = 8657,
4911 ["DoubleRightArrow"] = 8658,
4912 ["Implies"] = 8658,
4913 ["Rightarrow"] = 8658,
4914 ["rArr"] = 8658,
4915 ["DoubleDownArrow"] = 8659,
4916 ["Downarrow"] = 8659,
4917 ["dArr"] = 8659,
4918 ["DoubleLeftRightArrow"] = 8660,
4919 ["Leftrightarrow"] = 8660,
4920 ["hArr"] = 8660,
4921 ["iff"] = 8660,
4922 ["DoubleUpDownArrow"] = 8661,
4923 ["Updownarrow"] = 8661,
4924 ["vArr"] = 8661,
4925 ["nwArr"] = 8662,
4926 ["neArr"] = 8663,
4927 ["seArr"] = 8664,
4928 ["swArr"] = 8665,
4929 ["Lleftarrow"] = 8666,
4930 ["lAarr"] = 8666,
4931 ["Rrightarrow"] = 8667,
4932 ["rAarr"] = 8667,
4933 ["zigrarr"] = 8669,
4934 ["LeftArrowBar"] = 8676,
4935 ["larrb"] = 8676,
4936 ["RightArrowBar"] = 8677,
4937 ["rarrb"] = 8677,
4938 ["DownArrowUpArrow"] = 8693,
4939 ["duarr"] = 8693,
4940 ["loarr"] = 8701,
4941 ["roarr"] = 8702,
4942 ["hoarr"] = 8703,
4943 ["ForAll"] = 8704,
4944 ["forall"] = 8704,
4945 ["comp"] = 8705,
4946 ["complement"] = 8705,
4947 ["PartialD"] = 8706,
4948 ["npart"] = {8706, 824},
4949 ["part"] = 8706,

```

4950 ["Exists"] = 8707,
 4951 ["exist"] = 8707,
 4952 ["NotExists"] = 8708,
 4953 ["nexist"] = 8708,
 4954 ["nexists"] = 8708,
 4955 ["empty"] = 8709,
 4956 ["emptyset"] = 8709,
 4957 ["emptyv"] = 8709,
 4958 ["varnothing"] = 8709,
 4959 ["Del"] = 8711,
 4960 ["nabla"] = 8711,
 4961 ["Element"] = 8712,
 4962 ["in"] = 8712,
 4963 ["isin"] = 8712,
 4964 ["isinv"] = 8712,
 4965 ["NotElement"] = 8713,
 4966 ["notin"] = 8713,
 4967 ["notinva"] = 8713,
 4968 ["ReverseElement"] = 8715,
 4969 ["SuchThat"] = 8715,
 4970 ["ni"] = 8715,
 4971 ["niv"] = 8715,
 4972 ["NotReverseElement"] = 8716,
 4973 ["notni"] = 8716,
 4974 ["notniva"] = 8716,
 4975 ["Product"] = 8719,
 4976 ["prod"] = 8719,
 4977 ["Coproduct"] = 8720,
 4978 ["coprod"] = 8720,
 4979 ["Sum"] = 8721,
 4980 ["sum"] = 8721,
 4981 ["minus"] = 8722,
 4982 ["MinusPlus"] = 8723,
 4983 ["mnplus"] = 8723,
 4984 ["mp"] = 8723,
 4985 ["dotplus"] = 8724,
 4986 ["plusdo"] = 8724,
 4987 ["Backslash"] = 8726,
 4988 ["setminus"] = 8726,
 4989 ["setmn"] = 8726,
 4990 ["smallsetminus"] = 8726,
 4991 ["ssetmn"] = 8726,
 4992 ["lowast"] = 8727,
 4993 ["SmallCircle"] = 8728,
 4994 ["compfn"] = 8728,
 4995 ["Sqrt"] = 8730,
 4996 ["radic"] = 8730,

4997 ["Proportional"] = 8733,
 4998 ["prop"] = 8733,
 4999 ["propto"] = 8733,
 5000 ["varpropto"] = 8733,
 5001 ["vprop"] = 8733,
 5002 ["infin"] = 8734,
 5003 ["angrt"] = 8735,
 5004 ["ang"] = 8736,
 5005 ["angle"] = 8736,
 5006 ["nang"] = {8736, 8402},
 5007 ["angmsd"] = 8737,
 5008 ["measuredangle"] = 8737,
 5009 ["angsph"] = 8738,
 5010 ["VerticalBar"] = 8739,
 5011 ["mid"] = 8739,
 5012 ["shortmid"] = 8739,
 5013 ["smid"] = 8739,
 5014 ["NotVerticalBar"] = 8740,
 5015 ["nmid"] = 8740,
 5016 ["nshortmid"] = 8740,
 5017 ["nsmid"] = 8740,
 5018 ["DoubleVerticalBar"] = 8741,
 5019 ["par"] = 8741,
 5020 ["parallel"] = 8741,
 5021 ["shortparallel"] = 8741,
 5022 ["spar"] = 8741,
 5023 ["NotDoubleVerticalBar"] = 8742,
 5024 ["npar"] = 8742,
 5025 ["nparallel"] = 8742,
 5026 ["nshortparallel"] = 8742,
 5027 ["nspar"] = 8742,
 5028 ["and"] = 8743,
 5029 ["wedge"] = 8743,
 5030 ["or"] = 8744,
 5031 ["vee"] = 8744,
 5032 ["cap"] = 8745,
 5033 ["caps"] = {8745, 65024},
 5034 ["cup"] = 8746,
 5035 ["cups"] = {8746, 65024},
 5036 ["Integral"] = 8747,
 5037 ["int"] = 8747,
 5038 ["Int"] = 8748,
 5039 ["iiint"] = 8749,
 5040 ["tint"] = 8749,
 5041 ["ContourIntegral"] = 8750,
 5042 ["conint"] = 8750,
 5043 ["oint"] = 8750,

5044 ["Conint"] = 8751,
5045 ["DoubleContourIntegral"] = 8751,
5046 ["Cconint"] = 8752,
5047 ["cwint"] = 8753,
5048 ["ClockwiseContourIntegral"] = 8754,
5049 ["cwconint"] = 8754,
5050 ["CounterClockwiseContourIntegral"] = 8755,
5051 ["awconint"] = 8755,
5052 ["Therefore"] = 8756,
5053 ["there4"] = 8756,
5054 ["therefore"] = 8756,
5055 ["Because"] = 8757,
5056 ["becaus"] = 8757,
5057 ["because"] = 8757,
5058 ["ratio"] = 8758,
5059 ["Colon"] = 8759,
5060 ["Proportion"] = 8759,
5061 ["dotminus"] = 8760,
5062 ["minusd"] = 8760,
5063 ["mDDot"] = 8762,
5064 ["homtht"] = 8763,
5065 ["Tilde"] = 8764,
5066 ["nvsim"] = {8764, 8402},
5067 ["sim"] = 8764,
5068 ["thicksim"] = 8764,
5069 ["thksim"] = 8764,
5070 ["backsim"] = 8765,
5071 ["bsim"] = 8765,
5072 ["race"] = {8765, 817},
5073 ["ac"] = 8766,
5074 ["acE"] = {8766, 819},
5075 ["mstpos"] = 8766,
5076 ["acd"] = 8767,
5077 ["VerticalTilde"] = 8768,
5078 ["wr"] = 8768,
5079 ["wreath"] = 8768,
5080 ["NotTilde"] = 8769,
5081 ["nsim"] = 8769,
5082 ["EqualTilde"] = 8770,
5083 ["NotEqualTilde"] = {8770, 824},
5084 ["eqsim"] = 8770,
5085 ["esim"] = 8770,
5086 ["nesim"] = {8770, 824},
5087 ["TildeEqual"] = 8771,
5088 ["sime"] = 8771,
5089 ["simeq"] = 8771,
5090 ["NotTildeEqual"] = 8772,

5091 ["nsime"] = 8772,
5092 ["nsimeq"] = 8772,
5093 ["TildeFullEqual"] = 8773,
5094 ["cong"] = 8773,
5095 ["simne"] = 8774,
5096 ["NotTildeFullEqual"] = 8775,
5097 ["ncong"] = 8775,
5098 ["TildeTilde"] = 8776,
5099 ["ap"] = 8776,
5100 ["approx"] = 8776,
5101 ["asymp"] = 8776,
5102 ["thickapprox"] = 8776,
5103 ["thkap"] = 8776,
5104 ["NotTildeTilde"] = 8777,
5105 ["nap"] = 8777,
5106 ["napprox"] = 8777,
5107 ["ape"] = 8778,
5108 ["approxpeq"] = 8778,
5109 ["apid"] = 8779,
5110 ["napid"] = {8779, 824},
5111 ["backcong"] = 8780,
5112 ["bcong"] = 8780,
5113 ["CupCap"] = 8781,
5114 ["asympeq"] = 8781,
5115 ["nvap"] = {8781, 8402},
5116 ["Bumpeq"] = 8782,
5117 ["HumpDownHump"] = 8782,
5118 ["NotHumpDownHump"] = {8782, 824},
5119 ["bump"] = 8782,
5120 ["nbump"] = {8782, 824},
5121 ["HumpEqual"] = 8783,
5122 ["NotHumpEqual"] = {8783, 824},
5123 ["bumpe"] = 8783,
5124 ["bumpeq"] = 8783,
5125 ["nbumpe"] = {8783, 824},
5126 ["DotEqual"] = 8784,
5127 ["doteq"] = 8784,
5128 ["esdot"] = 8784,
5129 ["nedot"] = {8784, 824},
5130 ["doteqdot"] = 8785,
5131 ["eDot"] = 8785,
5132 ["efDot"] = 8786,
5133 ["fallingdotseq"] = 8786,
5134 ["erDot"] = 8787,
5135 ["risingdotseq"] = 8787,
5136 ["Assign"] = 8788,
5137 ["colone"] = 8788,

5138 ["coloneq"] = 8788,
5139 ["ecolon"] = 8789,
5140 ["eqcolon"] = 8789,
5141 ["ecir"] = 8790,
5142 ["eqcirc"] = 8790,
5143 ["circeq"] = 8791,
5144 ["cire"] = 8791,
5145 ["wedgeq"] = 8793,
5146 ["veeeq"] = 8794,
5147 ["triangleq"] = 8796,
5148 ["trie"] = 8796,
5149 ["equest"] = 8799,
5150 ["questeq"] = 8799,
5151 ["NotEqual"] = 8800,
5152 ["ne"] = 8800,
5153 ["Congruent"] = 8801,
5154 ["bnequiv"] = {8801, 8421},
5155 ["equiv"] = 8801,
5156 ["NotCongruent"] = 8802,
5157 ["nequiv"] = 8802,
5158 ["le"] = 8804,
5159 ["leq"] = 8804,
5160 ["nvle"] = {8804, 8402},
5161 ["GreaterEqual"] = 8805,
5162 ["ge"] = 8805,
5163 ["geq"] = 8805,
5164 ["nvge"] = {8805, 8402},
5165 ["LessFullEqual"] = 8806,
5166 ["lE"] = 8806,
5167 ["leqq"] = 8806,
5168 ["nlE"] = {8806, 824},
5169 ["nleqq"] = {8806, 824},
5170 ["GreaterFullEqual"] = 8807,
5171 ["NotGreaterFullEqual"] = {8807, 824},
5172 ["gE"] = 8807,
5173 ["geqq"] = 8807,
5174 ["ngE"] = {8807, 824},
5175 ["ngeqq"] = {8807, 824},
5176 ["lnE"] = 8808,
5177 ["lneqq"] = 8808,
5178 ["lvertneqq"] = {8808, 65024},
5179 ["lvnE"] = {8808, 65024},
5180 ["gnE"] = 8809,
5181 ["gneqq"] = 8809,
5182 ["gvertneqq"] = {8809, 65024},
5183 ["gvnE"] = {8809, 65024},
5184 ["Lt"] = 8810,

5185 ["NestedLessLess"] = 8810,
5186 ["NotLessLess"] = {8810, 824},
5187 ["ll"] = 8810,
5188 ["nLt"] = {8810, 8402},
5189 ["nLtv"] = {8810, 824},
5190 ["Gt"] = 8811,
5191 ["NestedGreaterGreater"] = 8811,
5192 ["NotGreaterGreater"] = {8811, 824},
5193 ["gg"] = 8811,
5194 ["nGt"] = {8811, 8402},
5195 ["nGtv"] = {8811, 824},
5196 ["between"] = 8812,
5197 ["twixt"] = 8812,
5198 ["NotCupCap"] = 8813,
5199 ["NotLess"] = 8814,
5200 ["nless"] = 8814,
5201 ["nlt"] = 8814,
5202 ["NotGreater"] = 8815,
5203 ["ngt"] = 8815,
5204 ["ngtr"] = 8815,
5205 ["NotLessEqual"] = 8816,
5206 ["nle"] = 8816,
5207 ["nleq"] = 8816,
5208 ["NotGreaterEqual"] = 8817,
5209 ["nge"] = 8817,
5210 ["ngeq"] = 8817,
5211 ["LessTilde"] = 8818,
5212 ["lesssim"] = 8818,
5213 ["lsim"] = 8818,
5214 ["GreaterTilde"] = 8819,
5215 ["gsim"] = 8819,
5216 ["gtrsim"] = 8819,
5217 ["NotLessTilde"] = 8820,
5218 ["nlsim"] = 8820,
5219 ["NotGreaterTilde"] = 8821,
5220 ["ngsim"] = 8821,
5221 ["LessGreater"] = 8822,
5222 ["lessgtr"] = 8822,
5223 ["lg"] = 8822,
5224 ["GreaterLess"] = 8823,
5225 ["gl"] = 8823,
5226 ["gtrless"] = 8823,
5227 ["NotLessGreater"] = 8824,
5228 ["ntlgl"] = 8824,
5229 ["NotGreaterLess"] = 8825,
5230 ["ntgl"] = 8825,
5231 ["Precedes"] = 8826,

5232 ["pr"] = 8826,
5233 ["prec"] = 8826,
5234 ["Succeeds"] = 8827,
5235 ["sc"] = 8827,
5236 ["succ"] = 8827,
5237 ["PrecedesSlantEqual"] = 8828,
5238 ["prcue"] = 8828,
5239 ["preccurlyeq"] = 8828,
5240 ["SucceedsSlantEqual"] = 8829,
5241 ["sccue"] = 8829,
5242 ["succcurlyeq"] = 8829,
5243 ["PrecedesTilde"] = 8830,
5244 ["precsim"] = 8830,
5245 ["prsim"] = 8830,
5246 ["NotSucceedsTilde"] = {8831, 824},
5247 ["SucceedsTilde"] = 8831,
5248 ["scsim"] = 8831,
5249 ["succsim"] = 8831,
5250 ["NotPrecedes"] = 8832,
5251 ["npr"] = 8832,
5252 ["nprec"] = 8832,
5253 ["NotSucceeds"] = 8833,
5254 ["nsc"] = 8833,
5255 ["nsucc"] = 8833,
5256 ["NotSubset"] = {8834, 8402},
5257 ["nsubset"] = {8834, 8402},
5258 ["sub"] = 8834,
5259 ["subset"] = 8834,
5260 ["vnsup"] = {8834, 8402},
5261 ["NotSuperset"] = {8835, 8402},
5262 ["Superset"] = 8835,
5263 ["nsupset"] = {8835, 8402},
5264 ["sup"] = 8835,
5265 ["supset"] = 8835,
5266 ["vnsup"] = {8835, 8402},
5267 ["nsub"] = 8836,
5268 ["nsup"] = 8837,
5269 ["SubsetEqual"] = 8838,
5270 ["sube"] = 8838,
5271 ["subseteq"] = 8838,
5272 ["SupersetEqual"] = 8839,
5273 ["supe"] = 8839,
5274 ["supseteq"] = 8839,
5275 ["NotSubsetEqual"] = 8840,
5276 ["nsube"] = 8840,
5277 ["nsubseteq"] = 8840,
5278 ["NotSupersetEqual"] = 8841,

5279 ["nsupe"] = 8841,
5280 ["nsupseteq"] = 8841,
5281 ["subne"] = 8842,
5282 ["subsetneq"] = 8842,
5283 ["varsubsetneq"] = {8842, 65024},
5284 ["vsubne"] = {8842, 65024},
5285 ["supne"] = 8843,
5286 ["supsetneq"] = 8843,
5287 ["varsupsetneq"] = {8843, 65024},
5288 ["vsupne"] = {8843, 65024},
5289 ["cupdot"] = 8845,
5290 ["UnionPlus"] = 8846,
5291 ["uplus"] = 8846,
5292 ["NotSquareSubset"] = {8847, 824},
5293 ["SquareSubset"] = 8847,
5294 ["sqsub"] = 8847,
5295 ["sqsubset"] = 8847,
5296 ["NotSquareSuperset"] = {8848, 824},
5297 ["SquareSuperset"] = 8848,
5298 ["sqsup"] = 8848,
5299 ["sqsupset"] = 8848,
5300 ["SquareSubsetEqual"] = 8849,
5301 ["sqsube"] = 8849,
5302 ["sqsubseteq"] = 8849,
5303 ["SquareSupersetEqual"] = 8850,
5304 ["sqsupe"] = 8850,
5305 ["sqsupseteq"] = 8850,
5306 ["SquareIntersection"] = 8851,
5307 ["sqcap"] = 8851,
5308 ["sqcaps"] = {8851, 65024},
5309 ["SquareUnion"] = 8852,
5310 ["sqcup"] = 8852,
5311 ["sqcups"] = {8852, 65024},
5312 ["CirclePlus"] = 8853,
5313 ["oplus"] = 8853,
5314 ["CircleMinus"] = 8854,
5315 ["ominus"] = 8854,
5316 ["CircleTimes"] = 8855,
5317 ["otimes"] = 8855,
5318 ["osol"] = 8856,
5319 ["CircleDot"] = 8857,
5320 ["odot"] = 8857,
5321 ["circledcirc"] = 8858,
5322 ["ocir"] = 8858,
5323 ["circledast"] = 8859,
5324 ["oast"] = 8859,
5325 ["circleddash"] = 8861,

5326 ["odash"] = 8861,
5327 ["boxplus"] = 8862,
5328 ["plusb"] = 8862,
5329 ["boxminus"] = 8863,
5330 ["minusb"] = 8863,
5331 ["boxtimes"] = 8864,
5332 ["timesb"] = 8864,
5333 ["dotsquare"] = 8865,
5334 ["sdotb"] = 8865,
5335 ["RightTee"] = 8866,
5336 ["vdash"] = 8866,
5337 ["LeftTee"] = 8867,
5338 ["dashv"] = 8867,
5339 ["DownTee"] = 8868,
5340 ["top"] = 8868,
5341 ["UpTee"] = 8869,
5342 ["bot"] = 8869,
5343 ["bottom"] = 8869,
5344 ["perp"] = 8869,
5345 ["models"] = 8871,
5346 ["DoubleRightTee"] = 8872,
5347 ["vDash"] = 8872,
5348 ["VDash"] = 8873,
5349 ["Vvdash"] = 8874,
5350 ["VDash"] = 8875,
5351 ["nvdash"] = 8876,
5352 ["nvDash"] = 8877,
5353 ["nVdash"] = 8878,
5354 ["nVDash"] = 8879,
5355 ["prurel"] = 8880,
5356 ["LeftTriangle"] = 8882,
5357 ["vartriangleleft"] = 8882,
5358 ["vltri"] = 8882,
5359 ["RightTriangle"] = 8883,
5360 ["vartriangleright"] = 8883,
5361 ["vrtri"] = 8883,
5362 ["LeftTriangleEqual"] = 8884,
5363 ["ltrie"] = 8884,
5364 ["nvltrie"] = {8884, 8402},
5365 ["trianglelefteq"] = 8884,
5366 ["RightTriangleEqual"] = 8885,
5367 ["nvrtrie"] = {8885, 8402},
5368 ["rtrie"] = 8885,
5369 ["trianglerighteq"] = 8885,
5370 ["origof"] = 8886,
5371 ["imof"] = 8887,
5372 ["multimap"] = 8888,

5373 ["mumap"] = 8888,
5374 ["hercon"] = 8889,
5375 ["intcal"] = 8890,
5376 ["intercal"] = 8890,
5377 ["veebar"] = 8891,
5378 ["barvee"] = 8893,
5379 ["angrtvb"] = 8894,
5380 ["lrtri"] = 8895,
5381 ["Wedge"] = 8896,
5382 ["bigwedge"] = 8896,
5383 ["xwedge"] = 8896,
5384 ["Vee"] = 8897,
5385 ["bigvee"] = 8897,
5386 ["xvee"] = 8897,
5387 ["Intersection"] = 8898,
5388 ["bigcap"] = 8898,
5389 ["xcap"] = 8898,
5390 ["Union"] = 8899,
5391 ["bigcup"] = 8899,
5392 ["xcup"] = 8899,
5393 ["Diamond"] = 8900,
5394 ["diam"] = 8900,
5395 ["diamond"] = 8900,
5396 ["sdot"] = 8901,
5397 ["Star"] = 8902,
5398 ["sstarf"] = 8902,
5399 ["divideontimes"] = 8903,
5400 ["divonx"] = 8903,
5401 ["bowtie"] = 8904,
5402 ["ltimes"] = 8905,
5403 ["rtimes"] = 8906,
5404 ["leftthreetimes"] = 8907,
5405 ["lthree"] = 8907,
5406 ["rightthreetimes"] = 8908,
5407 ["rthree"] = 8908,
5408 ["backsimeq"] = 8909,
5409 ["bsime"] = 8909,
5410 ["curlyvee"] = 8910,
5411 ["cuvee"] = 8910,
5412 ["curlywedge"] = 8911,
5413 ["cuwed"] = 8911,
5414 ["Sub"] = 8912,
5415 ["Subset"] = 8912,
5416 ["Sup"] = 8913,
5417 ["Supset"] = 8913,
5418 ["Cap"] = 8914,
5419 ["Cup"] = 8915,

5420 ["fork"] = 8916,
5421 ["pitchfork"] = 8916,
5422 ["epar"] = 8917,
5423 ["lessdot"] = 8918,
5424 ["ltdot"] = 8918,
5425 ["gtdot"] = 8919,
5426 ["gtrdot"] = 8919,
5427 ["Ll"] = 8920,
5428 ["nLl"] = {8920, 824},
5429 ["Gg"] = 8921,
5430 ["ggg"] = 8921,
5431 ["nGg"] = {8921, 824},
5432 ["LessEqualGreater"] = 8922,
5433 ["leg"] = 8922,
5434 ["lesg"] = {8922, 65024},
5435 ["lesseqgtr"] = 8922,
5436 ["GreaterEqualLess"] = 8923,
5437 ["gel"] = 8923,
5438 ["gesl"] = {8923, 65024},
5439 ["gtreqless"] = 8923,
5440 ["cuepr"] = 8926,
5441 ["curlyeqprec"] = 8926,
5442 ["cuesc"] = 8927,
5443 ["curlyeqsucc"] = 8927,
5444 ["NotPrecedesSlantEqual"] = 8928,
5445 ["nprcue"] = 8928,
5446 ["NotSucceedsSlantEqual"] = 8929,
5447 ["nsccue"] = 8929,
5448 ["NotSquareSubsetEqual"] = 8930,
5449 ["nsqsube"] = 8930,
5450 ["NotSquareSupersetEqual"] = 8931,
5451 ["nsqsupe"] = 8931,
5452 ["lnsim"] = 8934,
5453 ["gnsim"] = 8935,
5454 ["precnsim"] = 8936,
5455 ["prnsim"] = 8936,
5456 ["scnsim"] = 8937,
5457 ["succnsim"] = 8937,
5458 ["NotLeftTriangle"] = 8938,
5459 ["nltri"] = 8938,
5460 ["ntriangleleft"] = 8938,
5461 ["NotRightTriangle"] = 8939,
5462 ["nrtri"] = 8939,
5463 ["ntriangleright"] = 8939,
5464 ["NotLeftTriangleEqual"] = 8940,
5465 ["nltrie"] = 8940,
5466 ["ntrianglelefteq"] = 8940,

5467 ["NotRightTriangleEqual"] = 8941,
5468 ["nrtrie"] = 8941,
5469 ["ntrianglerighteq"] = 8941,
5470 ["vellip"] = 8942,
5471 ["ctdot"] = 8943,
5472 ["utdot"] = 8944,
5473 ["dtdot"] = 8945,
5474 ["disin"] = 8946,
5475 ["isinsv"] = 8947,
5476 ["isins"] = 8948,
5477 ["isindot"] = 8949,
5478 ["notindot"] = {8949, 824},
5479 ["notinvc"] = 8950,
5480 ["notinvb"] = 8951,
5481 ["isinE"] = 8953,
5482 ["notinE"] = {8953, 824},
5483 ["nisd"] = 8954,
5484 ["xnis"] = 8955,
5485 ["nis"] = 8956,
5486 ["notnivc"] = 8957,
5487 ["notnivb"] = 8958,
5488 ["barwed"] = 8965,
5489 ["barwedge"] = 8965,
5490 ["Barwed"] = 8966,
5491 ["doublebarwedge"] = 8966,
5492 ["LeftCeiling"] = 8968,
5493 ["lceil"] = 8968,
5494 ["RightCeiling"] = 8969,
5495 ["rceil"] = 8969,
5496 ["LeftFloor"] = 8970,
5497 ["lfloor"] = 8970,
5498 ["RightFloor"] = 8971,
5499 ["rfloor"] = 8971,
5500 ["drcrop"] = 8972,
5501 ["dlcrop"] = 8973,
5502 ["urcrop"] = 8974,
5503 ["ulcrop"] = 8975,
5504 ["bnot"] = 8976,
5505 ["proflin"] = 8978,
5506 ["profsurf"] = 8979,
5507 ["telrec"] = 8981,
5508 ["target"] = 8982,
5509 ["ulcorn"] = 8988,
5510 ["ulcorner"] = 8988,
5511 ["urcorn"] = 8989,
5512 ["urcorner"] = 8989,
5513 ["dlcorn"] = 8990,

5514 ["llcorner"] = 8990,
5515 ["drcorn"] = 8991,
5516 ["lrcorner"] = 8991,
5517 ["frown"] = 8994,
5518 ["sfrown"] = 8994,
5519 ["smile"] = 8995,
5520 ["ssmile"] = 8995,
5521 ["cylcty"] = 9005,
5522 ["profalar"] = 9006,
5523 ["topbot"] = 9014,
5524 ["ovbar"] = 9021,
5525 ["solbar"] = 9023,
5526 ["angzarr"] = 9084,
5527 ["lmoust"] = 9136,
5528 ["lmoustache"] = 9136,
5529 ["rmoust"] = 9137,
5530 ["rmoustache"] = 9137,
5531 ["OverBracket"] = 9140,
5532 ["tbrk"] = 9140,
5533 ["UnderBracket"] = 9141,
5534 ["bbrk"] = 9141,
5535 ["bbrktbrk"] = 9142,
5536 ["OverParenthesis"] = 9180,
5537 ["UnderParenthesis"] = 9181,
5538 ["OverBrace"] = 9182,
5539 ["UnderBrace"] = 9183,
5540 ["trpezium"] = 9186,
5541 ["elinters"] = 9191,
5542 ["blank"] = 9251,
5543 ["circledS"] = 9416,
5544 ["oS"] = 9416,
5545 ["HorizontalLine"] = 9472,
5546 ["boxh"] = 9472,
5547 ["boxv"] = 9474,
5548 ["boxdr"] = 9484,
5549 ["boxdl"] = 9488,
5550 ["boxur"] = 9492,
5551 ["boxul"] = 9496,
5552 ["boxvr"] = 9500,
5553 ["boxvl"] = 9508,
5554 ["boxhd"] = 9516,
5555 ["boxhu"] = 9524,
5556 ["boxvh"] = 9532,
5557 ["boxH"] = 9552,
5558 ["boxV"] = 9553,
5559 ["boxdR"] = 9554,
5560 ["boxDr"] = 9555,

5561 ["boxDR"] = 9556,
5562 ["boxdL"] = 9557,
5563 ["boxDl"] = 9558,
5564 ["boxDL"] = 9559,
5565 ["boxuR"] = 9560,
5566 ["boxUr"] = 9561,
5567 ["boxUR"] = 9562,
5568 ["boxuL"] = 9563,
5569 ["boxUl"] = 9564,
5570 ["boxUL"] = 9565,
5571 ["boxvR"] = 9566,
5572 ["boxVr"] = 9567,
5573 ["boxVR"] = 9568,
5574 ["boxvL"] = 9569,
5575 ["boxVl"] = 9570,
5576 ["boxVL"] = 9571,
5577 ["boxHd"] = 9572,
5578 ["boxhD"] = 9573,
5579 ["boxHD"] = 9574,
5580 ["boxHu"] = 9575,
5581 ["boxhU"] = 9576,
5582 ["boxHU"] = 9577,
5583 ["boxvH"] = 9578,
5584 ["boxVh"] = 9579,
5585 ["boxVH"] = 9580,
5586 ["uhblk"] = 9600,
5587 ["lhblk"] = 9604,
5588 ["block"] = 9608,
5589 ["blk14"] = 9617,
5590 ["blk12"] = 9618,
5591 ["blk34"] = 9619,
5592 ["Square"] = 9633,
5593 ["squ"] = 9633,
5594 ["square"] = 9633,
5595 ["FilledVerySmallSquare"] = 9642,
5596 ["blacksquare"] = 9642,
5597 ["squarf"] = 9642,
5598 ["squf"] = 9642,
5599 ["EmptyVerySmallSquare"] = 9643,
5600 ["rect"] = 9645,
5601 ["marker"] = 9646,
5602 ["fltns"] = 9649,
5603 ["bigtriangleup"] = 9651,
5604 ["xutri"] = 9651,
5605 ["blacktriangle"] = 9652,
5606 ["utrif"] = 9652,
5607 ["triangle"] = 9653,

5608 ["utri"] = 9653,
5609 ["blacktriangleright"] = 9656,
5610 ["rtrif"] = 9656,
5611 ["rtri"] = 9657,
5612 ["triangleright"] = 9657,
5613 ["bigtriangledown"] = 9661,
5614 ["xdtri"] = 9661,
5615 ["blacktriangledown"] = 9662,
5616 ["dtrif"] = 9662,
5617 ["dtri"] = 9663,
5618 ["triangledown"] = 9663,
5619 ["blacktriangleleft"] = 9666,
5620 ["ltrif"] = 9666,
5621 ["ltri"] = 9667,
5622 ["triangleleft"] = 9667,
5623 ["loz"] = 9674,
5624 ["lozenge"] = 9674,
5625 ["cir"] = 9675,
5626 ["tridot"] = 9708,
5627 ["bigcirc"] = 9711,
5628 ["xcirc"] = 9711,
5629 ["ultri"] = 9720,
5630 ["urtri"] = 9721,
5631 ["lltri"] = 9722,
5632 ["EmptySmallSquare"] = 9723,
5633 ["FilledSmallSquare"] = 9724,
5634 ["bigstar"] = 9733,
5635 ["starf"] = 9733,
5636 ["star"] = 9734,
5637 ["phone"] = 9742,
5638 ["female"] = 9792,
5639 ["male"] = 9794,
5640 ["spades"] = 9824,
5641 ["spadesuit"] = 9824,
5642 ["clubs"] = 9827,
5643 ["clubsuit"] = 9827,
5644 ["hearts"] = 9829,
5645 ["heartsuit"] = 9829,
5646 ["diamondsuit"] = 9830,
5647 ["diams"] = 9830,
5648 ["sung"] = 9834,
5649 ["flat"] = 9837,
5650 ["natur"] = 9838,
5651 ["natural"] = 9838,
5652 ["sharp"] = 9839,
5653 ["check"] = 10003,
5654 ["checkmark"] = 10003,

5655 ["cross"] = 10007,
5656 ["malt"] = 10016,
5657 ["maltese"] = 10016,
5658 ["sext"] = 10038,
5659 ["VerticalSeparator"] = 10072,
5660 ["lbrk"] = 10098,
5661 ["rbrk"] = 10099,
5662 ["bsolhsub"] = 10184,
5663 ["suphsol"] = 10185,
5664 ["LeftDoubleBracket"] = 10214,
5665 ["lobrk"] = 10214,
5666 ["RightDoubleBracket"] = 10215,
5667 ["robrk"] = 10215,
5668 ["LeftAngleBracket"] = 10216,
5669 ["lang"] = 10216,
5670 ["langle"] = 10216,
5671 ["RightAngleBracket"] = 10217,
5672 ["rang"] = 10217,
5673 ["rangle"] = 10217,
5674 ["Lang"] = 10218,
5675 ["Rang"] = 10219,
5676 ["loang"] = 10220,
5677 ["roang"] = 10221,
5678 ["LongLeftArrow"] = 10229,
5679 ["longleftarrow"] = 10229,
5680 ["xlarr"] = 10229,
5681 ["LongRightArrow"] = 10230,
5682 ["longrightarrow"] = 10230,
5683 ["xrarr"] = 10230,
5684 ["LongLeftRightArrow"] = 10231,
5685 ["longleftrightarrow"] = 10231,
5686 ["xharr"] = 10231,
5687 ["DoubleLongLeftArrow"] = 10232,
5688 ["Longleftarrow"] = 10232,
5689 ["xlArr"] = 10232,
5690 ["DoubleLongRightArrow"] = 10233,
5691 ["Longrightarrow"] = 10233,
5692 ["xrArr"] = 10233,
5693 ["DoubleLongLeftRightArrow"] = 10234,
5694 ["Longleftrightarrow"] = 10234,
5695 ["xhArr"] = 10234,
5696 ["longmapsto"] = 10236,
5697 ["xmap"] = 10236,
5698 ["dzigrarr"] = 10239,
5699 ["nvlArr"] = 10498,
5700 ["nvrArr"] = 10499,
5701 ["nvHarr"] = 10500,

5702 ["Map"] = 10501,
5703 ["lbarr"] = 10508,
5704 ["bkarow"] = 10509,
5705 ["rbarr"] = 10509,
5706 ["lBarr"] = 10510,
5707 ["dbkarow"] = 10511,
5708 ["rBarr"] = 10511,
5709 ["RBarr"] = 10512,
5710 ["drbkarow"] = 10512,
5711 ["DDotrahd"] = 10513,
5712 ["UpArrowBar"] = 10514,
5713 ["DownArrowBar"] = 10515,
5714 ["Rarrtl"] = 10518,
5715 ["latail"] = 10521,
5716 ["ratail"] = 10522,
5717 ["lAtail"] = 10523,
5718 ["rAtail"] = 10524,
5719 ["larrfs"] = 10525,
5720 ["rarrfs"] = 10526,
5721 ["larrbfs"] = 10527,
5722 ["rarrbfs"] = 10528,
5723 ["nwarhk"] = 10531,
5724 ["nearhk"] = 10532,
5725 ["hksearow"] = 10533,
5726 ["searhk"] = 10533,
5727 ["hkswarow"] = 10534,
5728 ["swarhk"] = 10534,
5729 ["nwnear"] = 10535,
5730 ["nesear"] = 10536,
5731 ["toea"] = 10536,
5732 ["seswar"] = 10537,
5733 ["tosa"] = 10537,
5734 ["swnwar"] = 10538,
5735 ["nrarrc"] = {10547, 824},
5736 ["rarrc"] = 10547,
5737 ["cudarr"] = 10549,
5738 ["ldca"] = 10550,
5739 ["rdca"] = 10551,
5740 ["cudarrl"] = 10552,
5741 ["larrpl"] = 10553,
5742 ["curarrm"] = 10556,
5743 ["cularrp"] = 10557,
5744 ["rarrpl"] = 10565,
5745 ["harrcir"] = 10568,
5746 ["Uarroccir"] = 10569,
5747 ["lurdshar"] = 10570,
5748 ["ldrushar"] = 10571,

5749 ["LeftRightVector"] = 10574,
5750 ["RightUpDownVector"] = 10575,
5751 ["DownLeftRightVector"] = 10576,
5752 ["LeftUpDownVector"] = 10577,
5753 ["LeftVectorBar"] = 10578,
5754 ["RightVectorBar"] = 10579,
5755 ["RightUpVectorBar"] = 10580,
5756 ["RightDownVectorBar"] = 10581,
5757 ["DownLeftVectorBar"] = 10582,
5758 ["DownRightVectorBar"] = 10583,
5759 ["LeftUpVectorBar"] = 10584,
5760 ["LeftDownVectorBar"] = 10585,
5761 ["LeftTeeVector"] = 10586,
5762 ["RightTeeVector"] = 10587,
5763 ["RightUpTeeVector"] = 10588,
5764 ["RightDownTeeVector"] = 10589,
5765 ["DownLeftTeeVector"] = 10590,
5766 ["DownRightTeeVector"] = 10591,
5767 ["LeftUpTeeVector"] = 10592,
5768 ["LeftDownTeeVector"] = 10593,
5769 ["lHar"] = 10594,
5770 ["uHar"] = 10595,
5771 ["rHar"] = 10596,
5772 ["dHar"] = 10597,
5773 ["luruhar"] = 10598,
5774 ["ldrdhar"] = 10599,
5775 ["ruluhar"] = 10600,
5776 ["rdldhar"] = 10601,
5777 ["lharul"] = 10602,
5778 ["llhard"] = 10603,
5779 ["rharul"] = 10604,
5780 ["lrhard"] = 10605,
5781 ["UpEquilibrium"] = 10606,
5782 ["udhar"] = 10606,
5783 ["ReverseUpEquilibrium"] = 10607,
5784 ["duhar"] = 10607,
5785 ["RoundImplies"] = 10608,
5786 ["erarr"] = 10609,
5787 ["simrarr"] = 10610,
5788 ["larrsim"] = 10611,
5789 ["rarrsim"] = 10612,
5790 ["rarrap"] = 10613,
5791 ["ltlarr"] = 10614,
5792 ["gtrarr"] = 10616,
5793 ["subrarr"] = 10617,
5794 ["suplarr"] = 10619,
5795 ["lfisht"] = 10620,

5796 ["rfisht"] = 10621,
5797 ["ufisht"] = 10622,
5798 ["dfisht"] = 10623,
5799 ["lopar"] = 10629,
5800 ["ropar"] = 10630,
5801 ["lbrke"] = 10635,
5802 ["rbrke"] = 10636,
5803 ["lbrkslu"] = 10637,
5804 ["rbrksld"] = 10638,
5805 ["lbrksld"] = 10639,
5806 ["rbrkslu"] = 10640,
5807 ["langd"] = 10641,
5808 ["rangd"] = 10642,
5809 ["lparlt"] = 10643,
5810 ["rpargt"] = 10644,
5811 ["gtlPar"] = 10645,
5812 ["ltrPar"] = 10646,
5813 ["vzigzag"] = 10650,
5814 ["vangrt"] = 10652,
5815 ["angrtvbd"] = 10653,
5816 ["ange"] = 10660,
5817 ["range"] = 10661,
5818 ["dwangle"] = 10662,
5819 ["uwangle"] = 10663,
5820 ["angmsdaa"] = 10664,
5821 ["angmsdab"] = 10665,
5822 ["angmsdac"] = 10666,
5823 ["angmsdad"] = 10667,
5824 ["angmsdae"] = 10668,
5825 ["angmsdaf"] = 10669,
5826 ["angmsdag"] = 10670,
5827 ["angmsdah"] = 10671,
5828 ["bemptyv"] = 10672,
5829 ["demptyv"] = 10673,
5830 ["cemptyv"] = 10674,
5831 ["raemptyv"] = 10675,
5832 ["laemptyv"] = 10676,
5833 ["ohbar"] = 10677,
5834 ["omid"] = 10678,
5835 ["opar"] = 10679,
5836 ["operp"] = 10681,
5837 ["olcross"] = 10683,
5838 ["odsold"] = 10684,
5839 ["olcir"] = 10686,
5840 ["ofcir"] = 10687,
5841 ["olt"] = 10688,
5842 ["ogt"] = 10689,

5843 ["cirscir"] = 10690,
5844 ["cirE"] = 10691,
5845 ["solb"] = 10692,
5846 ["bsolb"] = 10693,
5847 ["boxbox"] = 10697,
5848 ["trisb"] = 10701,
5849 ["rtriltri"] = 10702,
5850 ["LeftTriangleBar"] = 10703,
5851 ["NotLeftTriangleBar"] = {10703, 824},
5852 ["NotRightTriangleBar"] = {10704, 824},
5853 ["RightTriangleBar"] = 10704,
5854 ["iinfin"] = 10716,
5855 ["infintie"] = 10717,
5856 ["nvinfin"] = 10718,
5857 ["eparsl"] = 10723,
5858 ["smeparsl"] = 10724,
5859 ["eqvparsl"] = 10725,
5860 ["blacklozenge"] = 10731,
5861 ["lozf"] = 10731,
5862 ["RuleDelayed"] = 10740,
5863 ["dsol"] = 10742,
5864 ["bigodot"] = 10752,
5865 ["xodot"] = 10752,
5866 ["bigoplus"] = 10753,
5867 ["xoplus"] = 10753,
5868 ["bigotimes"] = 10754,
5869 ["xotime"] = 10754,
5870 ["biguplus"] = 10756,
5871 ["xuplus"] = 10756,
5872 ["bigscup"] = 10758,
5873 ["xscup"] = 10758,
5874 ["iiiint"] = 10764,
5875 ["qint"] = 10764,
5876 ["fpartint"] = 10765,
5877 ["cirfnint"] = 10768,
5878 ["awint"] = 10769,
5879 ["rppolint"] = 10770,
5880 ["scpolint"] = 10771,
5881 ["npolint"] = 10772,
5882 ["pointint"] = 10773,
5883 ["quatint"] = 10774,
5884 ["intlarhk"] = 10775,
5885 ["pluscir"] = 10786,
5886 ["plusacir"] = 10787,
5887 ["simplus"] = 10788,
5888 ["plusdu"] = 10789,
5889 ["plussim"] = 10790,

5890 ["plustwo"] = 10791,
5891 ["mcomma"] = 10793,
5892 ["minusdu"] = 10794,
5893 ["loplus"] = 10797,
5894 ["roplus"] = 10798,
5895 ["Cross"] = 10799,
5896 ["timesd"] = 10800,
5897 ["timesbar"] = 10801,
5898 ["smashp"] = 10803,
5899 ["lotimes"] = 10804,
5900 ["rotimes"] = 10805,
5901 ["otimesas"] = 10806,
5902 ["Otimes"] = 10807,
5903 ["odiv"] = 10808,
5904 ["triplus"] = 10809,
5905 ["triminus"] = 10810,
5906 ["tritime"] = 10811,
5907 ["intprod"] = 10812,
5908 ["iprod"] = 10812,
5909 ["amalg"] = 10815,
5910 ["capdot"] = 10816,
5911 ["ncup"] = 10818,
5912 ["ncap"] = 10819,
5913 ["capand"] = 10820,
5914 ["cupor"] = 10821,
5915 ["cupcap"] = 10822,
5916 ["capcup"] = 10823,
5917 ["cupbrcap"] = 10824,
5918 ["capbrcup"] = 10825,
5919 ["cupcup"] = 10826,
5920 ["capcap"] = 10827,
5921 ["ccups"] = 10828,
5922 ["ccaps"] = 10829,
5923 ["ccupssm"] = 10832,
5924 ["And"] = 10835,
5925 ["Or"] = 10836,
5926 ["andand"] = 10837,
5927 ["oror"] = 10838,
5928 ["orslope"] = 10839,
5929 ["andslope"] = 10840,
5930 ["andv"] = 10842,
5931 ["orv"] = 10843,
5932 ["andd"] = 10844,
5933 ["ord"] = 10845,
5934 ["wedbar"] = 10847,
5935 ["sdote"] = 10854,
5936 ["simdot"] = 10858,

5937 ["congdot"] = 10861,
5938 ["ncongdot"] = {10861, 824},
5939 ["easter"] = 10862,
5940 ["apacir"] = 10863,
5941 ["apE"] = 10864,
5942 ["napE"] = {10864, 824},
5943 ["eplus"] = 10865,
5944 ["pluse"] = 10866,
5945 ["Esim"] = 10867,
5946 ["Colone"] = 10868,
5947 ["Equal"] = 10869,
5948 ["ddotseq"] = 10871,
5949 ["eDDot"] = 10871,
5950 ["equivDD"] = 10872,
5951 ["ltcir"] = 10873,
5952 ["gtcir"] = 10874,
5953 ["ltquest"] = 10875,
5954 ["gtquest"] = 10876,
5955 ["LessSlantEqual"] = 10877,
5956 ["NotLessSlantEqual"] = {10877, 824},
5957 ["leqslant"] = 10877,
5958 ["les"] = 10877,
5959 ["nleqslant"] = {10877, 824},
5960 ["nles"] = {10877, 824},
5961 ["GreaterSlantEqual"] = 10878,
5962 ["NotGreaterSlantEqual"] = {10878, 824},
5963 ["geqslant"] = 10878,
5964 ["ges"] = 10878,
5965 ["ngeqslant"] = {10878, 824},
5966 ["nges"] = {10878, 824},
5967 ["lesdot"] = 10879,
5968 ["gesdot"] = 10880,
5969 ["lesdoto"] = 10881,
5970 ["gesdoto"] = 10882,
5971 ["lesdotor"] = 10883,
5972 ["gesdoto1"] = 10884,
5973 ["lap"] = 10885,
5974 ["lessapprox"] = 10885,
5975 ["gap"] = 10886,
5976 ["gtrapprox"] = 10886,
5977 ["lne"] = 10887,
5978 ["lneq"] = 10887,
5979 ["gne"] = 10888,
5980 ["gneq"] = 10888,
5981 ["lnap"] = 10889,
5982 ["lnapprox"] = 10889,
5983 ["gnap"] = 10890,


```

5984 ["gnapprox"] = 10890,
5985 ["lEg"] = 10891,
5986 ["lesseqgtr"] = 10891,
5987 ["gEl"] = 10892,
5988 ["gtreqqlless"] = 10892,
5989 ["lsime"] = 10893,
5990 ["gsime"] = 10894,
5991 ["lsimg"] = 10895,
5992 ["gsiml"] = 10896,
5993 ["lgE"] = 10897,
5994 ["glE"] = 10898,
5995 ["lesges"] = 10899,
5996 ["gesles"] = 10900,
5997 ["els"] = 10901,
5998 ["eqslantless"] = 10901,
5999 ["egs"] = 10902,
6000 ["eqslantgtr"] = 10902,
6001 ["elsdot"] = 10903,
6002 ["egsdot"] = 10904,
6003 ["el"] = 10905,
6004 ["eg"] = 10906,
6005 ["siml"] = 10909,
6006 ["simg"] = 10910,
6007 ["simlE"] = 10911,
6008 ["simgE"] = 10912,
6009 ["LessLess"] = 10913,
6010 ["NotNestedLessLess"] = {10913, 824},
6011 ["GreaterGreater"] = 10914,
6012 ["NotNestedGreaterGreater"] = {10914, 824},
6013 ["glj"] = 10916,
6014 ["gla"] = 10917,
6015 ["ltcc"] = 10918,
6016 ["gtcc"] = 10919,
6017 ["lescc"] = 10920,
6018 ["gescc"] = 10921,
6019 ["smt"] = 10922,
6020 ["lat"] = 10923,
6021 ["smte"] = 10924,
6022 ["smtes"] = {10924, 65024},
6023 ["late"] = 10925,
6024 ["lates"] = {10925, 65024},
6025 ["bumpE"] = 10926,
6026 ["NotPrecedesEqual"] = {10927, 824},
6027 ["PrecedesEqual"] = 10927,
6028 ["npre"] = {10927, 824},
6029 ["npreceq"] = {10927, 824},
6030 ["pre"] = 10927,

```

6031 ["preceq"] = 10927,
6032 ["NotSucceedsEqual"] = {10928, 824},
6033 ["SucceedsEqual"] = 10928,
6034 ["nsce"] = {10928, 824},
6035 ["nsucceq"] = {10928, 824},
6036 ["sce"] = 10928,
6037 ["succeq"] = 10928,
6038 ["prE"] = 10931,
6039 ["scE"] = 10932,
6040 ["precneqq"] = 10933,
6041 ["prnE"] = 10933,
6042 ["scnE"] = 10934,
6043 ["succneqq"] = 10934,
6044 ["prap"] = 10935,
6045 ["precapprox"] = 10935,
6046 ["scap"] = 10936,
6047 ["succapprox"] = 10936,
6048 ["precnapprox"] = 10937,
6049 ["prnap"] = 10937,
6050 ["scnap"] = 10938,
6051 ["succnapprox"] = 10938,
6052 ["Pr"] = 10939,
6053 ["Sc"] = 10940,
6054 ["subdot"] = 10941,
6055 ["supdot"] = 10942,
6056 ["subplus"] = 10943,
6057 ["supplus"] = 10944,
6058 ["submult"] = 10945,
6059 ["supmult"] = 10946,
6060 ["subedot"] = 10947,
6061 ["supedot"] = 10948,
6062 ["nsubE"] = {10949, 824},
6063 ["nsubseteqq"] = {10949, 824},
6064 ["subE"] = 10949,
6065 ["subseteqq"] = 10949,
6066 ["nsupE"] = {10950, 824},
6067 ["nsupseteqq"] = {10950, 824},
6068 ["supE"] = 10950,
6069 ["supseteqq"] = 10950,
6070 ["subsim"] = 10951,
6071 ["supsim"] = 10952,
6072 ["subnE"] = 10955,
6073 ["subsetneqq"] = 10955,
6074 ["varsubsetneqq"] = {10955, 65024},
6075 ["vsubnE"] = {10955, 65024},
6076 ["supnE"] = 10956,
6077 ["supsetneqq"] = 10956,

6078 ["varsupsetneqq"] = {10956, 65024},
6079 ["vsupnE"] = {10956, 65024},
6080 ["csub"] = 10959,
6081 ["csup"] = 10960,
6082 ["csube"] = 10961,
6083 ["csupe"] = 10962,
6084 ["subsup"] = 10963,
6085 ["supsub"] = 10964,
6086 ["subsub"] = 10965,
6087 ["supsup"] = 10966,
6088 ["suphsub"] = 10967,
6089 ["supdsub"] = 10968,
6090 ["forkv"] = 10969,
6091 ["topfork"] = 10970,
6092 ["mlcp"] = 10971,
6093 ["Dashv"] = 10980,
6094 ["DoubleLeftTee"] = 10980,
6095 ["Vdashl"] = 10982,
6096 ["Barv"] = 10983,
6097 ["vBar"] = 10984,
6098 ["vBarv"] = 10985,
6099 ["Vbar"] = 10987,
6100 ["Not"] = 10988,
6101 ["bNot"] = 10989,
6102 ["rnmid"] = 10990,
6103 ["cirmid"] = 10991,
6104 ["midcir"] = 10992,
6105 ["topcir"] = 10993,
6106 ["nhpar"] = 10994,
6107 ["parsim"] = 10995,
6108 ["nparsl"] = {11005, 8421},
6109 ["parsl"] = 11005,
6110 ["fflig"] = 64256,
6111 ["filig"] = 64257,
6112 ["fllig"] = 64258,
6113 ["ffilig"] = 64259,
6114 ["ffllig"] = 64260,
6115 ["Ascr"] = 119964,
6116 ["Cscr"] = 119966,
6117 ["Dscr"] = 119967,
6118 ["Gscr"] = 119970,
6119 ["Jscr"] = 119973,
6120 ["Kscr"] = 119974,
6121 ["Nscr"] = 119977,
6122 ["Oscr"] = 119978,
6123 ["Pscr"] = 119979,
6124 ["Qscr"] = 119980,

6125 ["Sscr"] = 119982,
6126 ["Tscr"] = 119983,
6127 ["Uscr"] = 119984,
6128 ["Vscr"] = 119985,
6129 ["Wscr"] = 119986,
6130 ["Xscr"] = 119987,
6131 ["Yscr"] = 119988,
6132 ["Zscr"] = 119989,
6133 ["ascr"] = 119990,
6134 ["bscr"] = 119991,
6135 ["cscr"] = 119992,
6136 ["dscr"] = 119993,
6137 ["fscr"] = 119995,
6138 ["hscr"] = 119997,
6139 ["iscr"] = 119998,
6140 ["jscr"] = 119999,
6141 ["kscr"] = 120000,
6142 ["lscr"] = 120001,
6143 ["mscr"] = 120002,
6144 ["nscr"] = 120003,
6145 ["pscr"] = 120005,
6146 ["qscr"] = 120006,
6147 ["rscr"] = 120007,
6148 ["sscr"] = 120008,
6149 ["tscr"] = 120009,
6150 ["uscr"] = 120010,
6151 ["vscr"] = 120011,
6152 ["wscr"] = 120012,
6153 ["xscr"] = 120013,
6154 ["yscr"] = 120014,
6155 ["zscr"] = 120015,
6156 ["Afr"] = 120068,
6157 ["Bfr"] = 120069,
6158 ["Dfr"] = 120071,
6159 ["Efr"] = 120072,
6160 ["Ffr"] = 120073,
6161 ["Gfr"] = 120074,
6162 ["Jfr"] = 120077,
6163 ["Kfr"] = 120078,
6164 ["Lfr"] = 120079,
6165 ["Mfr"] = 120080,
6166 ["Nfr"] = 120081,
6167 ["Ofr"] = 120082,
6168 ["Pfr"] = 120083,
6169 ["Qfr"] = 120084,
6170 ["Sfr"] = 120086,
6171 ["Tfr"] = 120087,

6172 ["Ufr"] = 120088,
6173 ["Vfr"] = 120089,
6174 ["Wfr"] = 120090,
6175 ["Xfr"] = 120091,
6176 ["Yfr"] = 120092,
6177 ["afr"] = 120094,
6178 ["bfr"] = 120095,
6179 ["cfr"] = 120096,
6180 ["dfr"] = 120097,
6181 ["efr"] = 120098,
6182 ["ffr"] = 120099,
6183 ["gfr"] = 120100,
6184 ["hfr"] = 120101,
6185 ["ifr"] = 120102,
6186 ["jfr"] = 120103,
6187 ["kfr"] = 120104,
6188 ["lfr"] = 120105,
6189 ["mfr"] = 120106,
6190 ["nfr"] = 120107,
6191 ["ofr"] = 120108,
6192 ["pfr"] = 120109,
6193 ["qfr"] = 120110,
6194 ["rfr"] = 120111,
6195 ["sfr"] = 120112,
6196 ["tfr"] = 120113,
6197 ["ufr"] = 120114,
6198 ["vfr"] = 120115,
6199 ["wfr"] = 120116,
6200 ["xfr"] = 120117,
6201 ["yfr"] = 120118,
6202 ["zfr"] = 120119,
6203 ["Aopf"] = 120120,
6204 ["Bopf"] = 120121,
6205 ["Dopf"] = 120123,
6206 ["Eopf"] = 120124,
6207 ["Fopf"] = 120125,
6208 ["Gopf"] = 120126,
6209 ["Iopf"] = 120128,
6210 ["Jopf"] = 120129,
6211 ["Kopf"] = 120130,
6212 ["Lopf"] = 120131,
6213 ["Mopf"] = 120132,
6214 ["Oopf"] = 120134,
6215 ["Sopf"] = 120138,
6216 ["Topf"] = 120139,
6217 ["Uopf"] = 120140,
6218 ["Vopf"] = 120141,

```

6219 ["Wopf"] = 120142,
6220 ["Xopf"] = 120143,
6221 ["Yopf"] = 120144,
6222 ["aopf"] = 120146,
6223 ["bopf"] = 120147,
6224 ["copf"] = 120148,
6225 ["dopf"] = 120149,
6226 ["eopf"] = 120150,
6227 ["fopf"] = 120151,
6228 ["gopf"] = 120152,
6229 ["hopf"] = 120153,
6230 ["iopf"] = 120154,
6231 ["jopf"] = 120155,
6232 ["kopf"] = 120156,
6233 ["lopf"] = 120157,
6234 ["mopf"] = 120158,
6235 ["nopf"] = 120159,
6236 ["oopf"] = 120160,
6237 ["popf"] = 120161,
6238 ["qopf"] = 120162,
6239 ["ropf"] = 120163,
6240 ["sopf"] = 120164,
6241 ["topf"] = 120165,
6242 ["uopf"] = 120166,
6243 ["vopf"] = 120167,
6244 ["wopf"] = 120168,
6245 ["xopf"] = 120169,
6246 ["yopf"] = 120170,
6247 ["zopf"] = 120171,
6248 }

```

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

```

6249 function entities.dec_entity(s)
6250   local n = tonumber(s)
6251   if n == nil then
6252     return "&#" .. s .. ";" -- fallback for unknown entities
6253   end
6254   return unicode.utf8.char(n)
6255 end

```

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

```

6256 function entities.hex_entity(s)
6257   local n = tonumber("0x"..s)
6258   if n == nil then
6259     return "&#x" .. s .. ";" -- fallback for unknown entities
6260   end

```

```

6261 return unicode.utf8.char(n)
6262 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.

```

6263 function entities.hex_entity_with_x_char(x, s)
6264     local n = tonumber("0x"..s)
6265     if n == nil then
6266         return "&#" .. x .. s .. ";" -- fallback for unknown entities
6267     end
6268     return unicode.utf8.char(n)
6269 end

```

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

```

6270 function entities.char_entity(s)
6271     local code_points = character_entities[s]
6272     if code_points == nil then
6273         return "&" .. s .. ";"
6274     end
6275     if type(code_points) ~= 'table' then
6276         code_points = {code_points}
6277     end
6278     local char_table = {}
6279     for _, code_point in ipairs(code_points) do
6280         table.insert(char_table, unicode.utf8.char(code_point))
6281     end
6282     return table.concat(char_table)
6283 end

```

3.1.3 Plain T_EX Writer

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

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

```

6284 M.writer = {}

```

The `writer.new` method creates and returns a new T_EX writer object associated with the Lua interface options (see Section 2.1.3) `options`. When `options` are unspecified, it is assumed that an empty table was passed to the method.

The objects produced by the `writer.new` method expose instance methods and variables of their own. As a convention, I will refer to these *<member>*s as `writer-><member>`. All member variables are immutable unless explicitly stated otherwise.

```
6285 function M.writer.new(options)
6286   local self = {}
```

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

```
6287   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.

```
6288   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.

```
6289   local slice_specifiers = {}
6290   for specifier in options.slice:gmatch("[^%s]+") do
6291     table.insert(slice_specifiers, specifier)
6292   end
6293
6294   if #slice_specifiers == 2 then
6295     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
6296     local slice_begin_type = self.slice_begin:sub(1, 1)
6297     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
6298       self.slice_begin = "^" .. self.slice_begin
6299     end
6300     local slice_end_type = self.slice_end:sub(1, 1)
6301     if slice_end_type ~= "^" and slice_end_type ~= "$" then
6302       self.slice_end = "$" .. self.slice_end
6303     end
6304   elseif #slice_specifiers == 1 then
6305     self.slice_begin = "^" .. slice_specifiers[1]
6306     self.slice_end = "$" .. slice_specifiers[1]
6307   end
6308
6309   self.slice_begin_type = self.slice_begin:sub(1, 1)
6310   self.slice_begin_identifier = self.slice_begin:sub(2) or ""
6311   self.slice_end_type = self.slice_end:sub(1, 1)
6312   self.slice_end_identifier = self.slice_end:sub(2) or ""
6313
6314   if self.slice_begin == "^" and self.slice_end ~= "^" then
6315     self.is_writing = true
6316   else
6317     self.is_writing = false
6318   end
```


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

```
6319 self.space = " "
```

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

```
6320 self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
6321 function self.plain(s)
6322   return s
6323 end
```

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

```
6324 function self.paragraph(s)
6325   if not self.is_writing then return "" end
6326   return s
6327 end
```

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

```
6328 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{}"
6329 function self.interblocksep()
6330   if not self.is_writing then return "" end
6331   return self.interblocksep_text
6332 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.

```
6333 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{}"
6334 function self.paragraphsep()
6335   if not self.is_writing then return "" end
6336   return self.paragraphsep_text
6337 end
```

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

```
6338 self.undosep_text = "\\markdownRendererUndoSeparator\n{}"
6339 function self.undosep()
6340   if not self.is_writing then return "" end
6341   return self.undosep_text
6342 end
```

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

```
6343 self.soft_line_break = function()
6344   if self.flatten_inlines then return "\n" end
6345   return "\\markdownRendererSoftLineBreak\n{}"
6346 end
```

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

```
6347 self.hard_line_break = function()
6348   if self.flatten_inlines then return "\n" end
6349   return "\\markdownRendererHardLineBreak\n{"
6350 end
```

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

```
6351 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
6352 function self.thematic_break()
6353   if not self.is_writing then return "" end
6354   return "\\markdownRendererThematicBreak{"
6355 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.

```
6356 self.escaped_uri_chars = {
6357   [{""] = "\\markdownRendererLeftBrace{"",
6358   ["}"] = "\\markdownRendererRightBrace{"",
6359   [{"\"}] = "\\markdownRendererBackslash{"",
6360   [{"r"}] = " ",
6361   [{"n"}] = " ",
6362 }
6363 self.escaped_minimal_strings = {
6364   [{"^"}] = "\\markdownRendererCircumflex",
6365   .. "\\markdownRendererCircumflex ",
6366   [{"☒"}] = "\\markdownRendererTickedBox{"",
6367   [{"☐"}] = "\\markdownRendererHalfTickedBox{"",
6368   [{"□"}] = "\\markdownRendererUntickedBox{"",
6369   [entities.hex_entity('FFFD')]
6370   = "\\markdownRendererReplacementCharacter{"",
6371 }
```

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

```
6372 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6373 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.

```
6374 self.escaped_chars = {
6375   [{"{"] = "\\markdownRendererLeftBrace{"",
6376   [{"}"] = "\\markdownRendererRightBrace{"",
6377   [{"%"}] = "\\markdownRendererPercentSign{"",
6378   [{"\"}] = "\\markdownRendererBackslash{"",
```

```

6379     ["#"] = "\\markdownRendererHash{}",
6380     ["$"] = "\\markdownRendererDollarSign{}",
6381     ["&"] = "\\markdownRendererAmpersand{}",
6382     ["_"] = "\\markdownRendererUnderscore{}",
6383     ["^"] = "\\markdownRendererCircumflex{}",
6384     ["~"] = "\\markdownRendererTilde{}",
6385     ["|"] = "\\markdownRendererPipe{}",
6386     [entities.hex_entity('0000')]
6387     = "\\markdownRendererReplacementCharacter{}",
6388 }

```

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

```

6389 local function create_escaper(char_escapes, string_escapes)
6390     local escape = util.escaper(char_escapes, string_escapes)
6391     return function(s)
6392         if self.flatten_inlines then return s end
6393         return escape(s)
6394     end
6395 end
6396 local escape_typographic_text = create_escaper(
6397     self.escaped_chars, self.escaped_strings)
6398 local escape_programmatic_text = create_escaper(
6399     self.escaped_uri_chars, self.escaped_minimal_strings)
6400 local escape_minimal = create_escaper(
6401     {}, 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.

```

6402 self.escape = escape_typographic_text
6403 self.math = escape_minimal
6404 if options.hybrid then
6405     self.identifier = escape_minimal
6406     self.string = escape_minimal
6407     self.uri = escape_minimal
6408     self.infostring = escape_minimal
6409 else

```

```

6410     self.identifier = escape_programmatic_text
6411     self.string = escape_typographic_text
6412     self.uri = escape_programmatic_text
6413     self.infostring = escape_programmatic_text
6414 end

```

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

```

6415 function self.warning(t, m)
6416     return {"\markdownRendererWarning{", self.escape(t), "}{" ,
6417             escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6418             escape_minimal(m or ""), "}"}
6419 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.

```

6420 function self.error(t, m)
6421     return {"\markdownRendererError{", self.escape(t), "}{" ,
6422             escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6423             escape_minimal(m or ""), "}"}
6424 end

```

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

```

6425 function self.code(s, attributes)
6426     if self.flatten_inlines then return s end
6427     local buf = {}
6428     if attributes ~= nil then
6429         table.insert(buf,
6430             "\markdownRendererCodeSpanAttributeContextBegin\n")
6431         table.insert(buf, self.attributes(attributes))
6432     end
6433     table.insert(buf,
6434         {"\markdownRendererCodeSpan{", self.escape(s), "}"}
6435     if attributes ~= nil then
6436         table.insert(buf,
6437             "\markdownRendererCodeSpanAttributeContextEnd{")
6438     end
6439     return buf
6440 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.

```

6441 function self.link(lab, src, tit, attributes)
6442     if self.flatten_inlines then return lab end
6443     local buf = {}
6444     if attributes ~= nil then

```

```

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

```

6459     function self.image(lab, src, tit, attributes)
6460         if self.flatten_inlines then return lab end
6461         local buf = {}
6462         if attributes ~= nil then
6463             table.insert(buf,
6464                 "\\markdownRendererImageAttributeContextBegin\n")
6465             table.insert(buf, self.attributes(attributes))
6466         end
6467         table.insert(buf, {"\\markdownRendererImage{" ,lab,"} ",
6468             "{" ,self.string(src),"} ",
6469             "{" ,self.uri(src),"} ",
6470             "{" ,self.string(tit or ""),"} "})
6471         if attributes ~= nil then
6472             table.insert(buf,
6473                 "\\markdownRendererImageAttributeContextEnd{ }")
6474         end
6475         return buf
6476     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.

```

6477     function self.bulletlist(items,tight)
6478         if not self.is_writing then return "" end
6479         local buffer = {}
6480         for _,item in ipairs(items) do
6481             if item ~= "" then
6482                 buffer[#buffer + 1] = self.bulletitem(item)
6483             end

```

```

6484     end
6485     local contents = util.intersperse(buffer,"\n")
6486     if tight and options.tightLists then
6487         return {"\\markdownRendererUlBeginTight\n",contents,
6488             "\n\\markdownRendererUlEndTight "}
6489     else
6490         return {"\\markdownRendererUlBegin\n",contents,
6491             "\n\\markdownRendererUlEnd "}
6492     end
6493 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.

```

6494 function self.bulletitem(s)
6495     return {"\\markdownRendererUlItem ",s,
6496         "\\markdownRendererUlItemEnd "}
6497 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.

```

6498 function self.orderedlist(items,tight,startnum)
6499     if not self.is_writing then return "" end
6500     local buffer = {}
6501     local num = startnum
6502     for _,item in ipairs(items) do
6503         if item ~= "" then
6504             buffer[#buffer + 1] = self.ordereditem(item,num)
6505         end
6506         if num ~= nil and item ~= "" then
6507             num = num + 1
6508         end
6509     end
6510     local contents = util.intersperse(buffer,"\n")
6511     if tight and options.tightLists then
6512         return {"\\markdownRendererOlBeginTight\n",contents,
6513             "\n\\markdownRendererOlEndTight "}
6514     else
6515         return {"\\markdownRendererOlBegin\n",contents,
6516             "\n\\markdownRendererOlEnd "}
6517     end
6518 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.

```

6519 function self.ordereditem(s,num)

```

```

6520     if num ~= nil then
6521         return {"\\markdownRendererOliItemWithNumber{" ,num,"} ",s,
6522             "\\markdownRendererOliItemEnd "}
6523     else
6524         return {"\\markdownRendererOliItem " ,s,
6525             "\\markdownRendererOliItemEnd "}
6526     end
6527 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.

```

6528     function self.inline_html_comment(contents)
6529         if self.flatten_inlines then return contents end
6530         return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
6531     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.

```

6532     function self.inline_html_tag(contents)
6533         if self.flatten_inlines then return contents end
6534         return {"\\markdownRendererInlineHtmlTag{" ,
6535             self.string(contents),""}
6536     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.

```

6537     function self.block_html_element(s)
6538         if not self.is_writing then return "" end
6539         local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6540         return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
6541     end

```

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

```

6542     function self.emphasis(s)
6543         if self.flatten_inlines then return s end
6544         return {"\\markdownRendererEmphasis{" ,s,"}"}
6545     end

```

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

```

6546     function self.checkbox(f)
6547         if f == 1.0 then
6548             return "☒ "
6549         elseif f == 0.0 then

```

```

6550     return "□ "
6551   else
6552     return "◻ "
6553   end
6554 end

```

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

```

6555 function self.strong(s)
6556   if self.flatten_inlines then return s end
6557   return {"\markdownRendererStrongEmphasis{" ,s,"}"}
6558 end

```

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

```

6559 function self.blockquote(s)
6560   if not self.is_writing then return "" end
6561   return {"\markdownRendererBlockQuoteBegin\n",s,
6562         "\markdownRendererBlockQuoteEnd "}
6563 end

```

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

```

6564 function self.verbatim(s)
6565   if not self.is_writing then return "" end
6566   s = s:gsub("\n$", "")
6567   local name = util.cache_verbatim(options.cacheDir, s)
6568   return {"\markdownRendererInputVerbatim{" ,name,"}"}
6569 end

```

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

```

6570 function self.document(d)
6571   local buf = {"\markdownRendererDocumentBegin\n"}
6572
6573   -- warn against the `hybrid` option
6574   if options.hybrid then
6575     local text = "The `hybrid` option has been soft-deprecated."
6576     local more = "Consider using one of the following better options "
6577     .. "for mixing TeX and markdown: `contentBlocks`, "
6578     .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6579     .. "`texMathSingleBackslash`, and "
6580     .. "`texMathDoubleBackslash`. "
6581     .. "For more information, see the user manual at "
6582     .. "<https://witiko.github.io/markdown/>."
6583     table.insert(buf, self.warning(text, more))
6584   end
6585 end

```



```

6586     -- insert the text of the document
6587     table.insert(buf, d)
6588
6589     -- pop all attributes
6590     table.insert(buf, self.pop_attributes())
6591
6592     table.insert(buf, "\\markdownRendererDocumentEnd")
6593
6594     return buf
6595 end

```

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

```

6596     local seen_identifiers = {}
6597     local key_value_regex = "([^\s= ]+)%s*=%s*(.*)"
6598     local function normalize_attributes(attributes, auto_identifiers)
6599         -- normalize attributes
6600         local normalized_attributes = {}
6601         local has_explicit_identifiers = false
6602         local key, value
6603         for _, attribute in ipairs(attributes or {}) do
6604             if attribute:sub(1, 1) == "#" then
6605                 table.insert(normalized_attributes, attribute)
6606                 has_explicit_identifiers = true
6607                 seen_identifiers[attribute:sub(2)] = true
6608             elseif attribute:sub(1, 1) == "." then
6609                 table.insert(normalized_attributes, attribute)
6610             else
6611                 key, value = attribute:match(key_value_regex)
6612                 if key:lower() == "id" then
6613                     table.insert(normalized_attributes, "#" .. value)
6614                 elseif key:lower() == "class" then
6615                     local classes = {}
6616                     for class in value:gmatch("%S+") do
6617                         table.insert(classes, class)
6618                     end
6619                     table.sort(classes)
6620                     for _, class in ipairs(classes) do
6621                         table.insert(normalized_attributes, "." .. class)
6622                     end
6623                 else
6624                     table.insert(normalized_attributes, attribute)
6625                 end
6626             end
6627         end
6628         -- if no explicit identifiers exist, add auto identifiers

```

```

6630     if not has_explicit_identifiers and auto_identifiers ~= nil then
6631         local seen_auto_identifiers = {}
6632         for _, auto_identifier in ipairs(auto_identifiers) do
6633             if seen_auto_identifiers[auto_identifier] == nil then
6634                 seen_auto_identifiers[auto_identifier] = true
6635                 if seen_identifiers[auto_identifier] == nil then
6636                     seen_identifiers[auto_identifier] = true
6637                     table.insert(normalized_attributes,
6638                                 "#" .. auto_identifier)
6639             else
6640                 local auto_identifier_number = 1
6641                 while true do
6642                     local numbered_auto_identifier = auto_identifier .. "-"
6643                                             .. auto_identifier_number
6644                     if seen_identifiers[numbered_auto_identifier] == nil then
6645                         seen_identifiers[numbered_auto_identifier] = true
6646                         table.insert(normalized_attributes,
6647                                     "#" .. numbered_auto_identifier)
6648                     break
6649                 end
6650                 auto_identifier_number = auto_identifier_number + 1
6651             end
6652         end
6653     end
6654 end
6655 end
6656
6657 -- sort and deduplicate normalized attributes
6658 table.sort(normalized_attributes)
6659 local seen_normalized_attributes = {}
6660 local deduplicated_normalized_attributes = {}
6661 for _, attribute in ipairs(normalized_attributes) do
6662     if seen_normalized_attributes[attribute] == nil then
6663         seen_normalized_attributes[attribute] = true
6664         table.insert(deduplicated_normalized_attributes, attribute)
6665     end
6666 end
6667
6668 return deduplicated_normalized_attributes
6669 end
6670
6671 function self.attributes(attributes, should_normalize_attributes)
6672     local normalized_attributes
6673     if should_normalize_attributes == false then
6674         normalized_attributes = attributes
6675     else
6676         normalized_attributes = normalize_attributes(attributes)

```

```

6677     end
6678
6679     local buf = {}
6680     local key, value
6681     for _, attribute in ipairs(normalized_attributes) do
6682         if attribute:sub(1, 1) == "#" then
6683             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6684                 attribute:sub(2), "}")})
6685         elseif attribute:sub(1, 1) == "." then
6686             table.insert(buf, {"\\markdownRendererAttributeName{" ,
6687                 attribute:sub(2), "}")})
6688         else
6689             key, value = attribute:match(key_value_regex)
6690             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6691                 key, "}{", value, "}")})
6692         end
6693     end
6694
6695     return buf
6696 end

```

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

```

6697 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`.

```

6698 self.attribute_type_levels = {}
6699 setmetatable(self.attribute_type_levels,
6700     { __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`.

```

6701 local function apply_attributes()
6702     local buf = {}
6703     for i = 1, #self.active_attributes do
6704         local start_output = self.active_attributes[i][3]
6705         if start_output ~= nil then
6706             table.insert(buf, start_output)
6707         end
6708     end
6709     return buf
6710 end
6711
6712 local function tear_down_attributes()
6713     local buf = {}
6714     for i = #self.active_attributes, 1, -1 do

```

```

6715     local end_output = self.active_attributes[i][4]
6716     if end_output ~= nil then
6717         table.insert(buf, end_output)
6718     end
6719     end
6720     return buf
6721 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.

```

6722 function self.push_attributes(attribute_type, attributes,
6723                             start_output, end_output)
6724     local attribute_type_level
6725         = self.attribute_type_levels[attribute_type]
6726     self.attribute_type_levels[attribute_type]
6727         = attribute_type_level + 1
6728
6729     -- index attributes in a hash table for easy lookup
6730     attributes = attributes or {}
6731     for i = 1, #attributes do
6732         attributes[attributes[i]] = true
6733     end
6734
6735     local buf = {}
6736     -- handle slicing
6737     if attributes["#" .. self.slice_end_identififier] ~= nil and
6738         self.slice_end_type == "^" then
6739         if self.is_writing then
6740             table.insert(buf, self.undosep())
6741             table.insert(buf, tear_down_attributes())
6742         end
6743         self.is_writing = false
6744     end
6745     if attributes["#" .. self.slice_begin_identififier] ~= nil and
6746         self.slice_begin_type == "^" then
6747         table.insert(buf, apply_attributes())
6748         self.is_writing = true
6749     end
6750     if self.is_writing and start_output ~= nil then
6751         table.insert(buf, start_output)
6752     end
6753     table.insert(self.active_attributes,
6754                 {attribute_type, attributes,
6755                 start_output, end_output})

```

```

6756     return buf
6757 end
6758

```

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`).

```

6759 function self.pop_attributes(attribute_type)
6760     local buf = {}
6761     -- pop attributes until we find attributes of correct type
6762     -- or until no attributes remain
6763     local current_attribute_type = false
6764     while current_attribute_type ~= attribute_type and
6765           #self.active_attributes > 0 do
6766         local attributes, _, end_output
6767         current_attribute_type, attributes, _, end_output = table.unpack(
6768             self.active_attributes[#self.active_attributes])
6769         local attribute_type_level
6770             = self.attribute_type_levels[current_attribute_type]
6771         self.attribute_type_levels[current_attribute_type]
6772             = attribute_type_level - 1
6773         if self.is_writing and end_output ~= nil then
6774             table.insert(buf, end_output)
6775         end
6776         table.remove(self.active_attributes, #self.active_attributes)
6777         -- handle slicing
6778         if attributes["#" .. self.slice_end_identifier] ~= nil
6779             and self.slice_end_type == "$" then
6780             if self.is_writing then
6781                 table.insert(buf, self.undosep())
6782                 table.insert(buf, tear_down_attributes())
6783             end
6784             self.is_writing = false
6785         end
6786         if attributes["#" .. self.slice_begin_identifier] ~= nil and
6787             self.slice_begin_type == "$" then
6788             self.is_writing = true
6789             table.insert(buf, apply_attributes())
6790         end
6791     end
6792     return buf
6793 end

```

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

```

6794 local function create_auto_identifier(s)
6795     local buffer = {}
6796     local prev_space = false
6797     local letter_found = false
6798     local normalized_s = s
6799     if not options.unicodeNormalization
6800         or options.unicodeNormalizationForm ~= "nfc" then
6801         normalized_s = uni_algos.normalize.NFC(normalized_s)
6802     end
6803
6804     for _, code in utf8.codes(normalized_s) do
6805         local char = utf8.char(code)
6806
6807         -- Remove everything up to the first letter.
6808         if not letter_found then
6809             local is_letter = unicode.utf8.match(char, "%a")
6810             if is_letter then
6811                 letter_found = true
6812             else
6813                 goto continue
6814             end
6815         end
6816
6817         -- Remove all non-alphanumeric characters, except underscores,
6818         -- hyphens, and periods.
6819         if not unicode.utf8.match(char, "[%w_-%.%s]") then
6820             goto continue
6821         end
6822
6823         -- Replace all spaces and newlines with hyphens.
6824         if unicode.utf8.match(char, "[%s\n]") then
6825             char = "-"
6826             if prev_space then
6827                 goto continue
6828             else
6829                 prev_space = true
6830             end
6831         else
6832             -- Convert all alphabetic characters to lowercase.
6833             char = unicode.utf8.lower(char)
6834             prev_space = false
6835         end
6836
6837         table.insert(buffer, char)
6838
6839         ::continue::
6840     end

```

```

6841
6842     if prev_space then
6843         table.remove(buffer)
6844     end
6845
6846     local identifier = #buffer == 0 and "section"
6847                     or table.concat(buffer, "")
6848     return identifier
6849 end

```

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

```

6850 local function create_gfm_auto_identifier(s)
6851     local buffer = {}
6852     local prev_space = false
6853     local letter_found = false
6854     local normalized_s = s
6855     if not options.unicodeNormalization
6856         or options.unicodeNormalizationForm ~= "nfc" then
6857         normalized_s = uni_algos.normalize.NFC(normalized_s)
6858     end
6859
6860     for _, code in utf8.codes(normalized_s) do
6861         local char = utf8.char(code)
6862
6863         -- Remove everything up to the first non-space.
6864         if not letter_found then
6865             local is_letter = unicode.utf8.match(char, "%S")
6866             if is_letter then
6867                 letter_found = true
6868             else
6869                 goto continue
6870             end
6871         end
6872
6873         -- Remove all non-alphanumeric characters, except underscores
6874         -- and hyphens.
6875         if not unicode.utf8.match(char, "[%w_-%s]") then
6876             prev_space = false
6877             goto continue
6878         end
6879
6880         -- Replace all spaces and newlines with hyphens.
6881         if unicode.utf8.match(char, "[%s\n]") then
6882             char = "-"
6883             if prev_space then
6884                 goto continue

```

```

6885     else
6886         prev_space = true
6887     end
6888     else
6889         -- Convert all alphabetic characters to lowercase.
6890         char = unicode.utf8.lower(char)
6891         prev_space = false
6892     end
6893
6894     table.insert(buffer, char)
6895
6896     ::continue::
6897 end
6898
6899 if prev_space then
6900     table.remove(buffer)
6901 end
6902
6903 local identifier = #buffer == 0 and "section"
6904                 or table.concat(buffer, "")
6905 return identifier
6906 end

```

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

```

6907 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6908 self.secend_text = "\\n\\markdownRendererSectionEnd "
6909 function self.heading(s, level, attributes)
6910     local buf = {}
6911     local flat_text, inlines = table.unpack(s)
6912
6913     -- push empty attributes for implied sections
6914     while self.attribute_type_levels["heading"] < level - 1 do
6915         table.insert(buf,
6916                     self.push_attributes("heading",
6917                                         nil,
6918                                         self.secbegin_text,
6919                                         self.secend_text))
6920     end
6921
6922     -- pop attributes for sections that have ended
6923     while self.attribute_type_levels["heading"] >= level do
6924         table.insert(buf, self.pop_attributes("heading"))
6925     end
6926
6927     -- construct attributes for the new section
6928     local auto_identifiers = {}

```



```

6929     if self.options.autoIdentifiers then
6930         table.insert(auto_identifiers, create_auto_identifier(flat_text))
6931     end
6932     if self.options.gfmAutoIdentifiers then
6933         table.insert(auto_identifiers,
6934             create_gfm_auto_identifier(flat_text))
6935     end
6936     local normalized_attributes = normalize_attributes(attributes,
6937                                                         auto_identifiers)
6938
6939     -- push attributes for the new section
6940     local start_output = {}
6941     local end_output = {}
6942     table.insert(start_output, self.secbegin_text)
6943     table.insert(end_output, self.secend_text)
6944
6945     table.insert(buf, self.push_attributes("heading",
6946                                           normalized_attributes,
6947                                           start_output,
6948                                           end_output))
6949     assert(self.attribute_type_levels["heading"] == level)
6950
6951     -- render the heading and its attributes
6952     if self.is_writing and #normalized_attributes > 0 then
6953         table.insert(buf,
6954             "\\markdownRendererHeaderAttributeContextBegin\n")
6955         table.insert(buf, self.attributes(normalized_attributes, false))
6956     end
6957
6958     local cmd
6959     level = level + options.shiftHeadings
6960     if level <= 1 then
6961         cmd = "\\markdownRendererHeadingOne"
6962     elseif level == 2 then
6963         cmd = "\\markdownRendererHeadingTwo"
6964     elseif level == 3 then
6965         cmd = "\\markdownRendererHeadingThree"
6966     elseif level == 4 then
6967         cmd = "\\markdownRendererHeadingFour"
6968     elseif level == 5 then
6969         cmd = "\\markdownRendererHeadingFive"
6970     elseif level >= 6 then
6971         cmd = "\\markdownRendererHeadingSix"
6972     else
6973         cmd = ""
6974     end
6975     if self.is_writing then

```

```

6976     table.insert(buf, {cmd, "{", inlines, "}"})
6977 end
6978
6979 if self.is_writing and #normalized_attributes > 0 then
6980     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6981 end
6982
6983 return buf
6984 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.

```

6985 function self.get_state()
6986     return {
6987         is_writing=self.is_writing,
6988         flatten_inlines=self.flatten_inlines,
6989         active_attributes={table.unpack(self.active_attributes)},
6990     }
6991 end

```

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

```

6992 function self.set_state(s)
6993     local previous_state = self.get_state()
6994     for key, value in pairs(s) do
6995         self[key] = value
6996     end
6997     return previous_state
6998 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`.

```

6999 function self.defer_call(f)
7000     local previous_state = self.get_state()
7001     return function(...)
7002         local state = self.set_state(previous_state)
7003         local return_value = f(...)
7004         self.set_state(state)
7005         return return_value
7006     end
7007 end
7008
7009 return self
7010 end

```

3.1.4 Parsers

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

```
7011 local parsers = {}
```

3.1.4.1 Basic Parsers

```
7012 parsers.percent = P("%")
7013 parsers.at = P("@")
7014 parsers.comma = P(",")
7015 parsers.asterisk = P("*")
7016 parsers.dash = P("-")
7017 parsers.plus = P("+")
7018 parsers.underscore = P("_")
7019 parsers.period = P(".")
7020 parsers.hash = P("#")
7021 parsers.dollar = P("$")
7022 parsers.ampersand = P("&")
7023 parsers.backtick = P("`")
7024 parsers.less = P("<")
7025 parsers.more = P(">")
7026 parsers.space = P(" ")
7027 parsers.squote = P("'")
7028 parsers.dquote = P('"')
7029 parsers.lparent = P("(")
7030 parsers.rparent = P(")")
7031 parsers.lbracket = P("[")
7032 parsers.rbracket = P("]")
7033 parsers.lbrace = P("{")
7034 parsers.rbrace = P("}")
7035 parsers.circumflex = P("^")
7036 parsers.slash = P("/")
7037 parsers.equal = P("=")
7038 parsers.colon = P(":")
7039 parsers.semicolon = P(";")
7040 parsers.exclamation = P("!")
7041 parsers.pipe = P("|")
7042 parsers.tilde = P("~")
7043 parsers.backslash = P("\\")
7044 parsers.tab = P("\t")
7045 parsers.newline = P("\n")
7046
7047 parsers.digit = R("09")
7048 parsers.hexdigit = R("09", "af", "AF")
7049 parsers.letter = R("AZ", "az")
7050 parsers.alphanumeric = R("AZ", "az", "09")
```

```

7051 parsers.keyword           = parsers.letter
7052                           * (parsers.alphanumeric + parsers.dash)^0
7053
7054 parsers.doubleasterisks    = P("**")
7055 parsers.doubleunderscores  = P("__")
7056 parsers.doubletildes      = P("~~")
7057 parsers.fourspace         = P("    ")
7058
7059 parsers.any                 = P(1)
7060 parsers.succeed            = P(true)
7061 parsers.fail               = P(false)
7062
7063 parsers.internal_punctuation = S(":,;.?.")
7064 parsers.ascii_punctuation  = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
7065

```

3.1.5 Unicode punctuation

This section documents the Unicode punctuation³³ 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.

```

7066 ;(function()
7067   local pathname = assert(kpse.find_file("UnicodeData.txt"),
7068     [[Could not locate file "UnicodeData.txt"]])
7069   local file = assert(io.open(pathname, "r"),
7070     [[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.

```

7071   local prefix_trees = {}
7072   for line in file:lines() do
7073     local codepoint, major_category = line:match("^(%x+);[^\;]*;(%a)")
7074     if major_category == "P" or major_category == "S" then
7075       local code = unicode.utf8.char(tonumber(codepoint, 16))
7076       if prefix_trees[#code] == nil then
7077         prefix_trees[#code] = {}
7078       end
7079       local node = prefix_trees[#code]

```

³³See <https://spec.commonmark.org/0.31.2/#unicode-punctuation-character>.

```

7080     for i = 1, #code do
7081         local byte = code:sub(i, i)
7082         if i < #code then
7083             if node[byte] == nil then
7084                 node[byte] = {}
7085             end
7086             node = node[byte]
7087         else
7088             table.insert(node, byte)
7089         end
7090     end
7091 end
7092 end
7093 assert(file:close())
7094

```

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

```

7095 local function depth_first_search(node, path, visit, leave)
7096     visit(node, path)
7097     for label, child in pairs(node) do
7098         if type(child) == "table" then
7099             depth_first_search(child, path .. label, visit, leave)
7100         else
7101             visit(child, path)
7102         end
7103     end
7104     leave(node, path)
7105 end
7106
7107 print("M.punctuation = {}")
7108 print("local S = lpeg.S")
7109 print("-- luacheck: push no max line length")
7110 for length, prefix_tree in pairs(prefix_trees) do
7111     local subparsers = {}
7112     depth_first_search(prefix_tree, "", function(node, path)
7113         if type(node) == "string" then
7114             local suffix
7115             if node == "]" then
7116                 suffix = "S('" .. node .. "')"
7117             else
7118                 suffix = "S([[ " .. node .. " ]])"
7119             end
7120             if subparsers[path] ~= nil then
7121                 subparsers[path] = subparsers[path] .. " + " .. suffix
7122             else
7123                 subparsers[path] = suffix
7124             end
7125         end
7126     end)

```

```

7126     end, function(_, path)
7127         if #path > 0 then
7128             local byte = path:sub(#path, #path)
7129             local parent_path = path:sub(1, #path-1)
7130             if subparsers[path] ~= nil then
7131                 local suffix
7132                 if byte == "]" then
7133                     suffix = "S('" .. byte .. "')"
7134                 else
7135                     suffix = "S([[ " .. byte .. " ]])"
7136                 end
7137                 suffix = suffix .. " * (" .. subparsers[path] .. ")"
7138                 if subparsers[parent_path] ~= nil then
7139                     subparsers[parent_path] = subparsers[parent_path]
7140                                             .. " + " .. suffix
7141                 else
7142                     subparsers[parent_path] = suffix
7143                 end
7144             end
7145         else
7146             print("M.punctuation[" .. length .. "] = " .. subparsers[path])
7147         end
7148     end)
7149 end
7150 print("-- luacheck: pop")
7151 end()
7152 print("return M")

```

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

```

7153 local unicode_data = require("markdown-unicode-data")
7154 if metadata.version ~= unicode_data.metadata.version then
7155     util.warning(
7156         "markdown.lua " .. metadata.version .. " used with " ..
7157         "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
7158 )
7159 end
7160 parsers.punctuation = unicode_data.punctuation
7161
7162 parsers.escapable           = parsers.ascii_punctuation
7163 parsers.anyescaped         = parsers.backslash / ""
7164                             * parsers.escapable
7165                             + parsers.any
7166
7167 parsers.spacechar          = S("\t ")
7168 parsers.spacing            = S(" \n\r\t")
7169 parsers.nonspacechar       = parsers.any - parsers.spacing

```

```

7170 parsers.optionalspace      = parsers.spacechar^0
7171
7172 parsers.normalchar          = parsers.any - (V("SpecialChar")
7173                               + parsers.spacing)
7174 parsers.eof                  = -parsers.any
7175 parsers.nonindentspace      = parsers.space^-3 * - parsers.spacechar
7176 parsers.indent              = parsers.space^-3 * parsers.tab
7177                               + parsers.fourspace / ""
7178 parsers.linechar            = P(1 - parsers.newline)
7179
7180 parsers.blankline           = parsers.optionalspace
7181                               * parsers.newline / "\n"
7182 parsers.blanklines          = parsers.blankline^0
7183 parsers.skipblanklines      = ( parsers.optionalspace
7184                               * parsers.newline)^0
7185 parsers.indentedline        = parsers.indent /""
7186                               * C( parsers.linechar^1
7187                                   * parsers.newline^-1)
7188 parsers.optionallyindentedline = parsers.indent^-1 /""
7189                               * C( parsers.linechar^1
7190                                   * parsers.newline^-1)
7191 parsers.sp                   = parsers.spacing^0
7192 parsers.spnl                 = parsers.optionalspace
7193                               * ( parsers.newline
7194                                   * parsers.optionalspace)^-1
7195 parsers.line                 = parsers.linechar^0 * parsers.newline
7196 parsers.nonemptyline        = parsers.line - parsers.blankline

```

3.1.5.1 Parsers Used for Indentation

```

7197
7198 parsers.leader              = parsers.space^-3
7199

```

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

```

7200 local function has_trail(indent_table)
7201   return indent_table ~= nil and
7202     indent_table.trail ~= nil and
7203     next(indent_table.trail) ~= nil
7204 end
7205

```

Check if indent table `indent_table` has any indents.

```

7206 local function has_indents(indent_table)
7207   return indent_table ~= nil and
7208     indent_table.indents ~= nil and
7209     next(indent_table.indents) ~= nil
7210 end

```

7211

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

```
7212 local function add_trail(indent_table, trail_info)
7213     indent_table.trail = trail_info
7214     return indent_table
7215 end
7216
```

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

```
7217 local function remove_trail(indent_table)
7218     indent_table.trail = nil
7219     return indent_table
7220 end
7221
```

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

```
7222 local function update_indent_table(indent_table, new_indent, add)
7223     indent_table = remove_trail(indent_table)
7224
7225     if not has_indents(indent_table) then
7226         indent_table.indents = {}
7227     end
7228
7229
7230     if add then
7231         indent_table.indents[#indent_table.indents + 1] = new_indent
7232     else
7233         if indent_table.indents[#indent_table.indents].name
7234             == new_indent.name then
7235             indent_table.indents[#indent_table.indents] = nil
7236         end
7237     end
7238
7239     return indent_table
7240 end
7241
```

Remove an indent by its name `name`.

```
7242 local function remove_indent(name)
7243     local remove_indent_level =
7244         function(s, i, indent_table) -- luacheck: ignore s i
7245             indent_table = update_indent_table(indent_table, {name=name},
7246                                                 false)
7247             return true, indent_table
7248         end
7249
7250     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
7251 end
```


7252

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`.

```
7253 local function process_starter_spacing(indent, spacing,
7254                                     minimum, left_strip_length)
7255   left_strip_length = left_strip_length or 0
7256
7257   local count = 0
7258   local tab_value = 4 - (indent) % 4
7259
7260   local code_started, minimum_found = false, false
7261   local code_start, minimum_remainder = "", ""
7262
7263   local left_total_stripped = 0
7264   local full_remainder = ""
7265
7266   if spacing ~= nil then
7267     for i = 1, #spacing do
7268       local character = spacing:sub(i, i)
7269
7270       if character == "\t" then
7271         count = count + tab_value
7272         tab_value = 4
7273       elseif character == " " then
7274         count = count + 1
7275         tab_value = 4 - (1 - tab_value) % 4
7276       end
7277
7278       if (left_strip_length ~= 0) then
7279         local possible_to_strip = math.min(count, left_strip_length)
7280         count = count - possible_to_strip
7281         left_strip_length = left_strip_length - possible_to_strip
7282         left_total_stripped = left_total_stripped + possible_to_strip
7283       else
7284         full_remainder = full_remainder .. character
7285       end
7286
7287       if (minimum_found) then
7288         minimum_remainder = minimum_remainder .. character
7289       elseif (count >= minimum) then
7290         minimum_found = true
7291         minimum_remainder = minimum_remainder
7292         .. string.rep(" ", count - minimum)
```

```

7293     end
7294
7295     if (code_started) then
7296         code_start = code_start .. character
7297     elseif (count >= minimum + 4) then
7298         code_started = true
7299         code_start = code_start
7300             .. string.rep(" ", count - (minimum + 4))
7301     end
7302 end
7303 end
7304
7305 local remainder
7306 if (code_started) then
7307     remainder = code_start
7308 else
7309     remainder = string.rep(" ", count - minimum)
7310 end
7311
7312 local is_minimum = count >= minimum
7313 return {
7314     is_code = code_started,
7315     remainder = remainder,
7316     left_total_stripped = left_total_stripped,
7317     is_minimum = is_minimum,
7318     minimum_remainder = minimum_remainder,
7319     total_length = count,
7320     full_remainder = full_remainder
7321 }
7322 end
7323

```

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

```

7324 local function count_indent_tab_level(indent_table)
7325     local count = 0
7326     if not has_indents(indent_table) then
7327         return count
7328     end
7329
7330     for i=1, #indent_table.indents do
7331         count = count + indent_table.indents[i].length
7332     end
7333     return count
7334 end
7335

```

Count the total width of a delimiter `delimiter`.

```

7336 local function total_delimiter_length(delimiter)

```

```

7337 local count = 0
7338 if type(delimiter) == "string" then return #delimiter end
7339 for _, value in pairs(delimiter) do
7340     count = count + total_delimiter_length(value)
7341 end
7342 return count
7343 end
7344

```

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`.

```

7345 local function process_starter_indent(_, _, indent_table, starter,
7346                                     is_blank, indent_type, breakable)
7347     local last_trail = starter[1]
7348     local delimiter = starter[2]
7349     local raw_new_trail = starter[3]
7350
7351     if indent_type == "bq" and not breakable then
7352         indent_table.ignore_blockquote_blank = true
7353     end
7354
7355     if has_trail(indent_table) then
7356         local trail = indent_table.trail
7357         if trail.is_code then
7358             return false
7359         end
7360         last_trail = trail.remainder
7361     else
7362         local sp = process_starter_spacing(0, last_trail, 0, 0)
7363
7364         if sp.is_code then
7365             return false
7366         end
7367         last_trail = sp.remainder
7368     end
7369
7370     local preceding_indentation = count_indent_tab_level(indent_table) % 4
7371     local last_trail_length = #last_trail
7372     local delimiter_length = total_delimiter_length(delimiter)
7373
7374     local total_indent_level = preceding_indentation + last_trail_length
7375                             + delimiter_length
7376
7377     local sp = {}
7378     if not is_blank then
7379         sp = process_starter_spacing(total_indent_level, raw_new_trail,
7380                                   0, 1)

```

```

7381 end
7382
7383 local del_trail_length = sp.left_total_stripped
7384 if is_blank then
7385     del_trail_length = 1
7386 elseif not sp.is_code then
7387     del_trail_length = del_trail_length + #sp.remainder
7388 end
7389
7390 local indent_length = last_trail_length + delimiter_length
7391                       + del_trail_length
7392 local new_indent_info = {name=indent_type, length=indent_length}
7393
7394 indent_table = update_indent_table(indent_table, new_indent_info,
7395                                   true)
7396 indent_table = add_trail(indent_table,
7397                          {is_code=sp.is_code,
7398                           remainder=sp.remainder,
7399                           total_length=sp.total_length,
7400                           full_remainder=sp.full_remainder})
7401
7402 return true, indent_table
7403 end
7404

```

Return the pattern corresponding with the indent name `name`.

```

7405 local function decode_pattern(name)
7406     local delimiter = parsers.succeed
7407     if name == "bq" then
7408         delimiter = parsers.more
7409     end
7410
7411     return C(parsers.optionalspace) * C(delimiter)
7412           * C(parsers.optionalspace) * Cp()
7413 end
7414

```

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

```

7415 local function left_blank_starter(indent_table)
7416     local blank_starter_index
7417
7418     if not has_indents(indent_table) then
7419         return
7420     end
7421
7422     for i = #indent_table.indents,1,-1 do
7423         local value = indent_table.indents[i]

```

```

7424     if value.name == "li" then
7425         blank_starter_index = i
7426     else
7427         break
7428     end
7429 end
7430
7431 return blank_starter_index
7432 end
7433

```

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.

```

7434 local function traverse_indent(s, i, indent_table, is_optional,
7435                               is_blank, current_line_indents)
7436     local new_index = i
7437
7438     local preceding_indentation = 0
7439     local current_trail = {}
7440
7441     local blank_starter = left_blank_starter(indent_table)
7442
7443     if current_line_indents == nil then
7444         current_line_indents = {}
7445     end
7446
7447     for index = 1, #indent_table.indents do
7448         local value = indent_table.indents[index]
7449         local pattern = decode_pattern(value.name)
7450
7451         -- match decoded pattern
7452         local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7453         if new_indent_info == nil then
7454             local blankline_end = lpeg.match(
7455                 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7456             if is_optional or not indent_table.ignore_blockquote_blank
7457                 or not blankline_end then
7458                 return is_optional, new_index, current_trail,
7459                     current_line_indents
7460             end
7461
7462             return traverse_indent(s, tonumber(blankline_end.pos),
7463                                 indent_table, is_optional, is_blank,
7464                                 current_line_indents)

```

```

7465     end
7466
7467     local raw_last_trail = new_indent_info[1]
7468     local delimiter = new_indent_info[2]
7469     local raw_new_trail = new_indent_info[3]
7470     local next_index = new_indent_info[4]
7471
7472     local space_only = delimiter == ""
7473
7474     -- check previous trail
7475     if not space_only and next(current_trail) == nil then
7476         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7477         current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7478                         total_length=sp.total_length,
7479                         full_remainder=sp.full_remainder}
7480     end
7481
7482     if next(current_trail) ~= nil then
7483         if not space_only and current_trail.is_code then
7484             return is_optional, new_index, current_trail,
7485                    current_line_indents
7486         end
7487         if current_trail.internal_remainder ~= nil then
7488             raw_last_trail = current_trail.internal_remainder
7489         end
7490     end
7491
7492     local raw_last_trail_length = 0
7493     local delimiter_length = 0
7494
7495     if not space_only then
7496         delimiter_length = #delimiter
7497         raw_last_trail_length = #raw_last_trail
7498     end
7499
7500     local total_indent_level = preceding_indentation
7501                               + raw_last_trail_length + delimiter_length
7502
7503     local spacing_to_process
7504     local minimum = 0
7505     local left_strip_length = 0
7506
7507     if not space_only then
7508         spacing_to_process = raw_new_trail
7509         left_strip_length = 1
7510     else
7511         spacing_to_process = raw_last_trail

```

```

7512     minimum = value.length
7513 end
7514
7515 local sp = process_starter_spacing(total_indent_level,
7516                                 spacing_to_process, minimum,
7517                                 left_strip_length)
7518
7519 if space_only and not sp.is_minimum then
7520     return is_optional or (is_blank and blank_starter <= index),
7521     new_index, current_trail, current_line_indents
7522 end
7523
7524 local indent_length = raw_last_trail_length + delimiter_length
7525                    + sp.left_total_stripped
7526
7527 -- update info for the next pattern
7528 if not space_only then
7529     preceding_indentation = preceding_indentation + indent_length
7530 else
7531     preceding_indentation = preceding_indentation + value.length
7532 end
7533
7534 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7535                internal_remainder=sp.minimum_remainder,
7536                total_length=sp.total_length,
7537                full_remainder=sp.full_remainder}
7538
7539 current_line_indents[#current_line_indents + 1] = new_indent_info
7540 new_index = next_index
7541 end
7542
7543 return true, new_index, current_trail, current_line_indents
7544 end
7545

```

Check if a code trail is expected.

```

7546 local function check_trail(expect_code, is_code)
7547     return (expect_code and is_code) or (not expect_code and not is_code)
7548 end
7549

```

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`.

```

7550 local check_trail_joined =
7551     function(s, i, indent_table, -- luacheck: ignore s i
7552            spacing, expect_code, omit_remainder)
7553     local is_code

```

```

7554     local remainder
7555
7556     if has_trail(indent_table) then
7557         local trail = indent_table.trail
7558         is_code = trail.is_code
7559         if is_code then
7560             remainder = trail.remainder
7561         else
7562             remainder = trail.full_remainder
7563         end
7564     else
7565         local sp = process_starter_spacing(0, spacing, 0, 0)
7566         is_code = sp.is_code
7567         if is_code then
7568             remainder = sp.remainder
7569         else
7570             remainder = sp.full_remainder
7571         end
7572     end
7573
7574     local result = check_trail(expect_code, is_code)
7575     if omit_remainder then
7576         return result
7577     end
7578     return result, remainder
7579 end
7580

```

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

```

7581 local check_trail_length =
7582 function(s, i, indent_table, -- luacheck: ignore s i
7583         spacing, min, max)
7584     local trail
7585
7586     if has_trail(indent_table) then
7587         trail = indent_table.trail
7588     else
7589         trail = process_starter_spacing(0, spacing, 0, 0)
7590     end
7591
7592     local total_length = trail.total_length
7593     if total_length == nil then
7594         return false
7595     end
7596
7597     return min <= total_length and total_length <= max
7598 end
7599

```


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

```
7600 local function check_continuation_indentation(s, i, indent_table,
7601                                             is_optional, is_blank)
7602   if not has_indents(indent_table) then
7603     return true
7604   end
7605
7606   local passes, new_index, current_trail, current_line_indents =
7607     traverse_indent(s, i, indent_table, is_optional, is_blank)
7608
7609   if passes then
7610     indent_table.current_line_indents = current_line_indents
7611     indent_table = add_trail(indent_table, current_trail)
7612     return new_index, indent_table
7613   end
7614   return false
7615 end
7616
```

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

```
7617 local function get_last_indent_name(indent_table)
7618   if has_indents(indent_table) then
7619     return indent_table.indents[#indent_table.indents].name
7620   end
7621 end
7622
```

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

```
7623 local function remove_remainder_if_blank(indent_table, remainder)
7624   if get_last_indent_name(indent_table) == "li" then
7625     return ""
7626   end
7627   return remainder
7628 end
7629
```

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.

```
7630 local check_trail_type =
7631   function(s, i, -- luacheck: ignore s i
7632           trail, spacing, trail_type)
7633     if trail == nil then
7634       trail = process_starter_spacing(0, spacing, 0, 0)
7635     end
7636
7637     if trail_type == "non-code" then
```

```

7638     return check_trail(false, trail.is_code)
7639 end
7640 if trail_type == "code" then
7641     return check_trail(true, trail.is_code)
7642 end
7643 if trail_type == "full-code" then
7644     if (trail.is_code) then
7645         return i, trail.remainder
7646     end
7647     return i, ""
7648 end
7649 if trail_type == "full-any" then
7650     return i, trail.internal_remainder
7651 end
7652 end
7653

```

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

```

7654 local trail_freezing =
7655     function(s, i, -- luacheck: ignore s i
7656         indent_table, is_freezing)
7657     if is_freezing then
7658         if indent_table.is_trail_frozen then
7659             indent_table.trail = indent_table.frozen_trail
7660         else
7661             indent_table.frozen_trail = indent_table.trail
7662             indent_table.is_trail_frozen = true
7663         end
7664     else
7665         indent_table.frozen_trail = nil
7666         indent_table.is_trail_frozen = false
7667     end
7668     return true, indent_table
7669 end
7670

```

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`.

```

7671 local check_continuation_indentation_and_trail =
7672     function (s, i, indent_table, is_optional, is_blank, trail_type,
7673         reset_rem, omit_remainder)
7674     if not has_indents(indent_table) then
7675         local spacing, new_index = lpeg.match( C(parsers.spacechar^0)
7676             * Cp(), s, i)
7677         local result, remainder = check_trail_type(s, i,
7678             indent_table.trail, spacing, trail_type)
7679         if remainder == nil then

```

```

7680         if result then
7681             return new_index
7682         end
7683         return false
7684     end
7685     if result then
7686         return new_index, remainder
7687     end
7688     return false
7689 end
7690
7691 local passes, new_index, current_trail = traverse_indent(s, i,
7692     indent_table, is_optional, is_blank)
7693
7694 if passes then
7695     local spacing
7696     if current_trail == nil then
7697         local newer_spacing, newer_index = lpeg.match(
7698             C(parsers.spacechar^0) * Cp(), s, i)
7699         current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7700         new_index = newer_index
7701         spacing = newer_spacing
7702     else
7703         spacing = current_trail.remainder
7704     end
7705     local result, remainder = check_trail_type(s, new_index,
7706         current_trail, spacing, trail_type)
7707     if remainder == nil or omit_remainder then
7708         if result then
7709             return new_index
7710         end
7711         return false
7712     end
7713
7714     if is_blank and reset_rem then
7715         remainder = remove_remainder_if_blank(indent_table, remainder)
7716     end
7717     if result then
7718         return new_index, remainder
7719     end
7720     return false
7721 end
7722 return false
7723 end
7724

```

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

```

7725 parsers.check_trail = Cmt( Cb("indent_info") * C(parsers.spacechar^0)

```

```

7726             * Cc(false), check_trail_joined)
7727
7728 parsers.check_trail_no_rem = Cmt( Cb("indent_info")
7729             * C(parsers.spacechar^0) * Cc(false)
7730             * Cc(true), check_trail_joined)
7731
7732 parsers.check_code_trail = Cmt( Cb("indent_info")
7733             * C(parsers.spacechar^0)
7734             * Cc(true), check_trail_joined)
7735
7736 parsers.check_trail_length_range = function(min, max)
7737   return Cmt( Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7738             * Cc(max), check_trail_length)
7739 end
7740
7741 parsers.check_trail_length = function(n)
7742   return parsers.check_trail_length_range(n, n)
7743 end
7744

```

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

```

7745 parsers.freeze_trail = Cg( Cmt(Cb("indent_info")
7746             * Cc(true), trail_freezing), "indent_info")
7747
7748 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7749             trail_freezing), "indent_info")
7750

```

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

```

7751 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7752             check_continuation_indentation)
7753
7754 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7755             check_continuation_indentation)
7756
7757 parsers.check_minimal_blank_indent
7758   = Cmt( Cb("indent_info") * Cc(false)
7759         * Cc(true)
7760         , check_continuation_indentation)
7761

```

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

```

7762
7763 parsers.check_minimal_indent_and_trail =
7764   Cmt( Cb("indent_info")

```

```

7765     * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7766     , check_continuation_indentation_and_trail)
7767
7768 parsers.check_minimal_indent_and_code_trail =
7769     Cmt( Cb("indent_info")
7770         * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7771         , check_continuation_indentation_and_trail)
7772
7773 parsers.check_minimal_blank_indent_and_full_code_trail =
7774     Cmt( Cb("indent_info")
7775         * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7776         , check_continuation_indentation_and_trail)
7777
7778 parsers.check_minimal_indent_and_any_trail =
7779     Cmt( Cb("indent_info")
7780         * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7781         , check_continuation_indentation_and_trail)
7782
7783 parsers.check_minimal_blank_indent_and_any_trail =
7784     Cmt( Cb("indent_info")
7785         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7786         , check_continuation_indentation_and_trail)
7787
7788 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7789     Cmt( Cb("indent_info")
7790         * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7791         , check_continuation_indentation_and_trail)
7792
7793 parsers.check_optional_indent_and_any_trail =
7794     Cmt( Cb("indent_info")
7795         * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7796         , check_continuation_indentation_and_trail)
7797
7798 parsers.check_optional_blank_indent_and_any_trail =
7799     Cmt( Cb("indent_info")
7800         * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7801         , check_continuation_indentation_and_trail)
7802

```

The following patterns specify behaviour around newlines.

```

7803
7804 parsers.spnlc_noexc = parsers.optionalspace
7805                     * ( parsers.newline
7806                       * parsers.check_minimal_indent_and_any_trail)^-1
7807
7808 parsers.spnlc = parsers.optionalspace
7809               * (V("EndlineNoSub"))^-1
7810

```

```

7811 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7812                   + parsers.spacechar^1
7813
7814 parsers.only_blank = parsers.spacechar^0
7815                   * (parsers.newline + parsers.eof)
7816
7817 % \end{macrocode}
7818 % \begin{figure}
7819 % \hspace*{-0.1\textwidth}
7820 % \begin{minipage}{1.2\textwidth}
7821 % \centering
7822 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
7823 % \node[state, initial by diamond, accepting] (noop) {initial};
7824 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
7825 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
7826 % \node[state] (comment) [below=of noop] {comment};
7827 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
7828 % \node[state] (blank_line) [below right=of comment] {blank line};
7829 % \path[->]
7830 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^\wedge$}
7831 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
7832 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
7833 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge$}\drsh$}
7834 % edge [bend left=10] node {match $\drsh$} (leading_spaces)
7835 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$}
7836 % edge [bend right=90] node [right] {match \textbackslash} (odd_back
7837 % edge [bend left=10] node {match \%} (comment)
7838 % edge [bend right=10] node {$\epsilon$} (blank_line)
7839 % edge [bend left=10] node [align=center, right=0.3cm] {match [$^\wedge$}
7840 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
7841 % edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\\
7842 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
7843 % edge [bend right=10] node [align=center, above left=-
7844 % 0.3cm, xshift=0.1cm] {match [$^\wedge$}\textbackslash}\\for \%, capture \textbackslash
7845 % (even_backslash) edge [bend left=10] node {$\epsilon$} (noop);
7846 % \end{tikzpicture}
7847 % \caption{A pushdown automaton that recognizes \TeX{} comments}
7848 % \label{fig:commented_line}
7849 % \end{minipage}
7850 % \end{figure}
7851 % \begin{markdown}
7852 % The \luamdef{parsers.commented_line}^^1` parser recognizes the regular
7853 % language of \TeX{} comments, see an equivalent finite automaton in Figure
7854 % <#fig:commented_line>.
7855 %
7856 % \end{markdown}

```

```

7857 % \begin{macrocode}
7858 parsers.commented_line_letter = parsers.linechar
7859                               + parsers.newline
7860                               - parsers.backslash
7861                               - parsers.percent
7862 parsers.commented_line = Cg(Cc(""), "backslashes")
7863 * ((#(parsers.commented_line_letter
7864   - parsers.newline)
7865  * Cb("backslashes")
7866  * Cs(parsers.commented_line_letter
7867   - parsers.newline)^1 -- initial
7868  * Cg(Cc(""), "backslashes"))
7869 + #( parsers.backslash
7870   * (parsers.backslash + parsers.newline))
7871 * Cg((parsers.backslash -- even backslash
7872   * ( parsers.backslash
7873     + #parsers.newline))^1, "backslashes")
7874 + (parsers.backslash
7875  * (#parsers.percent
7876  * Cb("backslashes")
7877  / function(backslashes)
7878    return string.rep("\\", #backslashes / 2)
7879  end
7880  * C(parsers.percent)
7881  + #parsers.commented_line_letter
7882  * Cb("backslashes")
7883  * Cc("\\")
7884  * C(parsers.commented_line_letter))
7885  * Cg(Cc(""), "backslashes"))^0
7886 * (#parsers.percent
7887 * Cb("backslashes")
7888 / function(backslashes)
7889   return string.rep("\\", #backslashes / 2)
7890 end
7891 * ((parsers.percent -- comment
7892   * parsers.line
7893   * #parsers.blankline) -- blank line
7894 / "\n"
7895 + parsers.percent -- comment
7896 * parsers.line
7897 * parsers.optionalspace) -- leading spaces
7898 + #(parsers.newline)
7899 * Cb("backslashes")
7900 * C(parsers.newline))
7901
7902 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7903   - parsers.blankline)^0

```

```

7904
7905 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7906 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7907 parsers.attribute_key = (parsers.attribute_key_char
7908     - parsers.dash - parsers.digit)
7909     * parsers.attribute_key_char^0
7910 parsers.attribute_value = ( (parsers.dquote / "\"")
7911     * (parsers.anyescaped - parsers.dquote)^0
7912     * (parsers.dquote / "\""))
7913 + ( (parsers.squote / "\"")
7914     * (parsers.anyescaped - parsers.squote)^0
7915     * (parsers.squote / "\""))
7916 + ( parsers.anyescaped
7917     - parsers.dquote
7918     - parsers.rbrace
7919     - parsers.space)^0
7920 parsers.attribute_identfier = parsers.attribute_key_char^1
7921 parsers.attribute_classname = parsers.letter
7922     * parsers.attribute_key_char^0
7923 parsers.attribute_raw = parsers.attribute_raw_char^1
7924
7925 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7926 + C( parsers.hash
7927     * parsers.attribute_identfier)
7928 + C( parsers.period
7929     * parsers.attribute_classname)
7930 + Cs( parsers.attribute_key
7931     * parsers.optionalspace
7932     * parsers.equal
7933     * parsers.optionalspace
7934     * parsers.attribute_value)
7935 parsers.attributes = parsers.lbrace
7936     * parsers.optionalspace
7937     * parsers.attribute
7938     * (parsers.spacechar^1
7939     * parsers.attribute)^0
7940     * parsers.optionalspace
7941     * parsers.rbrace
7942
7943 parsers.raw_attribute = parsers.lbrace
7944     * parsers.optionalspace
7945     * parsers.equal
7946     * C(parsers.attribute_raw)
7947     * parsers.optionalspace
7948     * parsers.rbrace
7949
7950 -- block followed by 0 or more optionally

```



```

7951 -- indented blocks with first line indented.
7952 parsers.indented_blocks = function(bl)
7953   return Cs( bl
7954     * ( parsers.blankline^1
7955       * parsers.indent
7956       * -parsers.blankline
7957       * bl)^0
7958     * (parsers.blankline^1 + parsers.eof) )
7959 end

```

3.1.5.2 Parsers Used for HTML Entities

```

7960 local function repeat_between(pattern, min, max)
7961   return -pattern^(max + 1) * pattern^min
7962 end
7963
7964 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7965                   * C(repeat_between(parsers.hexdigit, 1, 6))
7966                   * parsers.semicolon
7967 parsers.decentity = parsers.ampersand * parsers.hash
7968                   * C(repeat_between(parsers.digit, 1, 7))
7969                   * parsers.semicolon
7970 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7971                   * parsers.semicolon
7972
7973 parsers.html_entities
7974   = parsers.hexentity / entities.hex_entity_with_x_char
7975   + parsers.decentity / entities.dec_entity
7976   + parsers.tagentity / entities.char_entity

```

3.1.5.3 Parsers Used for Markdown Lists

```

7977 parsers.bullet = function(bullet_char, interrupting)
7978   local allowed_end
7979   if interrupting then
7980     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7981   else
7982     allowed_end = C(parsers.spacechar^1)
7983                 + #(parsers.newline + parsers.eof)
7984   end
7985   return parsers.check_trail
7986         * Ct(C(bullet_char) * Cc(""))
7987         * allowed_end
7988 end
7989
7990 local function tickbox(interior)
7991   return parsers.optionalspace * parsers.lbracket
7992         * interior * parsers.rbracket * parsers.spacechar^1

```

```

7993 end
7994
7995 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7996 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7997 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7998

```

3.1.5.4 Parsers Used for Markdown Code Spans

```

7999 parsers.openticks = Cg(parsers.backtick^1, "ticks")
8000
8001 local function captures_equal_length(_,i,a,b)
8002   return #a == #b and i
8003 end
8004
8005 parsers.closeticks = Cmt(C(parsers.backtick^1)
8006   * Cb("ticks"), captures_equal_length)
8007
8008 parsers.intickschar = (parsers.any - S("\n\r`"))
8009   + V("NoSoftLineBreakEndline")
8010   + (parsers.backtick^1 - parsers.closeticks)
8011
8012 local function process_inticks(s)
8013   s = s:gsub("\n", " ")
8014   s = s:gsub("^ (.*) $", "%1")
8015   return s
8016 end
8017
8018 parsers.inticks = parsers.openticks
8019   * C(parsers.space^0)
8020   * parsers.closeticks
8021   + parsers.openticks
8022   * Cs(Cs(parsers.intickschar^0) / process_inticks)
8023   * parsers.closeticks
8024

```

3.1.5.5 Parsers Used for HTML

```

8025 -- case-insensitive match (we assume s is lowercase)
8026 -- must be single byte encoding
8027 parsers.keyword_exact = function(s)
8028   local parser = P(0)
8029   for i=1,#s do
8030     local c = s:sub(i,i)
8031     local m = c .. upper(c)
8032     parser = parser * S(m)
8033   end
8034   return parser

```

```
8035 end
8036
8037 parsers.special_block_keyword =
8038     parsers.keyword_exact("pre") +
8039     parsers.keyword_exact("script") +
8040     parsers.keyword_exact("style") +
8041     parsers.keyword_exact("textarea")
8042
8043 parsers.block_keyword =
8044     parsers.keyword_exact("address") +
8045     parsers.keyword_exact("article") +
8046     parsers.keyword_exact("aside") +
8047     parsers.keyword_exact("base") +
8048     parsers.keyword_exact("basefont") +
8049     parsers.keyword_exact("blockquote") +
8050     parsers.keyword_exact("body") +
8051     parsers.keyword_exact("caption") +
8052     parsers.keyword_exact("center") +
8053     parsers.keyword_exact("col") +
8054     parsers.keyword_exact("colgroup") +
8055     parsers.keyword_exact("dd") +
8056     parsers.keyword_exact("details") +
8057     parsers.keyword_exact("dialog") +
8058     parsers.keyword_exact("dir") +
8059     parsers.keyword_exact("div") +
8060     parsers.keyword_exact("dl") +
8061     parsers.keyword_exact("dt") +
8062     parsers.keyword_exact("fieldset") +
8063     parsers.keyword_exact("figcaption") +
8064     parsers.keyword_exact("figure") +
8065     parsers.keyword_exact("footer") +
8066     parsers.keyword_exact("form") +
8067     parsers.keyword_exact("frame") +
8068     parsers.keyword_exact("frameset") +
8069     parsers.keyword_exact("h1") +
8070     parsers.keyword_exact("h2") +
8071     parsers.keyword_exact("h3") +
8072     parsers.keyword_exact("h4") +
8073     parsers.keyword_exact("h5") +
8074     parsers.keyword_exact("h6") +
8075     parsers.keyword_exact("head") +
8076     parsers.keyword_exact("header") +
8077     parsers.keyword_exact("hr") +
8078     parsers.keyword_exact("html") +
8079     parsers.keyword_exact("iframe") +
8080     parsers.keyword_exact("legend") +
8081     parsers.keyword_exact("li") +
```

```

8082     parsers.keyword_exact("link") +
8083     parsers.keyword_exact("main") +
8084     parsers.keyword_exact("menu") +
8085     parsers.keyword_exact("menuitem") +
8086     parsers.keyword_exact("nav") +
8087     parsers.keyword_exact("noframes") +
8088     parsers.keyword_exact("ol") +
8089     parsers.keyword_exact("optgroup") +
8090     parsers.keyword_exact("option") +
8091     parsers.keyword_exact("p") +
8092     parsers.keyword_exact("param") +
8093     parsers.keyword_exact("section") +
8094     parsers.keyword_exact("source") +
8095     parsers.keyword_exact("summary") +
8096     parsers.keyword_exact("table") +
8097     parsers.keyword_exact("tbody") +
8098     parsers.keyword_exact("td") +
8099     parsers.keyword_exact("tfoot") +
8100     parsers.keyword_exact("th") +
8101     parsers.keyword_exact("thead") +
8102     parsers.keyword_exact("title") +
8103     parsers.keyword_exact("tr") +
8104     parsers.keyword_exact("track") +
8105     parsers.keyword_exact("ul")
8106
8107 -- end conditions
8108 parsers.html_blankline_end_condition
8109 = parsers.linechar^0
8110 * ( parsers.newline
8111     * (parsers.check_minimal_blank_indent_and_any_trail
8112         * #parsers.blankline
8113         + parsers.check_minimal_indent_and_any_trail)
8114     * parsers.linechar^1)^0
8115 * (parsers.newline^-1 / "")
8116
8117 local function remove_trailing_blank_lines(s)
8118     return s:gsub("[\n\r]+%s*$", "")
8119 end
8120
8121 parsers.html_until_end = function(end_marker)
8122     return Cs(Cs((parsers.newline
8123         * (parsers.check_minimal_blank_indent_and_any_trail
8124             * #parsers.blankline
8125             + parsers.check_minimal_indent_and_any_trail)
8126         + parsers.linechar - end_marker)^0
8127         * parsers.linechar^0 * parsers.newline^-1)
8128         / remove_trailing_blank_lines)

```

```

8129 end
8130
8131 -- attributes
8132 parsers.html_attribute_spacing = parsers.optionalspace
8133                               * V("NoSoftLineBreakEndline")
8134                               * parsers.optionalspace
8135                               + parsers.spacechar^1
8136
8137 parsers.html_attribute_name = ( parsers.letter
8138                               + parsers.colon
8139                               + parsers.underscore)
8140                               * ( parsers.alphanumeric
8141                               + parsers.colon
8142                               + parsers.underscore
8143                               + parsers.period
8144                               + parsers.dash)^0
8145
8146 parsers.html_attribute_value = parsers.squote
8147                               * (parsers.linechar - parsers.squote)^0
8148                               * parsers.squote
8149                               + parsers.dquote
8150                               * (parsers.linechar - parsers.dquote)^0
8151                               * parsers.dquote
8152                               + ( parsers.any
8153                               - parsers.spacechar
8154                               - parsers.newline
8155                               - parsers.dquote
8156                               - parsers.squote
8157                               - parsers.backtick
8158                               - parsers.equal
8159                               - parsers.less
8160                               - parsers.more)^1
8161
8162 parsers.html_inline_attribute_value = parsers.squote
8163                                     * (V("NoSoftLineBreakEndline")
8164                                     + parsers.any
8165                                     - parsers.blankline^2
8166                                     - parsers.squote)^0
8167                                     * parsers.squote
8168                                     + parsers.dquote
8169                                     * (V("NoSoftLineBreakEndline")
8170                                     + parsers.any
8171                                     - parsers.blankline^2
8172                                     - parsers.dquote)^0
8173                                     * parsers.dquote
8174                                     + (parsers.any
8175                                     - parsers.spacechar

```

```

8176         - parsers.newline
8177         - parsers.dquote
8178         - parsers.squote
8179         - parsers.backtick
8180         - parsers.equal
8181         - parsers.less
8182         - parsers.more)^1
8183
8184 parsers.html_attribute_value_specification
8185     = parsers.optionalspace
8186     * parsers.equal
8187     * parsers.optionalspace
8188     * parsers.html_attribute_value
8189
8190 parsers.html_spnl = parsers.optionalspace
8191                 * (V("NoSoftLineBreakEndline")
8192                 * parsers.optionalspace)^-1
8193
8194 parsers.html_inline_attribute_value_specification
8195     = parsers.html_spnl
8196     * parsers.equal
8197     * parsers.html_spnl
8198     * parsers.html_inline_attribute_value
8199
8200 parsers.html_attribute
8201     = parsers.html_attribute_spacing
8202     * parsers.html_attribute_name
8203     * parsers.html_inline_attribute_value_specification^-1
8204
8205 parsers.html_non_newline_attribute
8206     = parsers.spacechar^1
8207     * parsers.html_attribute_name
8208     * parsers.html_attribute_value_specification^-1
8209
8210 parsers.nested_breaking_blank = parsers.newline
8211                               * parsers.check_minimal_blank_indent
8212                               * parsers.blankline
8213
8214 parsers.html_comment_start = P("<!--")
8215
8216 parsers.html_comment_end = P("-->")
8217
8218 parsers.html_comment
8219     = Cs( parsers.html_comment_start
8220         * parsers.html_until_end(parsers.html_comment_end))
8221
8222 parsers.html_inline_comment = (parsers.html_comment_start / "")

```

```

8223         * -P(">") * -P("->")
8224         * Cs(( V("NoSoftLineBreakEndline")
8225             + parsers.any
8226             - parsers.nested_breaking_blank
8227             - parsers.html_comment_end)^0)
8228         * (parsers.html_comment_end / "")
8229
8230 parsers.html_cdatasection_start = P("<![CDATA[")
8231
8232 parsers.html_cdatasection_end = P("]]>")
8233
8234 parsers.html_cdatasection
8235     = Cs( parsers.html_cdatasection_start
8236         * parsers.html_until_end(parsers.html_cdatasection_end))
8237
8238 parsers.html_inline_cdatasection
8239     = parsers.html_cdatasection_start
8240     * Cs(V("NoSoftLineBreakEndline") + parsers.any
8241         - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
8242     * parsers.html_cdatasection_end
8243
8244 parsers.html_declaration_start = P("<!") * parsers.letter
8245
8246 parsers.html_declaration_end = P(">")
8247
8248 parsers.html_declaration
8249     = Cs( parsers.html_declaration_start
8250         * parsers.html_until_end(parsers.html_declaration_end))
8251
8252 parsers.html_inline_declaration
8253     = parsers.html_declaration_start
8254     * Cs(V("NoSoftLineBreakEndline") + parsers.any
8255         - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
8256     * parsers.html_declaration_end
8257
8258 parsers.html_instruction_start = P("<?")
8259
8260 parsers.html_instruction_end = P(">")
8261
8262 parsers.html_instruction
8263     = Cs( parsers.html_instruction_start
8264         * parsers.html_until_end(parsers.html_instruction_end))
8265
8266 parsers.html_inline_instruction = parsers.html_instruction_start
8267     * Cs( V("NoSoftLineBreakEndline")
8268         + parsers.any
8269         - parsers.nested_breaking_blank

```

```

8270             - parsers.html_instruction_end)^0
8271             * parsers.html_instruction_end
8272
8273 parsers.html_blankline = parsers.newline
8274             * parsers.optionalspace
8275             * parsers.newline
8276
8277 parsers.html_tag_start = parsers.less
8278
8279 parsers.html_tag_closing_start = parsers.less
8280             * parsers.slash
8281
8282 parsers.html_tag_end = parsers.html_spnl
8283             * parsers.more
8284
8285 parsers.html_empty_tag_end = parsers.html_spnl
8286             * parsers.slash
8287             * parsers.more
8288
8289 -- opening tags
8290 parsers.html_any_open_inline_tag = parsers.html_tag_start
8291             * parsers.keyword
8292             * parsers.html_attribute^0
8293             * parsers.html_tag_end
8294
8295 parsers.html_any_open_tag = parsers.html_tag_start
8296             * parsers.keyword
8297             * parsers.html_non_newline_attribute^0
8298             * parsers.html_tag_end
8299
8300 parsers.html_open_tag = parsers.html_tag_start
8301             * parsers.block_keyword
8302             * parsers.html_attribute^0
8303             * parsers.html_tag_end
8304
8305 parsers.html_open_special_tag = parsers.html_tag_start
8306             * parsers.special_block_keyword
8307             * parsers.html_attribute^0
8308             * parsers.html_tag_end
8309
8310 -- incomplete tags
8311 parsers.incomplete_tag_following = parsers.spacechar
8312             + parsers.more
8313             + parsers.slash * parsers.more
8314             + #(parsers.newline + parsers.eof)
8315
8316 parsers.incomplete_special_tag_following = parsers.spacechar

```



```

8317         + parsers.more
8318         + #( parsers.newline
8319           + parsers.eof)
8320
8321 parsers.html_incomplete_open_tag = parsers.html_tag_start
8322         * parsers.block_keyword
8323         * parsers.incomplete_tag_following
8324
8325 parsers.html_incomplete_open_special_tag
8326 = parsers.html_tag_start
8327 * parsers.special_block_keyword
8328 * parsers.incomplete_special_tag_following
8329
8330 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
8331         * parsers.block_keyword
8332         * parsers.incomplete_tag_following
8333
8334 parsers.html_incomplete_close_special_tag
8335 = parsers.html_tag_closing_start
8336 * parsers.special_block_keyword
8337 * parsers.incomplete_tag_following
8338
8339 -- closing tags
8340 parsers.html_close_tag = parsers.html_tag_closing_start
8341         * parsers.block_keyword
8342         * parsers.html_tag_end
8343
8344 parsers.html_any_close_tag = parsers.html_tag_closing_start
8345         * parsers.keyword
8346         * parsers.html_tag_end
8347
8348 parsers.html_close_special_tag = parsers.html_tag_closing_start
8349         * parsers.special_block_keyword
8350         * parsers.html_tag_end
8351
8352 -- empty tags
8353 parsers.html_any_empty_inline_tag = parsers.html_tag_start
8354         * parsers.keyword
8355         * parsers.html_attribute^0
8356         * parsers.html_empty_tag_end
8357
8358 parsers.html_any_empty_tag = parsers.html_tag_start
8359         * parsers.keyword
8360         * parsers.html_non_newline_attribute^0
8361         * parsers.optionalspace
8362         * parsers.slash
8363         * parsers.more

```

```

8364
8365 parsers.html_empty_tag = parsers.html_tag_start
8366                        * parsers.block_keyword
8367                        * parsers.html_attribute^0
8368                        * parsers.html_empty_tag_end
8369
8370 parsers.html_empty_special_tag = parsers.html_tag_start
8371                                * parsers.special_block_keyword
8372                                * parsers.html_attribute^0
8373                                * parsers.html_empty_tag_end
8374
8375 parsers.html_incomplete_blocks
8376 = parsers.html_incomplete_open_tag
8377 + parsers.html_incomplete_open_special_tag
8378 + parsers.html_incomplete_close_tag
8379
8380 -- parse special html blocks
8381 parsers.html_blankline_ending_special_block_opening
8382 = ( parsers.html_close_special_tag
8383     + parsers.html_empty_special_tag)
8384 * #( parsers.optionalspace
8385     * (parsers.newline + parsers.eof))
8386
8387 parsers.html_blankline_ending_special_block
8388 = parsers.html_blankline_ending_special_block_opening
8389 * parsers.html_blankline_end_condition
8390
8391 parsers.html_special_block_opening
8392 = parsers.html_incomplete_open_special_tag
8393 - parsers.html_empty_special_tag
8394
8395 parsers.html_closing_special_block
8396 = parsers.html_special_block_opening
8397 * parsers.html_until_end(parsers.html_close_special_tag)
8398
8399 parsers.html_special_block
8400 = parsers.html_blankline_ending_special_block
8401 + parsers.html_closing_special_block
8402
8403 -- parse html blocks
8404 parsers.html_block_opening = parsers.html_incomplete_open_tag
8405                             + parsers.html_incomplete_close_tag
8406
8407 parsers.html_block = parsers.html_block_opening
8408                    * parsers.html_blankline_end_condition
8409
8410 -- parse any html blocks

```

```

8411 parsers.html_any_block_opening
8412     = ( parsers.html_any_open_tag
8413         + parsers.html_any_close_tag
8414         + parsers.html_any_empty_tag)
8415     * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8416
8417 parsers.html_any_block     = parsers.html_any_block_opening
8418                             * parsers.html_blankline_end_condition
8419
8420 parsers.html_inline_comment_full = parsers.html_comment_start
8421                                     * -P(">") * -P("->")
8422                                     * Cs(( V("NoSoftLineBreakEndline")
8423                                         + parsers.any - P("--")
8424                                         - parsers.nested_breaking_blank
8425                                         - parsers.html_comment_end)^0)
8426                                     * parsers.html_comment_end
8427
8428 parsers.html_inline_tags     = parsers.html_inline_comment_full
8429                             + parsers.html_any_empty_inline_tag
8430                             + parsers.html_inline_instruction
8431                             + parsers.html_inline_cdatasection
8432                             + parsers.html_inline_declaration
8433                             + parsers.html_any_open_inline_tag
8434                             + parsers.html_any_close_tag
8435

```

3.1.5.6 Parsers Used for Markdown Tags and Links

```

8436 parsers.urlchar = parsers.anyescaped
8437                 - parsers.newline
8438                 - parsers.more
8439
8440 parsers.auto_link_scheme_part = parsers.alphanumeric
8441                               + parsers.plus
8442                               + parsers.period
8443                               + parsers.dash
8444
8445 parsers.auto_link_scheme     = parsers.letter
8446                               * parsers.auto_link_scheme_part
8447                               * parsers.auto_link_scheme_part^-30
8448
8449 parsers.absolute_uri         = parsers.auto_link_scheme * parsers.colon
8450                               * ( parsers.any - parsers.spacing
8451                                   - parsers.less - parsers.more)^0
8452
8453 parsers.printable_characters = S(" !#$%&'*/=?^_`{|}~-")
8454

```

```

8455 parsers.email_address_local_part_char = parsers.alphanumeric
8456                                         + parsers.printable_characters
8457
8458 parsers.email_address_local_part
8459   = parsers.email_address_local_part_char^1
8460
8461 parsers.email_address_dns_label = parsers.alphanumeric
8462                                 * ( parsers.alphanumeric
8463                                   + parsers.dash)^-62
8464                                 * B(parsers.alphanumeric)
8465
8466 parsers.email_address_domain   = parsers.email_address_dns_label
8467                                 * ( parsers.period
8468                                   * parsers.email_address_dns_label)^0
8469
8470 parsers.email_address = parsers.email_address_local_part
8471                       * parsers.at
8472                       * parsers.email_address_domain
8473
8474 parsers.auto_link_url = parsers.less
8475                       * C(parsers.absolute_uri)
8476                       * parsers.more
8477
8478 parsers.auto_link_email = parsers.less
8479                         * C(parsers.email_address)
8480                         * parsers.more
8481
8482 parsers.auto_link_relative_reference = parsers.less
8483                                     * C(parsers.urlchar^1)
8484                                     * parsers.more
8485
8486 parsers.autolink = parsers.auto_link_url
8487                 + parsers.auto_link_email
8488
8489 -- content in balanced brackets, parentheses, or quotes:
8490 parsers.bracketed = P{ parsers.lbracket
8491                       * (( parsers.backslash / "\"" * parsers.rbracket
8492                           + parsers.any - (parsers.lbracket
8493                                           + parsers.rbracket
8494                                           + parsers.blankline^2)
8495                           ) + V(1))^0
8496                       * parsers.rbracket }
8497
8498 parsers.inparens = P{ parsers.lparent
8499                       * ((parsers.anyescaped - (parsers.lparent
8500                                               + parsers.rparent
8501                                               + parsers.blankline^2)

```

```

8502         ) + V(1))^0
8503         * parsers.rparent }
8504
8505 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8506         * ((parsers.anyescaped - (parsers.squote
8507         + parsers.blankline^2)
8508         ) + V(1))^0
8509         * parsers.squote }
8510
8511 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8512         * ((parsers.anyescaped - (parsers.dquote
8513         + parsers.blankline^2)
8514         ) + V(1))^0
8515         * parsers.dquote }
8516
8517 parsers.link_text = parsers.lbracket
8518         * Cs((parsers.alphanumeric^1
8519         + parsers.bracketed
8520         + parsers.inticks
8521         + parsers.autolink
8522         + V("InlineHtml")
8523         + ( parsers.backslash * parsers.backslash)
8524         + ( parsers.backslash
8525         * ( parsers.lbracket
8526         + parsers.rbracket)
8527         + V("NoSoftLineBreakSpace")
8528         + V("NoSoftLineBreakEndline")
8529         + (parsers.any
8530         - ( parsers.newline
8531         + parsers.lbracket
8532         + parsers.rbracket
8533         + parsers.blankline^2))))^0)
8534         * parsers.rbracket
8535
8536 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
8537         * #( ( parsers.any
8538         - parsers.rbracket)^-999
8539         * parsers.rbracket)
8540         * Cs((parsers.alphanumeric^1
8541         + parsers.inticks
8542         + parsers.autolink
8543         + V("InlineHtml")
8544         + ( parsers.backslash * parsers.backslash)
8545         + ( parsers.backslash
8546         * ( parsers.lbracket
8547         + parsers.rbracket)
8548         + V("NoSoftLineBreakSpace")

```

```

8549         + V("NoSoftLineBreakEndline")
8550     + (parsers.any
8551         - ( parsers.newline
8552           + parsers.lbracket
8553           + parsers.rbracket
8554           + parsers.blankline^2))))^1)
8555
8556 parsers.link_label = parsers.lbracket
8557                   * parsers.link_label_body
8558                   * parsers.rbracket
8559
8560 parsers.inparens_url = P{ parsers.lparent
8561                       * ((parsers.anyescaped - (parsers.lparent
8562                                                         + parsers.rparent
8563                                                         + parsers.spacing)
8564                          ) + V(1))^0
8565                       * parsers.rparent }
8566
8567 -- url for markdown links, allowing nested brackets:
8568 parsers.url         = parsers.less * Cs((parsers.anyescaped
8569                                         - parsers.newline
8570                                         - parsers.less
8571                                         - parsers.more)^0)
8572                   * parsers.more
8573                   + -parsers.less
8574                   * Cs((parsers.inparens_url + (parsers.anyescaped
8575                                                         - parsers.spacing
8576                                                         - parsers.lparent
8577                                                         - parsers.rparent))^1)
8578
8579 -- quoted text:
8580 parsers.title_s     = parsers.squote
8581                   * Cs((parsers.html_entities
8582                         + V("NoSoftLineBreakSpace")
8583                         + V("NoSoftLineBreakEndline")
8584                         + ( parsers.anyescaped
8585                           - parsers.newline
8586                           - parsers.squote
8587                           - parsers.blankline^2))^0)
8588                   * parsers.squote
8589
8590 parsers.title_d     = parsers.dquote
8591                   * Cs((parsers.html_entities
8592                         + V("NoSoftLineBreakSpace")
8593                         + V("NoSoftLineBreakEndline")
8594                         + ( parsers.anyescaped
8595                           - parsers.newline

```

```

8596         - parsers.dquote
8597         - parsers.blankline^2))^0)
8598     * parsers.dquote
8599
8600 parsers.title_p    = parsers.lparent
8601     * Cs((parsers.html_entities
8602         + V("NoSoftLineBreakSpace")
8603         + V("NoSoftLineBreakEndline")
8604         + ( parsers.anyescaped
8605         - parsers.newline
8606         - parsers.lparent
8607         - parsers.rparent
8608         - parsers.blankline^2))^0)
8609     * parsers.rparent
8610
8611 parsers.title
8612     = parsers.title_d + parsers.title_s + parsers.title_p
8613
8614 parsers.optionaltitle
8615     = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8616

```

3.1.5.7 Helpers for Links and Link Reference Definitions

```

8617 -- parse a reference definition: [foo]: /bar "title"
8618 parsers.define_reference_parser = (parsers.check_trail / "")
8619     * parsers.link_label * parsers.colon
8620     * parsers.spnlc * parsers.url
8621     * ( parsers.spnlc_sep * parsers.title
8622     * parsers.only_blank
8623     + Cc("") * parsers.only_blank)

```

3.1.5.8 Inline Elements

```

8624 parsers.Inline      = V("Inline")
8625
8626 -- parse many p between starter and ender
8627 parsers.between = function(p, starter, ender)
8628     local ender2 = B(parsers.nonspacechar) * ender
8629     return ( starter
8630         * #parsers.nonspacechar
8631         * Ct(p * (p - ender2)^0)
8632         * ender2)
8633 end
8634

```

3.1.5.9 Block Elements

```

8635 parsers.lineof = function(c)
8636     return ( parsers.check_trail_no_rem
8637             * (P(c) * parsers.optionalspace)^3
8638             * (parsers.newline + parsers.eof))
8639 end
8640
8641 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8642                               + parsers.lineof(parsers.dash)
8643                               + parsers.lineof(parsers.underscore)

```

3.1.5.10 Headings

```

8644 -- parse Atx heading start and return level
8645 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8646                       * -parsers.hash / length
8647
8648 -- parse setext header ending and return level
8649 parsers.heading_level
8650   = parsers.nonindentospace * parsers.equal^1
8651   * parsers.optionalspace * #parsers.newline * Cc(1)
8652   + parsers.nonindentospace * parsers.dash^1
8653   * parsers.optionalspace * #parsers.newline * Cc(2)
8654
8655 local function strip_atx_end(s)
8656   return s:gsub("%s+#+%s*\n$", "")
8657 end
8658
8659 parsers.atx_heading = parsers.check_trail_no_rem
8660                   * Cg(parsers.heading_start, "level")
8661                   * (C( parsers.optionalspace
8662                       * parsers.hash^0
8663                       * parsers.optionalspace
8664                       * parsers.newline)
8665                   + parsers.spacechar^1
8666                   * 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>`.

```
8667 M.reader = {}
8668 function M.reader.new(writer, options)
8669   local self = {}
```

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

```
8670   self.writer = writer
8671   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.

```
8672   self.parsers = {}
8673   (function(parsers)
8674     setmetatable(self.parsers, {
8675       __index = function (_, key)
8676         return parsers[key]
8677       end
8678     })
8679   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.

```
8680   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.

```
8681   function self.normalize_tag(tag)
8682     tag = util.rope_to_string(tag)
8683     tag = tag:gsub("[ \n\r\t]+", " ")
8684     tag = tag:gsub("^ ", ""):gsub(" $", "")
8685     tag = uni_algos.case.casefold(tag, true, false)
8686     return tag
8687   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.

```
8688   local function iterlines(s, f)
8689     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8690     return util.rope_to_string(rope)
8691   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.

```
8692  if options.preserveTabs then
8693      self.expandtabs = function(s) return s end
8694  else
8695      self.expandtabs = function(s)
8696                          if s:find("\t") then
8697                              return iterlines(s, util.expand_tabs_in_line)
8698                          else
8699                              return s
8700                          end
8701                      end
8702  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.

```
8703  self.parser_functions = {}
8704  self.create_parser = function(name, grammar, toplevel)
8705      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.

```
8706      if toplevel and options.stripIndent then
8707          local min_prefix_length, min_prefix = nil, ''
8708          str = iterlines(str, function(line)
8709              if lpeg.match(parsers.nonemptyline, line) == nil then
8710                  return line
8711              end
8712              line = util.expand_tabs_in_line(line)
8713              local prefix = lpeg.match(C(parsers.optionalspace), line)
8714              local prefix_length = #prefix
8715              local is_shorter = min_prefix_length == nil
8716              if not is_shorter then
8717                  is_shorter = prefix_length < min_prefix_length
8718              end
8719              if is_shorter then
8720                  min_prefix_length, min_prefix = prefix_length, prefix
8721              end
8722              return line
```

```

8723         end)
8724         str = str:gsub('^' .. min_prefix, '')
8725     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.

```

8726     if toplevel and (options.texComments or options.hybrid) then
8727         str = lpeg.match(Ct(parsers.commented_line^1), str)
8728         str = util.rope_to_string(str)
8729     end
8730     local res = lpeg.match(grammar(), str)
8731     if res == nil then
8732         return writer.error(
8733             format("Parser `%s` failed to process the input text.", name),
8734             format("Here are the first 20 characters of the remaining "
8735                 .. "unprocessed text: `%s`.", str:sub(1,20))
8736         )
8737     else
8738         return res
8739     end
8740 end
8741 end
8742
8743 self.create_parser("parse_blocks",
8744     function()
8745         return parsers.blocks
8746     end, true)
8747
8748 self.create_parser("parse_blocks_nested",
8749     function()
8750         return parsers.blocks_nested
8751     end, false)
8752
8753 self.create_parser("parse_inlines",
8754     function()
8755         return parsers.inlines
8756     end, false)
8757
8758 self.create_parser("parse_inlines_no_inline_note",
8759     function()
8760         return parsers.inlines_no_inline_note
8761     end, false)
8762
8763 self.create_parser("parse_inlines_no_html",
8764     function()
8765         return parsers.inlines_no_html

```

```

8766             end, false)
8767
8768 self.create_parser("parse_inlines_nbsp",
8769                 function()
8770                     return parsers.inlines_nbsp
8771                 end, false)
8772 self.create_parser("parse_inlines_no_link_or_emphasis",
8773                 function()
8774                     return parsers.inlines_no_link_or_emphasis
8775                 end, false)

```

3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8776 parsers.minimally_indented_blankline
8777     = parsers.check_minimal_indent * (parsers.blankline / "")
8778
8779 parsers.minimally_indented_block
8780     = parsers.check_minimal_indent * V("Block")
8781
8782 parsers.minimally_indented_block_or_paragraph
8783     = parsers.check_minimal_indent * V("BlockOrParagraph")
8784
8785 parsers.minimally_indented_paragraph
8786     = parsers.check_minimal_indent * V("Paragraph")
8787
8788 parsers.minimally_indented_plain
8789     = parsers.check_minimal_indent * V("Plain")
8790
8791 parsers.minimally_indented_par_or_plain
8792     = parsers.minimally_indented_paragraph
8793     + parsers.minimally_indented_plain
8794
8795 parsers.minimally_indented_par_or_plain_no_blank
8796     = parsers.minimally_indented_par_or_plain
8797     - parsers.minimally_indented_blankline
8798
8799 parsers.minimally_indented_ref
8800     = parsers.check_minimal_indent * V("Reference")
8801
8802 parsers.minimally_indented_blank
8803     = parsers.check_minimal_indent * V("Blank")
8804
8805 parsers.conditionally_indented_blankline
8806     = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8807
8808 parsers.minimally_indented_ref_or_block

```

```

8809     = parsers.minimally_indented_ref
8810     + parsers.minimally_indented_block
8811     - parsers.minimally_indented_blankline
8812
8813 parsers.minimally_indented_ref_or_block_or_par
8814     = parsers.minimally_indented_ref
8815     + parsers.minimally_indented_block_or_paragraph
8816     - parsers.minimally_indented_blankline
8817

```

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

```

8818
8819 function parsers.separator_loop(separated_block, paragraph,
8820                                block_separator, paragraph_separator)
8821     return separated_block
8822         + block_separator
8823         * paragraph
8824         * separated_block
8825         + paragraph_separator
8826         * paragraph
8827 end
8828
8829 function parsers.create_loop_body_pair(separated_block, paragraph,
8830                                       block_separator,
8831                                       paragraph_separator)
8832     return {
8833         block = parsers.separator_loop(separated_block, paragraph,
8834                                       block_separator, block_separator),
8835         par = parsers.separator_loop(separated_block, paragraph,
8836                                     block_separator, paragraph_separator)
8837     }
8838 end
8839
8840 parsers.block_sep_group = function(blank)
8841     return blank^0 * parsers.eof
8842         + ( blank^2 / writer.paragraphsep
8843           + blank^0 / writer.interblocksep
8844           )
8845 end
8846
8847 parsers.par_sep_group = function(blank)
8848     return blank^0 * parsers.eof
8849         + blank^0 / writer.paragraphsep
8850 end
8851
8852 parsers.sep_group_no_output = function(blank)

```

```

8853     return blank^0 * parsers.eof
8854         + blank^0
8855 end
8856
8857 parsers.content_blank = parsers.minimally_indented_blankline
8858
8859 parsers.ref_or_block_separated
8860     = parsers.sep_group_no_output(parsers.content_blank)
8861     * ( parsers.minimally_indented_ref
8862         - parsers.content_blank)
8863     + parsers.block_sep_group(parsers.content_blank)
8864     * ( parsers.minimally_indented_block
8865         - parsers.content_blank)
8866
8867 parsers.loop_body_pair =
8868     parsers.create_loop_body_pair(
8869         parsers.ref_or_block_separated,
8870         parsers.minimally_indented_par_or_plain_no_blank,
8871         parsers.block_sep_group(parsers.content_blank),
8872         parsers.par_sep_group(parsers.content_blank))
8873
8874 parsers.content_loop = ( V("Block")
8875                         * parsers.loop_body_pair.block^0
8876                         + (V("Paragraph") + V("Plain")))
8877                         * parsers.ref_or_block_separated
8878                         * parsers.loop_body_pair.block^0
8879                         + (V("Paragraph") + V("Plain")))
8880                         * parsers.loop_body_pair.par^0)
8881                         * parsers.content_blank^0
8882
8883 parsers.indented_content = function()
8884     return Ct( (V("Reference") + (parsers.blankline / ""))
8885               * parsers.content_blank^0
8886               * parsers.check_minimal_indent
8887               * parsers.content_loop
8888               + (V("Reference") + (parsers.blankline / ""))
8889               * parsers.content_blank^0
8890               + parsers.content_loop)
8891 end
8892
8893 parsers.add_indent = function(pattern, name, breakable)
8894     return Cg(Cmt( Cb("indent_info")
8895                  * Ct(pattern)
8896                  * ( #parsers.linechar -- check if starter is blank
8897                    * Cc(false) + Cc(true))
8898                  * Cc(name)
8899                  * Cc(breakable),

```

```

8900             process_starter_indent), "indent_info")
8901     end
8902

```

3.1.6.4 Parsers Used for Markdown Lists (local)

```

8903     if options.hashEnumerators then
8904         parsers.dig = parsers.digit + parsers.hash
8905     else
8906         parsers.dig = parsers.digit
8907     end
8908
8909     parsers.enumerator = function(delimiter_type, interrupting)
8910         local delimiter_range
8911         local allowed_end
8912         if interrupting then
8913             delimiter_range = P("1")
8914             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8915         else
8916             delimiter_range = parsers.dig * parsers.dig^-8
8917             allowed_end = C(parsers.spacechar^1)
8918                 + #(parsers.newline + parsers.eof)
8919         end
8920
8921         return parsers.check_trail
8922             * Ct(C(delimiter_range) * C(delimiter_type))
8923             * allowed_end
8924     end
8925
8926     parsers.starter = parsers.bullet(parsers.dash)
8927         + parsers.bullet(parsers.asterisk)
8928         + parsers.bullet(parsers.plus)
8929         + parsers.enumerator(parsers.period)
8930         + parsers.enumerator(parsers.rparent)
8931

```

3.1.6.5 Parsers Used for Blockquotes (local)

```

8932     parsers.blockquote_start
8933         = parsers.check_trail
8934         * C(parsers.more)
8935         * C(parsers.spacechar^0)
8936
8937     parsers.blockquote_body
8938         = parsers.add_indent(parsers.blockquote_start, "bq", true)
8939         * parsers.indented_content()
8940         * remove_indent("bq")
8941

```

```

8942 if not options.breakableBlockquotes then
8943     parsers.blockquote_body
8944     = parsers.add_indent(parsers.blockquote_start, "bq", false)
8945     * parsers.indented_content()
8946     * remove_indent("bq")
8947 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.

```

8948 local function parse_content_part(content_part)
8949     local rope = util.rope_to_string(content_part)
8950     local parsed
8951     = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8952     parsed.indent_info = nil
8953     return parsed
8954 end
8955

```

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

```

8956 local collect_emphasis_content =
8957     function(t, opening_index, closing_index)
8958         local content = {}
8959
8960         local content_part = {}
8961         for i = opening_index, closing_index do
8962             local value = t[i]
8963
8964             if value.rendered ~= nil then
8965                 content[#content + 1] = parse_content_part(content_part)
8966                 content_part = {}
8967                 content[#content + 1] = value.rendered
8968                 value.rendered = nil
8969             else
8970                 if value.type == "delimiter"
8971                     and value.element == "emphasis" then
8972                     if value.is_active then
8973                         content_part[#content_part + 1]
8974                             = string.rep(value.character, value.current_count)
8975                     end
8976                 else
8977                     content_part[#content_part + 1] = value.content
8978                 end
8979                 value.content = ''
8980                 value.is_active = false
8981             end

```



```

8982     end
8983
8984     if next(content_part) ~= nil then
8985         content[#content + 1] = parse_content_part(content_part)
8986     end
8987
8988     return content
8989 end
8990

```

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

```

8991 local function fill_emph(t, opening_index, closing_index)
8992     local content
8993         = collect_emphasis_content(t, opening_index + 1,
8994                                   closing_index - 1)
8995     t[opening_index + 1].is_active = true
8996     t[opening_index + 1].rendered = writer.emphasis(content)
8997 end
8998

```

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

```

8999 local function fill_strong(t, opening_index, closing_index)
9000     local content
9001         = collect_emphasis_content(t, opening_index + 1,
9002                                   closing_index - 1)
9003     t[opening_index + 1].is_active = true
9004     t[opening_index + 1].rendered = writer.strong(content)
9005 end
9006

```

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

```

9007 local function breaks_three_rule(opening_delimiter, closing_delimiter)
9008     return ( opening_delimiter.is_closing
9009             or closing_delimiter.is_opening)
9010         and (( opening_delimiter.original_count
9011                + closing_delimiter.original_count) % 3 == 0)
9012         and ( opening_delimiter.original_count % 3 ~= 0
9013             or closing_delimiter.original_count % 3 ~= 0)
9014 end
9015

```

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`.

```

9016 local find_emphasis_opener = function(t, bottom_index, latest_index,

```

```

9017                                     character, closing_delimiter)
9018   for i = latest_index, bottom_index, -1 do
9019     local value = t[i]
9020     if value.is_active and
9021        value.is_opening and
9022        value.type == "delimiter" and
9023        value.element == "emphasis" and
9024        (value.character == character) and
9025        (value.current_count > 0) then
9026       if not breaks_three_rule(value, closing_delimiter) then
9027         return i
9028       end
9029     end
9030   end
9031 end
9032

```

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

```

9033 local function process_emphasis(t, opening_index, closing_index)
9034   for i = opening_index, closing_index do
9035     local value = t[i]
9036     if value.type == "delimiter" and value.element == "emphasis" then
9037       local delimiter_length = string.len(value.content)
9038       value.character = string.sub(value.content, 1, 1)
9039       value.current_count = delimiter_length
9040       value.original_count = delimiter_length
9041     end
9042   end
9043
9044   local openers_bottom = {
9045     ['*'] = {
9046       [true] = {opening_index, opening_index, opening_index},
9047       [false] = {opening_index, opening_index, opening_index}
9048     },
9049     ['_'] = {
9050       [true] = {opening_index, opening_index, opening_index},
9051       [false] = {opening_index, opening_index, opening_index}
9052     }
9053   }
9054
9055   local current_position = opening_index
9056   local max_position = closing_index
9057
9058   while current_position <= max_position do
9059     local value = t[current_position]
9060

```

```

9061     if value.type ~= "delimiter" or
9062        value.element ~= "emphasis" or
9063        not value.is_active or
9064        not value.is_closing or
9065        (value.current_count <= 0) then
9066        current_position = current_position + 1
9067        goto continue
9068    end
9069
9070    local character = value.character
9071    local is_opening = value.is_opening
9072    local closing_length_modulo_three = value.original_count % 3
9073
9074    local current_openers_bottom
9075        = openers_bottom[character][is_opening]
9076            [closing_length_modulo_three + 1]
9077
9078    local opener_position
9079        = find_emphasis_opener(t, current_openers_bottom,
9080            current_position - 1, character, value)
9081
9082    if (opener_position == nil) then
9083        openers_bottom[character][is_opening]
9084            [closing_length_modulo_three + 1]
9085            = current_position
9086        current_position = current_position + 1
9087        goto continue
9088    end
9089
9090    local opening_delimiter = t[opener_position]
9091
9092    local current_opening_count = opening_delimiter.current_count
9093    local current_closing_count = t[current_position].current_count
9094
9095    if (current_opening_count >= 2)
9096        and (current_closing_count >= 2) then
9097        opening_delimiter.current_count = current_opening_count - 2
9098        t[current_position].current_count = current_closing_count - 2
9099        fill_strong(t, opener_position, current_position)
9100    else
9101        opening_delimiter.current_count = current_opening_count - 1
9102        t[current_position].current_count = current_closing_count - 1
9103        fill_emph(t, opener_position, current_position)
9104    end
9105
9106    ::continue::
9107 end

```

```

9108 end
9109
9110 local cont = lpeg.R("\128\191") -- continuation byte
9111

```

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

```

9112 local function utf8_by_byte_count(n)
9113   if (n == 1) then
9114     return lpeg.R("\0\127")
9115   end
9116   if (n == 2) then
9117     return lpeg.R("\194\223") * cont
9118   end
9119   if (n == 3) then
9120     return lpeg.R("\224\239") * cont * cont
9121   end
9122   if (n == 4) then
9123     return lpeg.R("\240\244") * cont * cont * cont
9124   end
9125 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`.

```

9126 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
9127   local c
9128   local char_length
9129   for pos = start_pos, end_pos, 1 do
9130     if (start_pos < 0) then
9131       char_length = -pos
9132     else
9133       char_length = pos + 1
9134     end
9135
9136     if (chartype == "punctuation") then
9137       if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
9138         return i
9139       end
9140     else
9141       c = lpeg.match({ C(utf8_by_byte_count(char_length)) }, s, i+pos)
9142       if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
9143         return i
9144       end
9145     end
9146   end
9147 end
9148
9149 local function check_preceding_unicode_punctuation(s, i)
9150   return check_unicode_type(s, i, -4, -1, "punctuation")

```

```

9151 end
9152
9153 local function check_preceding_unicode_whitespace(s, i)
9154     return check_unicode_type(s, i, -4, -1, "%s")
9155 end
9156
9157 local function check_following_unicode_punctuation(s, i)
9158     return check_unicode_type(s, i, 0, 3, "punctuation")
9159 end
9160
9161 local function check_following_unicode_whitespace(s, i)
9162     return check_unicode_type(s, i, 0, 3, "%s")
9163 end
9164
9165 parsers.unicode_preceding_punctuation
9166     = B(parsers.escapable)
9167     + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
9168
9169 parsers.unicode_preceding_whitespace
9170     = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
9171
9172 parsers.unicode_following_punctuation
9173     = #parsers.escapable
9174     + Cmt(parsers.succeed, check_following_unicode_punctuation)
9175
9176 parsers.unicode_following_whitespace
9177     = Cmt(parsers.succeed, check_following_unicode_whitespace)
9178
9179 parsers.delimiter_run = function(character)
9180     return (B(parsers.backslash * character) + -B(character))
9181           * character~1
9182           * -#character
9183 end
9184
9185 parsers.left_flanking_delimiter_run = function(character)
9186     return (B( parsers.any)
9187           * ( parsers.unicode_preceding_punctuation
9188             + parsers.unicode_preceding_whitespace)
9189           + -B(parsers.any))
9190           * parsers.delimiter_run(character)
9191           * parsers.unicode_following_punctuation
9192           + parsers.delimiter_run(character)
9193           * -( parsers.unicode_following_punctuation
9194             + parsers.unicode_following_whitespace
9195             + parsers.eof)
9196 end
9197

```

```

9198 parsers.right_flanking_delimiter_run = function(character)
9199     return parsers.unicode_preceding_punctuation
9200         * parsers.delimiter_run(character)
9201         * ( parsers.unicode_following_punctuation
9202           + parsers.unicode_following_whitespace
9203           + parsers.eof)
9204         + (B(parsers.any)
9205           * -( parsers.unicode_preceding_punctuation
9206              + parsers.unicode_preceding_whitespace))
9207         * parsers.delimiter_run(character)
9208     end
9209
9210     if options.underscores then
9211         parsers.emph_start
9212             = parsers.left_flanking_delimiter_run(parsers.asterisk)
9213             + ( -#parsers.right_flanking_delimiter_run(parsers.underscore)
9214               + ( parsers.unicode_preceding_punctuation
9215                 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
9216             * parsers.left_flanking_delimiter_run(parsers.underscore)
9217
9218         parsers.emph_end
9219             = parsers.right_flanking_delimiter_run(parsers.asterisk)
9220             + ( -#parsers.left_flanking_delimiter_run(parsers.underscore)
9221               + #( parsers.left_flanking_delimiter_run(parsers.underscore)
9222                 * parsers.unicode_following_punctuation))
9223             * parsers.right_flanking_delimiter_run(parsers.underscore)
9224     else
9225         parsers.emph_start
9226             = parsers.left_flanking_delimiter_run(parsers.asterisk)
9227
9228         parsers.emph_end
9229             = parsers.right_flanking_delimiter_run(parsers.asterisk)
9230     end
9231
9232     parsers.emph_capturing_open_and_close
9233         = #parsers.emph_start * #parsers.emph_end
9234         * Ct( Cg(Cc("delimiter"), "type")
9235             * Cg(Cc("emphasis"), "element")
9236             * Cg(C(parsers.emph_start), "content")
9237             * Cg(Cc(true), "is_opening")
9238             * Cg(Cc(true), "is_closing"))
9239
9240     parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
9241                                     * Cg(Cc("emphasis"), "element")
9242                                     * Cg(C(parsers.emph_start), "content")
9243                                     * Cg(Cc(true), "is_opening")
9244                                     * Cg(Cc(false), "is_closing"))

```

```

9245
9246 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
9247                               * Cg(Cc("emphasis"), "element")
9248                               * Cg(C(parsers.emph_end), "content")
9249                               * Cg(Cc(false), "is_opening")
9250                               * Cg(Cc(true), "is_closing"))
9251
9252 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
9253                               + parsers.emph_capturing_open
9254                               + parsers.emph_capturing_close
9255
9256 parsers.emph_open = parsers.emph_capturing_open_and_close
9257                   + parsers.emph_capturing_open
9258
9259 parsers.emph_close = parsers.emph_capturing_open_and_close
9260                   + parsers.emph_capturing_close
9261

```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

9262 -- List of references defined in the document
9263 local references
9264
9265 -- List of note references defined in the document
9266 parsers.rawnotes = {}
9267

```

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.

```

9268 function self.register_link(_, tag, url, title,
9269                             attributes)
9270   local normalized_tag = self.normalize_tag(tag)
9271   if references[normalized_tag] == nil then
9272     references[normalized_tag] = {
9273       url = url,
9274       title = title,
9275       attributes = attributes
9276     }
9277   end
9278   return ""
9279 end
9280

```

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

```

9281 function self.lookup_reference(tag)
9282   return references[self.normalize_tag(tag)]
9283 end

```

9284

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

```
9285 function self.lookup_note_reference(tag)
9286     return parsers.rawnotes[self.normalize_tag(tag)]
9287 end
9288
9289 parsers.title_s_direct_ref = parsers.squote
9290                             * Cs((parsers.html_entities
9291                                   + ( parsers.anyescaped
9292                                       - parsers.squote
9293                                       - parsers.blankline^2))^0)
9294                             * parsers.squote
9295
9296 parsers.title_d_direct_ref = parsers.dquote
9297                             * Cs((parsers.html_entities
9298                                   + ( parsers.anyescaped
9299                                       - parsers.dquote
9300                                       - parsers.blankline^2))^0)
9301                             * parsers.dquote
9302
9303 parsers.title_p_direct_ref = parsers.lparent
9304                             * Cs((parsers.html_entities
9305                                   + ( parsers.anyescaped
9306                                       - parsers.lparent
9307                                       - parsers.rparent
9308                                       - parsers.blankline^2))^0)
9309                             * parsers.rparent
9310
9311 parsers.title_direct_ref = parsers.title_s_direct_ref
9312                          + parsers.title_d_direct_ref
9313                          + parsers.title_p_direct_ref
9314
9315 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
9316                                   * Cg(parsers.url + Cc(""), "url")
9317                                   * parsers.spnl
9318                                   * Cg( parsers.title_direct_ref
9319                                       + Cc(""), "title")
9320                                   * parsers.spnl * parsers.rparent
9321
9322 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
9323                          * Cg(parsers.url + Cc(""), "url")
9324                          * parsers.spnlc
9325                          * Cg(parsers.title + Cc(""), "title")
9326                          * parsers.spnlc * parsers.rparent
9327
```



```

9328 parsers.empty_link = parsers.lbracket
9329                       * parsers.rbracket
9330
9331 parsers.inline_link = parsers.link_text
9332                       * parsers.inline_direct_ref
9333
9334 parsers.full_link = parsers.link_text
9335                       * parsers.link_label
9336
9337 parsers.shortcut_link = parsers.link_label
9338                       * -(parsers.empty_link + parsers.link_label)
9339
9340 parsers.collapsed_link = parsers.link_label
9341                       * parsers.empty_link
9342
9343 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
9344                       * Cg(Cc("inline"), "link_type")
9345                       + #(parsers.exclamation * parsers.full_link)
9346                       * Cg(Cc("full"), "link_type")
9347                       + #( parsers.exclamation
9348                           * parsers.collapsed_link)
9349                       * Cg(Cc("collapsed"), "link_type")
9350                       + #(parsers.exclamation * parsers.shortcut_link)
9351                       * Cg(Cc("shortcut"), "link_type")
9352                       + #(parsers.exclamation * parsers.empty_link)
9353                       * Cg(Cc("empty"), "link_type")
9354
9355 parsers.link_opening = #parsers.inline_link
9356                       * Cg(Cc("inline"), "link_type")
9357                       + #parsers.full_link
9358                       * Cg(Cc("full"), "link_type")
9359                       + #parsers.collapsed_link
9360                       * Cg(Cc("collapsed"), "link_type")
9361                       + #parsers.shortcut_link
9362                       * Cg(Cc("shortcut"), "link_type")
9363                       + #parsers.empty_link
9364                       * Cg(Cc("empty_link"), "link_type")
9365                       + #parsers.link_text
9366                       * Cg(Cc("link_text"), "link_type")
9367
9368 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9369                       * Cg(Cc("note_inline"), "link_type")
9370
9371 parsers.raw_note_opening = #( parsers.lbracket
9372                               * parsers.circumflex
9373                               * parsers.link_label_body
9374                               * parsers.rbracket)

```

```

9375             * Cg(Cc("raw_note"), "link_type")
9376
9377 local inline_note_element = Cg(Cc("note"), "element")
9378             * parsers.note_opening
9379             * Cg( parsers.circumflex
9380                 * parsers.lbracket, "content")
9381
9382 local image_element = Cg(Cc("image"), "element")
9383             * parsers.image_opening
9384             * Cg( parsers.exclamation
9385                 * parsers.lbracket, "content")
9386
9387 local note_element = Cg(Cc("note"), "element")
9388             * parsers.raw_note_opening
9389             * Cg( parsers.lbracket
9390                 * parsers.circumflex, "content")
9391
9392 local link_element = Cg(Cc("link"), "element")
9393             * parsers.link_opening
9394             * Cg(parsers.lbracket, "content")
9395
9396 local opening_elements = parsers.fail
9397
9398 if options.inlineNotes then
9399     opening_elements = opening_elements + inline_note_element
9400 end
9401
9402 opening_elements = opening_elements + image_element
9403
9404 if options.notes then
9405     opening_elements = opening_elements + note_element
9406 end
9407
9408 opening_elements = opening_elements + link_element
9409
9410 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
9411                                * Cg(Cc(true), "is_opening")
9412                                * Cg(Cc(false), "is_closing")
9413                                * opening_elements)
9414
9415 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
9416                                * Cg(Cc("link"), "element")
9417                                * Cg(Cc(false), "is_opening")
9418                                * Cg(Cc(true), "is_closing")
9419                                * ( Cg(Cc(true), "is_direct")
9420                                * Cg( parsers.rbracket
9421                                    * #parsers.inline_direct_ref,

```

```

9422         "content")
9423         + Cg(Cc(false), "is_direct")
9424         * Cg(parsers.rbracket, "content")))
9425
9426 parsers.link_image_open_or_close = parsers.link_image_opening
9427         + parsers.link_image_closing
9428
9429 if options.html then
9430     parsers.link_emph_precedence = parsers.inticks
9431         + parsers.autolink
9432         + parsers.html_inline_tags
9433 else
9434     parsers.link_emph_precedence = parsers.inticks
9435         + parsers.autolink
9436 end
9437
9438 parsers.link_and_emph_endline = parsers.newline
9439         * ((parsers.check_minimal_indent
9440         * -V("EndlineExceptions")
9441         + parsers.check_optional_indent
9442         * -V("EndlineExceptions")
9443         * -V("ListStarter"))) / "")
9444         * parsers.spacechar^0 / "\n"
9445
9446 parsers.link_and_emph_content
9447     = Ct( Cg(Cc("content"), "type")
9448         * Cg(Cs(( parsers.link_emph_precedence
9449             + parsers.backslash * parsers.linechar
9450             + parsers.link_and_emph_endline
9451             + (parsers.linechar
9452             - parsers.blankline^2
9453             - parsers.link_image_open_or_close
9454             - parsers.emph_open_or_close))^0), "content"))
9455
9456 parsers.link_and_emph_table
9457     = (parsers.link_image_opening + parsers.emph_open)
9458     * parsers.link_and_emph_content
9459     * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9460     * parsers.link_and_emph_content)^1
9461

```

Collect the content between the [opening_index](#) and [closing_index](#) in the delimiter table `t`.

```

9462 local function collect_link_content(t, opening_index, closing_index)
9463     local content = {}
9464     for i = opening_index, closing_index do
9465         content[#content + 1] = t[i].content

```

```

9466     end
9467     return util.rope_to_string(content)
9468 end
9469

```

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

```

9470 local function find_link_opener(t, bottom_index, latest_index)
9471     for i = latest_index, bottom_index, -1 do
9472         local value = t[i]
9473         if value.type == "delimiter" and
9474             value.is_opening and
9475             ( value.element == "link"
9476             or value.element == "image"
9477             or value.element == "note")
9478             and not value.removed then
9479             if value.is_active then
9480                 return i
9481             end
9482             value.removed = true
9483             return nil
9484         end
9485     end
9486 end
9487

```

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

```

9488 local function find_next_link_closing_index(t, latest_index)
9489     for i = latest_index, #t do
9490         local value = t[i]
9491         if value.is_closing and
9492             value.element == "link" and
9493             not value.removed then
9494             return i
9495         end
9496     end
9497 end
9498

```

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.

```

9499 local function disable_previous_link_openers(t, opening_index)
9500     if t[opening_index].element == "image" then
9501         return
9502     end
9503
9504     for i = opening_index, 1, -1 do

```

```

9505     local value = t[i]
9506     if value.is_active and
9507         value.type == "delimiter" and
9508         value.is_opening and
9509         value.element == "link" then
9510         value.is_active = false
9511     end
9512 end
9513 end
9514

```

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

```

9515 local function disable_range(t, opening_index, closing_index)
9516     for i = opening_index, closing_index do
9517         local value = t[i]
9518         if value.is_active then
9519             value.is_active = false
9520             if value.type == "delimiter" then
9521                 value.removed = true
9522             end
9523         end
9524     end
9525 end
9526

```

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

```

9527 local delete_parsed_content_in_range =
9528     function(t, opening_index, closing_index)
9529         for i = opening_index, closing_index do
9530             t[i].rendered = nil
9531         end
9532     end
9533

```

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

```

9534 local function empty_content_in_range(t, opening_index, closing_index)
9535     for i = opening_index, closing_index do
9536         t[i].content = ''
9537     end
9538 end
9539

```

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

```

9540 local function join_attributes(reference_attributes, own_attributes)

```

```

9541     local merged_attributes = {}
9542     for _, attribute in ipairs(reference_attributes or {}) do
9543         table.insert(merged_attributes, attribute)
9544     end
9545     for _, attribute in ipairs(own_attributes or {}) do
9546         table.insert(merged_attributes, attribute)
9547     end
9548     if next(merged_attributes) == nil then
9549         merged_attributes = nil
9550     end
9551     return merged_attributes
9552 end
9553

```

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

```

9554     local render_link_or_image =
9555         function(t, opening_index, closing_index, content_end_index,
9556             reference)
9557             process_emphasis(t, opening_index, content_end_index)
9558             local mapped = collect_emphasis_content(t, opening_index + 1,
9559                 content_end_index - 1)
9560
9561             local rendered = {}
9562             if (t[opening_index].element == "link") then
9563                 rendered = writer.link(mapped, reference.url,
9564                     reference.title, reference.attributes)
9565             end
9566
9567             if (t[opening_index].element == "image") then
9568                 rendered = writer.image(mapped, reference.url, reference.title,
9569                     reference.attributes)
9570             end
9571
9572             if (t[opening_index].element == "note") then
9573                 if (t[opening_index].link_type == "note_inline") then
9574                     rendered = writer.note(mapped)
9575                 end
9576                 if (t[opening_index].link_type == "raw_note") then
9577                     rendered = writer.note(reference)
9578                 end
9579             end
9580
9581             t[opening_index].rendered = rendered
9582             delete_parsed_content_in_range(t, opening_index + 1,
9583                 closing_index)
9584             empty_content_in_range(t, opening_index, closing_index)

```

```

9585     disable_previous_link_openers(t, opening_index)
9586     disable_range(t, opening_index, closing_index)
9587 end
9588

```

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.

```

9589 local resolve_inline_following_content =
9590   function(t, closing_index, match_reference, match_link_attributes)
9591     local content = ""
9592     for i = closing_index + 1, #t do
9593       content = content .. t[i].content
9594     end
9595
9596     local matching_content = parsers.succeed
9597
9598     if match_reference then
9599       matching_content = matching_content
9600         * parsers.inline_direct_ref_inside
9601     end
9602
9603     if match_link_attributes then
9604       matching_content = matching_content
9605         * Cg(Ct(parsers.attributes^-1), "attributes")
9606     end
9607
9608     local matched = lpeg.match(Ct( matching_content
9609       * Cg(Cp(), "end_position")), content)
9610
9611     local matched_count = matched.end_position - 1
9612     for i = closing_index + 1, #t do
9613       local value = t[i]
9614
9615       local chars_left = matched_count
9616       matched_count = matched_count - #value.content
9617
9618       if matched_count <= 0 then
9619         value.content = value.content:sub(chars_left + 1)
9620         break
9621       end
9622
9623       value.content = ''
9624       value.is_active = false
9625     end
9626
9627     local attributes = matched.attributes

```

```

9628     if attributes == nil or next(attributes) == nil then
9629         attributes = nil
9630     end
9631
9632     return {
9633         url = matched.url or "",
9634         title = matched.title or "",
9635         attributes = attributes
9636     }
9637 end
9638

```

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.

```

9639 local function resolve_inline_link(t, opening_index, closing_index)
9640     local inline_content
9641     = resolve_inline_following_content(t, closing_index, true,
9642                                       t.match_link_attributes)
9643     render_link_or_image(t, opening_index, closing_index,
9644                         closing_index, inline_content)
9645 end
9646

```

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

```

9647 local resolve_note_inline_link =
9648     function(t, opening_index, closing_index)
9649         local inline_content
9650         = resolve_inline_following_content(t, closing_index,
9651                                           false, false)
9652         render_link_or_image(t, opening_index, closing_index,
9653                             closing_index, inline_content)
9654     end
9655

```

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.

```

9656 local function resolve_shortcut_link(t, opening_index, closing_index)
9657     local content
9658     = collect_link_content(t, opening_index + 1, closing_index - 1)
9659     local r = self.lookup_reference(content)
9660
9661     if r then
9662         local inline_content
9663         = resolve_inline_following_content(t, closing_index, false,
9664                                           t.match_link_attributes)

```



```

9665     r.attributes
9666     = join_attributes(r.attributes, inline_content.attributes)
9667     render_link_or_image(t, opening_index, closing_index,
9668                         closing_index, r)
9669   end
9670 end
9671

```

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.

```

9672 local function resolve_raw_note_link(t, opening_index, closing_index)
9673   local content
9674     = collect_link_content(t, opening_index + 1, closing_index - 1)
9675   local r = self.lookup_note_reference(content)
9676
9677   if r then
9678     local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9679     render_link_or_image(t, opening_index, closing_index,
9680                         closing_index, parsed_ref)
9681   end
9682 end
9683

```

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.

```

9684 local function resolve_full_link(t, opening_index, closing_index)
9685   local next_link_closing_index
9686     = find_next_link_closing_index(t, closing_index + 4)
9687   local next_link_content
9688     = collect_link_content(t, closing_index + 3,
9689                           next_link_closing_index - 1)
9690   local r = self.lookup_reference(next_link_content)
9691
9692   if r then
9693     local inline_content
9694       = resolve_inline_following_content(t, next_link_closing_index,
9695                                         false,
9696                                         t.match_link_attributes)
9697     r.attributes
9698       = join_attributes(r.attributes, inline_content.attributes)
9699     render_link_or_image(t, opening_index, next_link_closing_index,
9700                         closing_index, r)
9701   end
9702 end
9703

```

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.

```

9704 local function resolve_collapsed_link(t, opening_index, closing_index)
9705     local next_link_closing_index
9706         = find_next_link_closing_index(t, closing_index + 4)
9707     local content
9708         = collect_link_content(t, opening_index + 1, closing_index - 1)
9709     local r = self.lookup_reference(content)
9710
9711     if r then
9712         local inline_content
9713             = resolve_inline_following_content(t, closing_index, false,
9714                                               t.match_link_attributes)
9715         r.attributes
9716             = join_attributes(r.attributes, inline_content.attributes)
9717         render_link_or_image(t, opening_index, next_link_closing_index,
9718                             closing_index, r)
9719     end
9720 end
9721

```

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.

```

9722 local function process_links_and_emphasis(t)
9723     for _,value in ipairs(t) do
9724         value.is_active = true
9725     end
9726
9727     for i,value in ipairs(t) do
9728         if not value.is_closing
9729             or value.type ~= "delimiter"
9730             or not ( value.element == "link"
9731                     or value.element == "image"
9732                     or value.element == "note")
9733             or value.removed then
9734             goto continue
9735         end
9736
9737         local opener_position = find_link_opener(t, 1, i - 1)
9738         if (opener_position == nil) then
9739             goto continue
9740         end
9741
9742         local opening_delimiter = t[opener_position]

```

```

9743     opening_delimiter.removed = true
9744
9745     local link_type = opening_delimiter.link_type
9746
9747     if (link_type == "inline") then
9748         resolve_inline_link(t, opener_position, i)
9749     end
9750     if (link_type == "shortcut") then
9751         resolve_shortcut_link(t, opener_position, i)
9752     end
9753     if (link_type == "full") then
9754         resolve_full_link(t, opener_position, i)
9755     end
9756     if (link_type == "collapsed") then
9757         resolve_collapsed_link(t, opener_position, i)
9758     end
9759     if (link_type == "note_inline") then
9760         resolve_note_inline_link(t, opener_position, i)
9761     end
9762     if (link_type == "raw_note") then
9763         resolve_raw_note_link(t, opener_position, i)
9764     end
9765
9766     ::continue::
9767 end
9768
9769 t[#t].content = t[#t].content:gsub("%s*$", "")
9770
9771 process_emphasis(t, 1, #t)
9772 local final_result = collect_emphasis_content(t, 1, #t)
9773 return final_result
9774 end
9775
9776 function self.defer_link_and_emphasis_processing(delimiter_table)
9777     return writer.defer_call(function()
9778         return process_links_and_emphasis(delimiter_table)
9779     end)
9780 end
9781

```

3.1.6.8 Inline Elements (local)

```

9782 parsers.Str      = ( parsers.normalchar
9783                    * (parsers.normalchar + parsers.at)^0)
9784                  / writer.string
9785
9786 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))

```

```

9787             / writer.string
9788
9789 parsers.Ellipsis = P("...") / writer.ellipsis
9790
9791 parsers.Smart     = parsers.Ellipsis
9792
9793 parsers.Code      = parsers.inticks / writer.code
9794
9795 if options.blankBeforeBlockquote then
9796   parsers.bqstart = parsers.fail
9797 else
9798   parsers.bqstart = parsers.blockquote_start
9799 end
9800
9801 if options.blankBeforeHeading then
9802   parsers.headerstart = parsers.fail
9803 else
9804   parsers.headerstart = parsers.atx_heading
9805 end
9806
9807 if options.blankBeforeList then
9808   parsers.interrupting_bullets = parsers.fail
9809   parsers.interrupting_enumerators = parsers.fail
9810 else
9811   parsers.interrupting_bullets
9812     = parsers.bullet(parsers.dash, true)
9813     + parsers.bullet(parsers.asterisk, true)
9814     + parsers.bullet(parsers.plus, true)
9815
9816   parsers.interrupting_enumerators
9817     = parsers.enumerator(parsers.period, true)
9818     + parsers.enumerator(parsers.rparent, true)
9819 end
9820
9821 if options.html then
9822   parsers.html_interrupting
9823     = parsers.check_trail
9824     * ( parsers.html_incomplete_open_tag
9825         + parsers.html_incomplete_close_tag
9826         + parsers.html_incomplete_open_special_tag
9827         + parsers.html_comment_start
9828         + parsers.html_cdatasection_start
9829         + parsers.html_declaration_start
9830         + parsers.html_instruction_start
9831         - parsers.html_close_special_tag
9832         - parsers.html_empty_special_tag)
9833 else

```

```

9834     parsers.html_interrupting = parsers.fail
9835 end
9836
9837 parsers.ListStarter = parsers.starter
9838
9839 parsers.EndlineExceptions
9840     = parsers.blankline -- paragraph break
9841     + parsers.eof      -- end of document
9842     + parsers.bqstart
9843     + parsers.thematic_break_lines
9844     + parsers.interrupting_bullets
9845     + parsers.interrupting_enumerators
9846     + parsers.headerstart
9847     + parsers.html_interrupting
9848
9849 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9850
9851 parsers.endline = parsers.newline
9852     * (parsers.check_minimal_indent
9853     * -V("EndlineExceptions")
9854     + parsers.check_optional_indent
9855     * -V("EndlineExceptions")
9856     * -V("ListStarter")) / function(_) return end
9857     * parsers.spacechar^0
9858
9859 parsers.Endline = parsers.endline
9860     / writer.soft_line_break
9861
9862 parsers.EndlineNoSub = parsers.endline
9863
9864 parsers.NoSoftLineBreakEndline
9865     = parsers.newline
9866     * (parsers.check_minimal_indent
9867     * -V("NoSoftLineBreakEndlineExceptions")
9868     + parsers.check_optional_indent
9869     * -V("NoSoftLineBreakEndlineExceptions")
9870     * -V("ListStarter"))
9871     * parsers.spacechar^0
9872     / writer.space
9873
9874 parsers.EndlineBreak = parsers.backslash * parsers.endline
9875     / writer.hard_line_break
9876
9877 parsers.OptionalIndent
9878     = parsers.spacechar^1 / writer.space
9879
9880 parsers.Space = parsers.spacechar^2 * parsers.endline

```

```

9881                                     / writer.hard_line_break
9882     + parsers.spacechar^1
9883     * parsers.endline^-1
9884     * parsers.eof / self.expandtabs
9885     + parsers.spacechar^1 * parsers.endline
9886                                     / writer.soft_line_break
9887     + parsers.spacechar^1
9888     * -parsers.newline / self.expandtabs
9889     + parsers.spacechar^1
9890
9891 parsers.NoSoftLineBreakSpace
9892     = parsers.spacechar^2 * parsers.endline
9893                                     / writer.hard_line_break
9894     + parsers.spacechar^1
9895     * parsers.endline^-1
9896     * parsers.eof / self.expandtabs
9897     + parsers.spacechar^1 * parsers.endline
9898                                     / writer.soft_line_break
9899     + parsers.spacechar^1
9900     * -parsers.newline / self.expandtabs
9901     + parsers.spacechar^1
9902
9903 parsers.NonbreakingEndline
9904     = parsers.endline
9905     / writer.nbsp
9906
9907 parsers.NonbreakingSpace
9908     = parsers.spacechar^2 * parsers.endline
9909                                     / writer.nbsp
9910     + parsers.spacechar^1
9911     * parsers.endline^-1 * parsers.eof / ""
9912     + parsers.spacechar^1 * parsers.endline
9913                                     * parsers.optionalspace
9914                                     / writer.nbsp
9915     + parsers.spacechar^1 * parsers.optionalspace
9916                                     / writer.nbsp
9917

```

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.

```

9918 function self.auto_link_url(url, attributes)
9919   return writer.link(writer.escape(url),
9920                     url, nil, attributes)
9921 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.

```
9922 function self.auto_link_email(email, attributes)
9923   return writer.link(writer.escape(email),
9924                     "mailto:".email,
9925                     nil, attributes)
9926 end
9927
9928 parsers.AutoLinkUrl = parsers.auto_link_url
9929                       / self.auto_link_url
9930
9931 parsers.AutoLinkEmail
9932                       = parsers.auto_link_email
9933                       / self.auto_link_email
9934
9935 parsers.AutoLinkRelativeReference
9936                       = parsers.auto_link_relative_reference
9937                       / self.auto_link_url
9938
9939 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9940                       / self.defer_link_and_emphasis_processing
9941
9942 parsers.EscapedChar = parsers.backslash
9943                       * C(parsers.escapable) / writer.string
9944
9945 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9946                       / writer.inline_html_comment
9947                       + Cs(parsers.html_any_empty_inline_tag
9948                           + parsers.html_inline_instruction
9949                           + parsers.html_inline_cdatasection
9950                           + parsers.html_inline_declaration
9951                           + parsers.html_any_open_inline_tag
9952                           + parsers.html_any_close_tag)
9953                       / writer.inline_html_tag
9954
9955 parsers.HtmlEntity = parsers.html_entities / writer.string
```

3.1.6.9 Block Elements (local)

```
9956 parsers.DisplayHtml = Cs(parsers.check_trail
9957                           * ( parsers.html_comment
9958                               + parsers.html_special_block
9959                               + parsers.html_block
9960                               + parsers.html_any_block
9961                               + parsers.html_instruction
9962                               + parsers.html_cdatasection
9963                               + parsers.html_declaration))
```

```

9964             / writer.block_html_element
9965
9966 parsers.indented_non_blank_line = parsers.indentedline
9967             - parsers.blankline
9968
9969 parsers.Verbatim
9970   = Cs( parsers.check_code_trail
9971       * (parsers.line - parsers.blankline)
9972       * (( parsers.check_minimal_blank_indent_and_full_code_trail
9973           * parsers.blankline)^0
9974         * ( (parsers.check_minimal_indent / "")
9975           * parsers.check_code_trail
9976           * (parsers.line - parsers.blankline))^1)^0)
9977   / self.expandtabs / writer.verbatim
9978
9979 parsers.Blockquote   = parsers.blockquote_body
9980                       / writer.blockquote
9981
9982 parsers.ThematicBreak = parsers.thematic_break_lines
9983                       / writer.thematic_break
9984
9985 parsers.Reference    = parsers.define_reference_parser
9986                       / self.register_link
9987
9988 parsers.Paragraph    = parsers.freeze_trail
9989                       * (Ct((parsers.Inline)^1)
9990                       * (parsers.newline + parsers.eof)
9991                       * parsers.unfreeze_trail
9992                       / writer.paragraph)
9993
9994 parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
9995                       / writer.plain

```

3.1.6.10 Lists (local)

```

9996
9997 if options.taskLists then
9998   parsers.tickbox = ( parsers.ticked_box
9999                     + parsers.halfticked_box
10000                     + parsers.unticked_box
10001                     ) / writer.tickbox
10002 else
10003   parsers.tickbox = parsers.fail
10004 end
10005
10006 parsers.list_blank = parsers.conditionally_indented_blankline
10007

```



```

10008 parsers.ref_or_block_list_separated
10009     = parsers.sep_group_no_output(parsers.list_blank)
10010     * parsers.minimally_indented_ref
10011     + parsers.block_sep_group(parsers.list_blank)
10012     * parsers.minimally_indented_block
10013
10014 parsers.ref_or_block_non_separated
10015     = parsers.minimally_indented_ref
10016     + (parsers.succeed / writer.interblocksep)
10017     * parsers.minimally_indented_block
10018     - parsers.minimally_indented_blankline
10019
10020 parsers.tight_list_loop_body_pair =
10021     parsers.create_loop_body_pair(
10022         parsers.ref_or_block_non_separated,
10023         parsers.minimally_indented_par_or_plain_no_blank,
10024         (parsers.succeed / writer.interblocksep),
10025         (parsers.succeed / writer.paragraphsep))
10026
10027 parsers.loose_list_loop_body_pair =
10028     parsers.create_loop_body_pair(
10029         parsers.ref_or_block_list_separated,
10030         parsers.minimally_indented_par_or_plain,
10031         parsers.block_sep_group(parsers.list_blank),
10032         parsers.par_sep_group(parsers.list_blank))
10033
10034 parsers.tight_list_content_loop
10035     = V("Block")
10036     * parsers.tight_list_loop_body_pair.block^0
10037     + (V("Paragraph") + V("Plain"))
10038     * parsers.ref_or_block_non_separated
10039     * parsers.tight_list_loop_body_pair.block^0
10040     + (V("Paragraph") + V("Plain"))
10041     * parsers.tight_list_loop_body_pair.par^0
10042
10043 parsers.loose_list_content_loop
10044     = V("Block")
10045     * parsers.loose_list_loop_body_pair.block^0
10046     + (V("Paragraph") + V("Plain"))
10047     * parsers.ref_or_block_list_separated
10048     * parsers.loose_list_loop_body_pair.block^0
10049     + (V("Paragraph") + V("Plain"))
10050     * parsers.loose_list_loop_body_pair.par^0
10051
10052 parsers.list_item_tightness_condition
10053     = -( parsers.list_blank^0
10054         * parsers.minimally_indented_ref_or_block_or_par)

```

```

10055     * remove_indent("li")
10056     + remove_indent("li")
10057     * parsers.fail
10058
10059 parsers.indented_content_tight
10060     = Ct( (parsers.blankline / "")
10061         * #parsers.list_blank
10062         * remove_indent("li")
10063         + ( (V("Reference") + (parsers.blankline / ""))
10064           * parsers.check_minimal_indent
10065           * parsers.tight_list_content_loop
10066           + (V("Reference") + (parsers.blankline / ""))
10067           + (parsers.tickbox~-1 / writer.escape)
10068           * parsers.tight_list_content_loop
10069         )
10070         * parsers.list_item_tightness_condition)
10071
10072 parsers.indented_content_loose
10073     = Ct( (parsers.blankline / "")
10074         * #parsers.list_blank
10075         + ( (V("Reference") + (parsers.blankline / ""))
10076           * parsers.check_minimal_indent
10077           * parsers.loose_list_content_loop
10078           + (V("Reference") + (parsers.blankline / ""))
10079           + (parsers.tickbox~-1 / writer.escape)
10080           * parsers.loose_list_content_loop))
10081
10082 parsers.TightListItem = function(starter)
10083     return -parsers.ThematicBreak
10084         * parsers.add_indent(starter, "li")
10085         * parsers.indented_content_tight
10086 end
10087
10088 parsers.LooseListItem = function(starter)
10089     return -parsers.ThematicBreak
10090         * parsers.add_indent(starter, "li")
10091         * parsers.indented_content_loose
10092         * remove_indent("li")
10093 end
10094
10095 parsers.BulletListOfType = function(bullet_type)
10096     local bullet = parsers.bullet(bullet_type)
10097     return ( Ct( parsers.TightListItem(bullet)
10098               * ( (parsers.check_minimal_indent / "")
10099                 * parsers.TightListItem(bullet)
10100               )~0
10101     )

```

```

10102         * Cc(true)
10103         * -#( (parsers.list_blank^0 / "")
10104             * parsers.check_minimal_indent
10105             * (bullet - parsers.ThematicBreak)
10106         )
10107         + Ct( parsers.LooseListItem(bullet)
10108             * ( (parsers.list_blank^0 / "")
10109             * (parsers.check_minimal_indent / "")
10110             * parsers.LooseListItem(bullet)
10111             )^0
10112         )
10113         * Cc(false)
10114     ) / writer.bulletlist
10115 end
10116
10117 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
10118                   + parsers.BulletListOfType(parsers.asterisk)
10119                   + parsers.BulletListOfType(parsers.plus)
10120
10121 local function ordered_list(items,tight,starter)
10122     local startnum = starter[2][1]
10123     if options.startNumber then
10124         startnum = tonumber(startnum) or 1 -- fallback for '#'
10125         if startnum ~= nil then
10126             startnum = math.floor(startnum)
10127         end
10128     else
10129         startnum = nil
10130     end
10131     return writer.orderedlist(items,tight,startnum)
10132 end
10133
10134 parsers.OrderedListOfTypes = function(delimiter_type)
10135     local enumerator = parsers.enumerator(delimiter_type)
10136     return Cg(enumerator, "listtype")
10137             * (Ct( parsers.TightListItem(Cb("listtype"))
10138                 * ( (parsers.check_minimal_indent / "")
10139                 * parsers.TightListItem(enumerator))^0
10140             * Cc(true)
10141             * -#((parsers.list_blank^0 / "")
10142                 * parsers.check_minimal_indent * enumerator)
10143             + Ct( parsers.LooseListItem(Cb("listtype"))
10144                 * ((parsers.list_blank^0 / "")
10145                 * (parsers.check_minimal_indent / "")
10146                 * parsers.LooseListItem(enumerator))^0
10147             * Cc(false)
10148             ) * Ct(Cb("listtype"))) / ordered_list

```

```

10149 end
10150
10151 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
10152 + parsers.OrderedListOfType(parsers.rparent)

```

3.1.6.11 Blank (local)

```

10153 parsers.Blank = parsers.blankline / ""
10154 + V("Reference")

```

3.1.6.12 Headings (local)

```

10155 function parsers.parse_heading_text(s)
10156   local inlines = self.parser_functions.parse_inlines(s)
10157   local flatten_inlines = self.writer.flatten_inlines
10158   self.writer.flatten_inlines = true
10159   local flat_text = self.parser_functions.parse_inlines(s)
10160   flat_text = util.ropes_to_string(flat_text)
10161   self.writer.flatten_inlines = flatten_inlines
10162   return {flat_text, inlines}
10163 end
10164
10165 -- parse atx header
10166 parsers.AtxHeading = parsers.check_trail_no_rem
10167 * Cg(parsers.heading_start, "level")
10168 * ((C( parsers.optionalspace
10169   * parsers.hash^0
10170   * parsers.optionalspace
10171   * parsers.newline)
10172 + parsers.spacechar^1
10173 * C(parsers.line))
10174 / strip_atx_end
10175 / parsers.parse_heading_text)
10176 * Cb("level")
10177 / writer.heading
10178
10179 parsers.heading_line = parsers.linechar^1
10180 - parsers.thematic_break_lines
10181
10182 parsers.heading_text = parsers.heading_line
10183 * ( ( V("Endline") / "\n")
10184 * ( parsers.heading_line
10185   - parsers.heading_level))^0
10186 * parsers.newline^-1
10187
10188 parsers.SetextHeading = parsers.freeze_trail
10189 * parsers.check_trail_no_rem
10190 * #( parsers.heading_text

```

```

10191         * parsers.check_minimal_indent
10192         * parsers.check_trail
10193         * parsers.heading_level)
10194     * Cs(parsers.heading_text)
10195     / parsers.parse_heading_text
10196     * parsers.check_minimal_indent_and_trail
10197     * parsers.heading_level
10198     * parsers.newline
10199     * parsers.unfreeze_trail
10200     / writer.heading
10201
10202     parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain T_EX output.

```

10203     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.

```

10204     local walkable_syntax = (function(global_walkable_syntax)
10205         local local_walkable_syntax = {}
10206         for lhs, rule in pairs(global_walkable_syntax) do
10207             local_walkable_syntax[lhs] = util.table_copy(rule)
10208         end
10209         return local_walkable_syntax
10210     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.

```

10211     local current_extension_name = nil
10212     self.insert_pattern = function(selector, pattern, pattern_name)
10213         assert(pattern_name == nil or type(pattern_name) == "string")
10214         local _, _, lhs, pos, rhs
10215         = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
10216         assert(lhs ~= nil,
10217             [[Expected selector in form ]]
10218             .. [["LHS (before|after|instead of) RHS", not "]]
10219             .. selector .. [["]])
10220         assert(walkable_syntax[lhs] ~= nil,
10221             [[Rule ]] .. lhs

```

```

10222     .. [[ -> ... does not exist in markdown grammar]])
10223     assert(pos == "before" or pos == "after" or pos == "instead of",
10224           [[Expected positional specifier "before", "after", ]]
10225           .. [[or "instead of", not "]]
10226           .. pos .. [["]])
10227     local rule = walkable_syntax[lhs]
10228     local index = nil
10229     for current_index, current_rhs in ipairs(rule) do
10230         if type(current_rhs) == "string" and current_rhs == rhs then
10231             index = current_index
10232             if pos == "after" then
10233                 index = index + 1
10234             end
10235             break
10236         end
10237     end
10238     assert(index ~= nil,
10239           [[Rule ]] .. lhs .. [[ -> ]] .. rhs
10240           .. [[ does not exist in markdown grammar]])
10241     local accountable_pattern
10242     if current_extension_name then
10243         accountable_pattern
10244             = {pattern, current_extension_name, pattern_name}
10245     else
10246         assert(type(pattern) == "string",
10247               [[reader->insert_pattern() was called outside ]]
10248               .. [[an extension with ]]
10249               .. [[a PEG pattern instead of a rule name]])
10250         accountable_pattern = pattern
10251     end
10252     if pos == "instead of" then
10253         rule[index] = accountable_pattern
10254     else
10255         table.insert(rule, index, accountable_pattern)
10256     end
10257 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.

```

10258     local syntax =
10259         { "Blocks",
10260
10261           Blocks = V("InitializeState")
10262                 * V("ExpectedJekyllData")
10263                 * 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.

```

10264         * ( V("Block")
10265         * ( V("Blank")^0 * parsers.eof
10266         + ( V("Blank")^2 / writer.paragraphsep
10267         + V("Blank")^0 / writer.interblocksep
10268         )
10269         )
10270         + ( V("Paragraph") + V("Plain") )
10271         * ( V("Blank")^0 * parsers.eof
10272         + ( V("Blank")^2 / writer.paragraphsep
10273         + V("Blank")^0 / writer.interblocksep
10274         )
10275         )
10276         * V("Block")
10277         * ( V("Blank")^0 * parsers.eof
10278         + ( V("Blank")^2 / writer.paragraphsep
10279         + V("Blank")^0 / writer.interblocksep
10280         )
10281         )
10282         + ( V("Paragraph") + V("Plain") )
10283         * ( V("Blank")^0 * parsers.eof
10284         + V("Blank")^0 / writer.paragraphsep
10285         )
10286         )^0,
10287
10288     ExpectedJekyllData = parsers.succeed,
10289
10290     Blank                = parsers.Blank,
10291     Reference            = parsers.Reference,
10292
10293     Blockquote           = parsers.Blockquote,
10294     Verbatim             = parsers.Verbatim,
10295     ThematicBreak        = parsers.ThematicBreak,
10296     BulletList           = parsers.BulletList,
10297     OrderedList          = parsers.OrderedList,
10298     DisplayHtml          = parsers.DisplayHtml,
10299     Heading              = parsers.Heading,
10300     Paragraph            = parsers.Paragraph,
10301     Plain                = parsers.Plain,
10302
10303     ListStarter          = parsers.ListStarter,
10304     EndlineExceptions    = parsers.EndlineExceptions,
10305     NoSoftLineBreakEndlineExceptions
10306                         = parsers.NoSoftLineBreakEndlineExceptions,
10307
10308     Str                  = parsers.Str,

```

```

10309     Space           = parsers.Space,
10310     NoSoftLineBreakSpace
10311         = parsers.NoSoftLineBreakSpace,
10312     OptionalIndent   = parsers.OptionalIndent,
10313     Endline           = parsers.Endline,
10314     EndlineNoSub     = parsers.EndlineNoSub,
10315     NoSoftLineBreakEndline
10316         = parsers.NoSoftLineBreakEndline,
10317     EndlineBreak     = parsers.EndlineBreak,
10318     LinkAndEmph      = parsers.LinkAndEmph,
10319     Code             = parsers.Code,
10320     AutoLinkUrl      = parsers.AutoLinkUrl,
10321     AutoLinkEmail    = parsers.AutoLinkEmail,
10322     AutoLinkRelativeReference
10323         = parsers.AutoLinkRelativeReference,
10324     InlineHtml       = parsers.InlineHtml,
10325     HtmlEntity       = parsers.HtmlEntity,
10326     EscapedChar      = parsers.EscapedChar,
10327     Smart            = parsers.Smart,
10328     Symbol           = parsers.Symbol,
10329     SpecialChar      = parsers.fail,
10330     InitializeState  = parsers.succeed,
10331 }

```

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].

```

10332     self.update_rule = function(rule_name, get_pattern)
10333         assert(current_extension_name ~= nil)
10334         assert(syntax[rule_name] ~= nil,
10335             [[Rule ]] .. rule_name
10336             .. [[ -> ... does not exist in markdown grammar]])
10337         local previous_pattern
10338         local extension_name
10339         if walkable_syntax[rule_name] then
10340             local previous_accountable_pattern
10341             = walkable_syntax[rule_name][1]
10342             previous_pattern = previous_accountable_pattern[1]
10343             extension_name
10344             = previous_accountable_pattern[2]
10345             .. ", " .. current_extension_name
10346         else
10347             previous_pattern = nil
10348             extension_name = current_extension_name
10349         end

```



```
10350     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
```

```
10351     if type(get_pattern) == "function" then
10352         pattern = get_pattern(previous_pattern)
10353     else
10354         assert(previous_pattern == nil,
10355             [[Rule ]] .. rule_name ..
10356             [[ has already been updated by ]] .. extension_name)
10357         pattern = get_pattern
10358     end
10359     local accountable_pattern = { pattern, extension_name, rule_name }
10360     walkable_syntax[rule_name] = { accountable_pattern }
10361 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.

```
10362     local special_characters = {}
10363     self.add_special_character = function(c)
10364         table.insert(special_characters, c)
10365         syntax.SpecialChar = S(table.concat(special_characters, ""))
10366     end
10367
10368     self.add_special_character("*")
10369     self.add_special_character("[")
10370     self.add_special_character("]")
10371     self.add_special_character("<")
10372     self.add_special_character("!")
10373     self.add_special_character("\\")
```

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

```
10374     self.initialize_named_group = function(name, value)
10375         local pattern = Ct("")
10376         if value ~= nil then
10377             pattern = pattern / value
10378         end
```

```

10379     syntax.InitializeState = syntax.InitializeState
10380         * Cg(pattern, name)
10381     end

```

Add a named group for indentation.

```

10382     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

10383     for _, extension in ipairs(extensions) do
10384         current_extension_name = extension.name
10385         extension.extend_writer(writer)
10386         extension.extend_reader(self)
10387     end
10388     current_extension_name = nil

```

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

```

10389     if options.debugExtensions then
10390         local sorted_lhs = {}
10391         for lhs, _ in pairs(walkable_syntax) do
10392             table.insert(sorted_lhs, lhs)
10393         end
10394         table.sort(sorted_lhs)
10395
10396         local output_lines = {"{"}
10397         for lhs_index, lhs in ipairs(sorted_lhs) do
10398             local encoded_lhs = util.encode_json_string(lhs)
10399             table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": []}])
10400             local rule = walkable_syntax[lhs]
10401             for rhs_index, rhs in ipairs(rule) do
10402                 local human_readable_rhs
10403                 if type(rhs) == "string" then
10404                     human_readable_rhs = rhs
10405                 else
10406                     local pattern_name
10407                     if rhs[3] then
10408                         pattern_name = rhs[3]
10409                     else
10410                         pattern_name = "Anonymous Pattern"
10411                     end
10412                     local extension_name = rhs[2]
10413                     human_readable_rhs = pattern_name .. [{" ("]
10414                         .. extension_name .. [{" )"}]
10415                 end
10416                 local encoded_rhs
10417                 = util.encode_json_string(human_readable_rhs)
10418                 local output_line = [{" "] .. encoded_rhs
10419                 if rhs_index < #rule then

```

```

10420         output_line = output_line .. ","
10421     end
10422     table.insert(output_lines, output_line)
10423 end
10424 local output_line = "    ]"
10425 if lhs_index < #sorted_lhs then
10426     output_line = output_line .. ","
10427 end
10428 table.insert(output_lines, output_line)
10429 end
10430 table.insert(output_lines, "}")
10431
10432 local output = table.concat(output_lines, "\n")
10433 local output_filename = options.debugExtensionsFileName
10434 local output_file = assert(io.open(output_filename, "w"),
10435     [[Could not open file ]] .. output_filename
10436     .. [[ for writing]])
10437 assert(output_file:write(output))
10438 assert(output_file:close())
10439 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`.

```

10440     for lhs, rule in pairs(walkable_syntax) do
10441         syntax[lhs] = parsers.fail
10442         for _, rhs in ipairs(rule) do
10443             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.

```

10444         if type(rhs) == "string" then
10445             pattern = V(rhs)
10446         else
10447             pattern = rhs[1]
10448             if type(pattern) == "string" then
10449                 pattern = V(pattern)
10450             end
10451         end
10452         syntax[lhs] = syntax[lhs] + pattern
10453     end
10454 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.

```
10455     if options.underscores then
10456         self.add_special_character("_")
10457     end
10458
10459     if not options.codeSpans then
10460         syntax.Code = parsers.fail
10461     else
10462         self.add_special_character("`")
10463     end
10464
10465     if not options.html then
10466         syntax.DisplayHtml = parsers.fail
10467         syntax.InlineHtml = parsers.fail
10468         syntax.HtmlEntity = parsers.fail
10469     else
10470         self.add_special_character("&")
10471     end
10472
10473     if options.preserveTabs then
10474         options.stripIndent = false
10475     end
10476
10477     if not options.smartEllipses then
10478         syntax.Smart = parsers.fail
10479     else
10480         self.add_special_character(".")
10481     end
10482
10483     if not options.relativeReferences then
10484         syntax.AutoLinkRelativeReference = parsers.fail
10485     end
10486
10487     if options.contentLevel == "inline" then
10488         syntax[1] = "Inlines"
10489         syntax.Inlines = V("InitializeState")
10490             * parsers.Inline^0
10491             * ( parsers.spacing^0
10492                 * parsers.eof / "")
10493         syntax.Space = parsers.Space + parsers.blankline / writer.space
10494     end
10495
10496     local blocks_nested_t = util.table_copy(syntax)
10497     blocks_nested_t.ExpectedJekyllData = parsers.succeed
10498     parsers.blocks_nested = Ct(blocks_nested_t)
10499
```

```

10500     parsers.blocks = Ct(syntax)
10501
10502     local inlines_t = util.table_copy(syntax)
10503     inlines_t[1] = "Inlines"
10504     inlines_t.Inlines = V("InitializeState")
10505         * parsers.Inline^0
10506         * ( parsers.spacing^0
10507           * parsers.eof / "")
10508     parsers.inlines = Ct(inlines_t)
10509
10510     local inlines_no_inline_note_t = util.table_copy(inlines_t)
10511     inlines_no_inline_note_t.InlineNote = parsers.fail
10512     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10513
10514     local inlines_no_html_t = util.table_copy(inlines_t)
10515     inlines_no_html_t.DisplayHtml = parsers.fail
10516     inlines_no_html_t.InlineHtml = parsers.fail
10517     inlines_no_html_t.HtmlEntity = parsers.fail
10518     parsers.inlines_no_html = Ct(inlines_no_html_t)
10519
10520     local inlines_nbsp_t = util.table_copy(inlines_t)
10521     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10522     inlines_nbsp_t.Space = parsers.NonbreakingSpace
10523     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10524
10525     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10526     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10527     inlines_no_link_or_emphasis_t.EndlineExceptions
10528     = parsers.EndlineExceptions - parsers.eof
10529     parsers.inlines_no_link_or_emphasis
10530     = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain $\text{T}_{\text{E}}\text{X}$ output and returns it..

```

10531     return function(input)
Unicode-normalize the input.
10532         if options.unicodeNormalization then
10533             local form = options.unicodeNormalizationForm
10534             if form == "nfc" then
10535                 input = uni_algos.normalize.NFC(input)
10536             elseif form == "nfd" then
10537                 input = uni_algos.normalize.NFD(input)
10538             elseif form == "nfkc" then
10539                 input = uni_algos.normalize.NFKC(input)
10540             elseif form == "nfkd" then
10541                 input = uni_algos.normalize.NFKD(input)
10542             else

```

```

10543         return writer.error(
10544             format("Unknown normalization form %s.", form))
10545     end
10546 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.

```

10547     input = input:gsub("\r\n?", "\n")
10548     if input:sub(-1) ~= "\n" then
10549         input = input .. "\n"
10550     end

```

Clear the table of references.

```

10551     references = {}
10552     local document = self.parser_functions.parse_blocks(input)
10553     local output = util.rope_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](#).

```

10554     local undosep_start, undosep_end
10555     local potential_secend_start, secend_start
10556     local potential_sep_start, sep_start
10557     while true do
10558         -- find a `writer->undosep`
10559         undosep_start, undosep_end
10560             = output:find(writer.undosep_text, 1, true)
10561         if undosep_start == nil then break end
10562         -- skip any preceding section ends
10563         secend_start = undosep_start
10564         while true do
10565             potential_secend_start = secend_start - #writer.secend_text
10566             if potential_secend_start < 1
10567                 or output:sub(potential_secend_start,
10568                     secend_start - 1) ~= writer.secend_text
10569             then
10570                 break
10571             end
10572             secend_start = potential_secend_start
10573         end
10574         -- find an immediately preceding
10575         -- block element / paragraph separator
10576         sep_start = secend_start
10577         potential_sep_start = sep_start - #writer.interblocksep_text
10578         if potential_sep_start >= 1
10579             and output:sub(potential_sep_start,
10580                 sep_start - 1) == writer.interblocksep_text
10581         then

```

```

10582         sep_start = potential_sep_start
10583     else
10584         potential_sep_start = sep_start - #writer.paragraphsep_text
10585         if potential_sep_start >= 1
10586             and output:sub(potential_sep_start,
10587                 sep_start - 1) == writer.paragraphsep_text
10588             then
10589                 sep_start = potential_sep_start
10590             end
10591         end
10592         -- remove `writer->undosep` and immediately preceding
10593         -- block element / paragraph separator
10594         output = output:sub(1, sep_start - 1)
10595             .. output:sub(secend_start, undosep_start - 1)
10596             .. output:sub(undosep_end + 1)
10597     end
10598     return output
10599 end
10600 end
10601 return self
10602 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.

```
10603 M.extensions = {}
```

3.1.7.1 Bracketed Spans

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

```

10604 M.extensions.bracketed_spans = function()
10605     return {
10606         name = "built-in bracketed_spans syntax extension",
10607         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.

```

10608         function self.span(s, attr)
10609             if self.flatten_inlines then return s end
10610             return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10611                 self.attributes(attr),
10612                 s,

```

```

10613         "\\markdownRendererBracketedSpanAttributeContextEnd{}}"
10614     end
10615 end, extend_reader = function(self)
10616     local parsers = self.parsers
10617     local writer = self.writer
10618
10619     local span_label = parsers.lbracket
10620                     * (Cs((parsers.alphanumeric^1
10621                          + parsers.inticks
10622                          + parsers.autolink
10623                          + V("InlineHtml")
10624                          + ( parsers.backslash * parsers.backslash)
10625                          + ( parsers.backslash
10626                            * (parsers.lbracket + parsers.rbracket)
10627                            + V("Space") + V("Endline")
10628                            + (parsers.any
10629                               - ( parsers.newline
10630                                  + parsers.lbracket
10631                                  + parsers.rbracket
10632                                  + parsers.blankline^2))))^1)
10633                     / self.parser_functions.parse_inlines)
10634                     * parsers.rbracket
10635
10636     local Span = span_label
10637                 * Ct(parsers.attributes)
10638                 / writer.span
10639
10640     self.insert_pattern("Inline before LinkAndEmph",
10641                       Span, "Span")
10642 end
10643 }
10644 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.

```

10645 M.extensions.citations = function(citation_nbsps)
10646     return {
10647         name = "built-in citations syntax extension",
10648         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.

```
10649     function self.citations(text_cites, cites)
10650         local buffer = {}
10651         if self.flatten_inlines then
10652             for _,cite in ipairs(cites) do
10653                 if cite.prenote then
10654                     table.insert(buffer, {cite.prenote, " "})
10655                 end
10656                 table.insert(buffer, cite.name)
10657                 if cite.postnote then
10658                     table.insert(buffer, {" ", cite.postnote})
10659                 end
10660             end
10661         else
10662             table.insert(buffer,
10663                 {"\\markdownRenderer",
10664                 text_cites and "TextCite" or "Cite",
10665                 "{", #cites, "}"}))
10666             for _,cite in ipairs(cites) do
10667                 table.insert(buffer,
10668                     {cite.suppress_author and "-" or "+", "{",
10669                     cite.prenote or "", "}{" ,
10670                     cite.postnote or "", "}{" , cite.name, "}"}))
10671             end
10672         end
10673         return buffer
10674     end
10675 end, extend_reader = function(self)
10676     local parsers = self.parsers
10677     local writer = self.writer
10678
10679     local citation_chars
10680         = parsers.alphanumeric
10681         + S("#$%&-+<>-/_")
10682
```

```

10683     local citation_name
10684         = Cs(parsers.dash^-1) * parsers.at
10685         * Cs(citation_chars
10686             * ((( citation_chars
10687                 + parsers.internal_punctuation
10688                 - parsers.comma - parsers.semicolon)
10689                 * -#(( parsers.internal_punctuation
10690                     - parsers.comma
10691                     - parsers.semicolon)^0
10692                 * -( citation_chars
10693                     + parsers.internal_punctuation
10694                     - parsers.comma
10695                     - parsers.semicolon)))^0
10696             * citation_chars)^-1)
10697
10698     local citation_body_prenote
10699         = Cs((parsers.alphanumeric^1
10700             + parsers.bracketed
10701             + parsers.inticks
10702             + parsers.autolink
10703             + V("InlineHtml")
10704             + V("Space") + V("EndlineNoSub")
10705             + (parsers.anyescaped
10706                 - ( parsers.newline
10707                     + parsers.rbracket
10708                     + parsers.blankline^2))
10709             - ( parsers.spnl
10710                 * parsers.dash^-1
10711                 * parsers.at))^1)
10712
10713     local citation_body_postnote
10714         = Cs((parsers.alphanumeric^1
10715             + parsers.bracketed
10716             + parsers.inticks
10717             + parsers.autolink
10718             + V("InlineHtml")
10719             + V("Space") + V("EndlineNoSub")
10720             + (parsers.anyescaped
10721                 - ( parsers.newline
10722                     + parsers.rbracket
10723                     + parsers.semicolon
10724                     + parsers.blankline^2))
10725             - (parsers.spnl * parsers.rbracket))^1)
10726
10727     local citation_body_chunk
10728         = ( citation_body_prenote
10729             * parsers.spnlc_sep

```

```

10730         + Cc("")
10731         * parsers.spnlc
10732     )
10733     * citation_name
10734     * ( parsers.internal_punctuation
10735       - parsers.semicolon)^-1
10736     * ( parsers.spnlc / function(_) return end
10737       * citation_body_postnote
10738       + Cc("")
10739       * parsers.spnlc
10740     )
10741
10742     local citation_body
10743         = citation_body_chunk
10744         * ( parsers.semicolon
10745           * parsers.spnlc
10746           * citation_body_chunk
10747         )^0
10748
10749     local citation_headless_body_postnote
10750         = Cs((parsers.alphanumeric^1
10751             + parsers.bracketed
10752             + parsers.inticks
10753             + parsers.autolink
10754             + V("InlineHtml")
10755             + V("Space") + V("Endline")
10756             + (parsers.anyescaped
10757               - ( parsers.newline
10758                 + parsers.rbracket
10759                 + parsers.at
10760                 + parsers.semicolon + parsers.blankline^2))
10761             - (parsers.spnl * parsers.rbracket))^0)
10762
10763     local citation_headless_body
10764         = citation_headless_body_postnote
10765         * ( parsers.semicolon
10766           * parsers.spnlc
10767           * citation_body_chunk
10768         )^0
10769
10770     local citations
10771         = function(text_cites, raw_cites)
10772             local function normalize(str)
10773                 if str == "" then
10774                     str = nil
10775                 else
10776                     str = (citation_nbsps and

```

```

10777             self.parser_functions.parse_inlines_nbsp or
10778             self.parser_functions.parse_inlines)(str)
10779         end
10780         return str
10781     end
10782
10783     local cites = {}
10784     for i = 1,#raw_cites,4 do
10785         cites[#cites+1] = {
10786             prenote = normalize(raw_cites[i]),
10787             suppress_author = raw_cites[i+1] == "-",
10788             name = writer.identifier(raw_cites[i+2]),
10789             postnote = normalize(raw_cites[i+3]),
10790         }
10791     end
10792     return writer.citations(text_cites, cites)
10793 end
10794
10795 local TextCitations
10796     = Ct((parsers.spnlc
10797     * Cc("")
10798     * citation_name
10799     * ((parsers.spnlc
10800     * parsers.lbracket
10801     * citation_headless_body
10802     * parsers.rbracket) + Cc("")))~1)
10803 / function(raw_cites)
10804     return citations(true, raw_cites)
10805 end
10806
10807 local ParenthesizedCitations
10808     = Ct((parsers.spnlc
10809     * parsers.lbracket
10810     * citation_body
10811     * parsers.rbracket)^1)
10812 / function(raw_cites)
10813     return citations(false, raw_cites)
10814 end
10815
10816 local Citations = TextCitations + ParenthesizedCitations
10817
10818 self.insert_pattern("Inline before LinkAndEmph",
10819     Citations, "Citations")
10820
10821 self.add_special_character("@")
10822 self.add_special_character("-")
10823 end

```

```

10824 }
10825 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.

```

10826 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.

```

10827 local languages_json = (function()
10828     local base, prev, curr
10829     for _, pathname in ipairs{kpse.lookup(language_map,
10830                                     {all=true})} do
10831         local file = io.open(pathname, "r")
10832         if not file then goto continue end
10833         local input = assert(file:read("*a"))
10834         assert(file:close())
10835         local json = input:gsub('[^\n]-:', '[%1]=')
10836         curr = load("_ENV = {}; return "..json")()
10837         if type(curr) == "table" then
10838             if base == nil then
10839                 base = curr
10840             else
10841                 setmetatable(prev, { __index = curr })
10842             end
10843             prev = curr
10844         end
10845         ::continue::
10846     end
10847     return base or {}
10848 end)()
10849
10850 return {
10851     name = "built-in content_blocks syntax extension",
10852     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.

```

10853         function self.contentblock(src,suf,type,tit)
10854             if not self.is_writing then return "" end

```

```

10855     src = src.." "..suf
10856     suf = suf:lower()
10857     if type == "onlineimage" then
10858         return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10859             "{" ,self.string(src),"} ",
10860             "{" ,self.uri(src),"} ",
10861             "{" ,self.string(tit or ""),"}"}
10862     elseif languages_json[suf] then
10863         return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10864             "{" ,self.string(languages_json[suf]),"} ",
10865             "{" ,self.string(src),"} ",
10866             "{" ,self.uri(src),"} ",
10867             "{" ,self.string(tit or ""),"}"}
10868     else
10869         return {"\\markdownRendererContentBlock{" ,suf,"} ",
10870             "{" ,self.string(src),"} ",
10871             "{" ,self.uri(src),"} ",
10872             "{" ,self.string(tit or ""),"}"}
10873     end
10874 end
10875 end, extend_reader = function(self)
10876     local parsers = self.parsers
10877     local writer = self.writer
10878
10879     local contentblock_tail
10880         = parsers.optionaltitle
10881         * (parsers.newline + parsers.eof)
10882
10883     -- case insensitive online image suffix:
10884     local onlineimagesuffix
10885         = (function(...)
10886             local parser = nil
10887             for _, suffix in ipairs({...}) do
10888                 local pattern=nil
10889                 for i=1,#suffix do
10890                     local char=suffix:sub(i,i)
10891                     char = S(char:lower()..char:upper())
10892                     if pattern == nil then
10893                         pattern = char
10894                     else
10895                         pattern = pattern * char
10896                     end
10897                 end
10898                 if parser == nil then
10899                     parser = pattern
10900                 else
10901                     parser = parser + pattern

```

```

10902             end
10903         end
10904         return parser
10905     end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10906
10907     -- online image url for iA Writer content blocks with
10908     -- mandatory suffix, allowing nested brackets:
10909     local onlineimageurl
10910         = (parsers.less
10911           * Cs((parsers.anyescaped
10912              - parsers.more
10913              - parsers.spacing
10914              - #(parsers.period
10915                 * onlineimagesuffix
10916                 * parsers.more
10917                 * contentblock_tail))^0)
10918           * parsers.period
10919           * Cs(onlineimagesuffix)
10920           * parsers.more
10921           + (Cs((parsers.inparens
10922              + (parsers.anyescaped
10923                 - parsers.spacing
10924                 - parsers.rparent
10925                 - #(parsers.period
10926                    * onlineimagesuffix
10927                    * contentblock_tail))))^0)
10928           * parsers.period
10929           * Cs(onlineimagesuffix)
10930           ) * Cc("onlineimage")
10931
10932     -- filename for iA Writer content blocks with mandatory suffix:
10933     local localfilepath
10934         = parsers.slash
10935         * Cs((parsers.anyescaped
10936            - parsers.tab
10937            - parsers.newline
10938            - #(parsers.period
10939               * parsers.alphanumeric^1
10940               * contentblock_tail))^1)
10941         * parsers.period
10942         * Cs(parsers.alphanumeric^1)
10943         * Cc("localfile")
10944
10945     local ContentBlock
10946         = parsers.check_trail_no_rem
10947         * (localfilepath + onlineimageurl)
10948         * contentblock_tail

```

```

10949             / writer.contentblock
10950
10951         self.insert_pattern("Block before Blockquote",
10952                             ContentBlock, "ContentBlock")
10953     end
10954 }
10955 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.

```

10956 M.extensions.definition_lists = function(tight_lists)
10957     return {
10958         name = "built-in definition_lists syntax extension",
10959         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.

```

10960     local function dlitem(term, defs)
10961         local retVal = {"\\markdownRendererDlItem{",term,""}
10962         for _, def in ipairs(defs) do
10963             retVal[#retVal+1]
10964                 = {"\\markdownRendererDlDefinitionBegin ",def,
10965                   "\\markdownRendererDlDefinitionEnd "}
10966         end
10967         retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10968         return retVal
10969     end
10970
10971     function self.definitionlist(items,tight)
10972         if not self.is_writing then return "" end
10973         local buffer = {}
10974         for _,item in ipairs(items) do
10975             buffer[#buffer + 1] = dlitem(item.term, item.definitions)
10976         end
10977         if tight and tight_lists then
10978             return {"\\markdownRendererDlBeginTight\n", buffer,
10979                   "\n\\markdownRendererDlEndTight"}
10980         else
10981             return {"\\markdownRendererDlBegin\n", buffer,
10982                   "\n\\markdownRendererDlEnd"}
10983         end
10984     end

```



```

10985 end, extend_reader = function(self)
10986     local parsers = self.parsers
10987     local writer = self.writer
10988
10989     local defstartchar = S("~:")
10990
10991     local defstart
10992         = parsers.check_trail_length(0) * defstartchar
10993         * #parsers.spacing
10994         * (parsers.tab + parsers.space^-3)
10995         + parsers.check_trail_length(1)
10996         * defstartchar * #parsers.spacing
10997         * (parsers.tab + parsers.space^-2)
10998         + parsers.check_trail_length(2)
10999         * defstartchar * #parsers.spacing
11000         * (parsers.tab + parsers.space^-1)
11001         + parsers.check_trail_length(3)
11002         * defstartchar * #parsers.spacing
11003
11004     local indented_line
11005         = (parsers.check_minimal_indent / "")
11006         * parsers.check_code_trail * parsers.line
11007
11008     local blank
11009         = parsers.check_minimal_blank_indent_and_any_trail
11010         * parsers.optionalspace * parsers.newline
11011
11012     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
11013
11014     local indented_blocks = function(bl)
11015         return Cs( bl
11016             * (blank^1 * (parsers.check_minimal_indent / ""))
11017             * parsers.check_code_trail * -parsers.blankline * bl)^0
11018             * (blank^1 + parsers.eof))
11019     end
11020
11021     local function definition_list_item(term, defs, _)
11022         return { term = self.parser_functions.parse_inlines(term),
11023             definitions = defs }
11024     end
11025
11026     local DefinitionListItemLoose
11027         = C(parsers.line) * blank^0
11028         * Ct((parsers.check_minimal_indent * (defstart
11029             * indented_blocks(dlchunk)
11030             / self.parser_functions.parse_blocks_nested))^1)
11031         * Cc(false) / definition_list_item

```

```

11032
11033     local DefinitionListItemTight
11034         = C(parsers.line)
11035         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
11036             / self.parser_functions.parse_blocks_nested))^1)
11037         * Cc(true) / definition_list_item
11038
11039     local DefinitionList
11040         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
11041           + Ct(DefinitionListItemTight^1)
11042           * (blank^0
11043             * -DefinitionListItemLoose * Cc(true))
11044           ) / writer.definitionlist
11045
11046     self.insert_pattern("Block after Heading",
11047                       DefinitionList, "DefinitionList")
11048   end
11049 }
11050 end

```

3.1.7.5 Fancy Lists

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

```

11051 M.extensions.fancy_lists = function()
11052   return {
11053     name = "built-in fancy_lists syntax extension",
11054     extend_writer = function(self)
11055       local options = self.options
11056

```

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.

```

11057     function self.fancylist(items,tight,startnum,numstyle,numdelim)
11058         if not self.is_writing then return "" end
11059         local buffer = {}
11060         local num = startnum
11061         for _,item in ipairs(items) do
11062             if item ~= "" then
11063                 buffer[#buffer + 1] = self.fancyitem(item,num)
11064             end
11065             if num ~= nil and item ~= "" then
11066                 num = num + 1
11067             end
11068         end
11069         local contents = util.intersperse(buffer,"\n")
11070         if tight and options.tightLists then
11071             return {"\\markdownRendererFancyOlBeginTight{" ,
11072                 numstyle,"}{" ,numdelim,"}"} ,contents,
11073                 "\\markdownRendererFancyOlEndTight "}
11074         else
11075             return {"\\markdownRendererFancyOlBegin{" ,
11076                 numstyle,"}{" ,numdelim,"}"} ,contents,
11077                 "\\markdownRendererFancyOlEnd "}
11078         end
11079     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.

```

11080     function self.fancyitem(s,num)
11081         if num ~= nil then
11082             return {"\\markdownRendererFancyOlItemWithNumber{" ,num,"}"} ,s,
11083                 "\\markdownRendererFancyOlItemEnd "}
11084         else
11085             return {"\\markdownRendererFancyOlItem " ,s,
11086                 "\\markdownRendererFancyOlItemEnd "}
11087         end
11088     end
11089     end, extend_reader = function(self)
11090         local parsers = self.parsers
11091         local options = self.options
11092         local writer = self.writer
11093
11094         local function combine_markers_and_delims(markers, delims)
11095             local markers_table = {}

```

```

11096     for _,marker in ipairs(markers) do
11097         local start_marker
11098         local continuation_marker
11099         if type(marker) == "table" then
11100             start_marker = marker[1]
11101             continuation_marker = marker[2]
11102         else
11103             start_marker = marker
11104             continuation_marker = marker
11105         end
11106         for _,delim in ipairs(delims) do
11107             table.insert(markers_table,
11108                 {start_marker, continuation_marker, delim})
11109         end
11110     end
11111     return markers_table
11112 end
11113
11114 local function join_table_with_func(func, markers_table)
11115     local pattern = func(table.unpack(markers_table[1]))
11116     for i = 2, #markers_table do
11117         pattern = pattern + func(table.unpack(markers_table[i]))
11118     end
11119     return pattern
11120 end
11121
11122 local lowercase_letter_marker = R("az")
11123 local uppercase_letter_marker = R("AZ")
11124
11125 local roman_marker = function(chars)
11126     local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
11127     local l, x, v, i
11128         = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
11129     return m^-3
11130         * (c*m + c*d + d^-1 * c^-3)
11131         * (x*c + x*l + l^-1 * x^-3)
11132         * (i*x + i*v + v^-1 * i^-3)
11133 end
11134
11135 local lowercase_roman_marker
11136     = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
11137 local uppercase_roman_marker
11138     = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
11139
11140 local lowercase_opening_roman_marker = P("i")
11141 local uppercase_opening_roman_marker = P("I")
11142

```

```

11143     local digit_marker = parsers.dig * parsers.dig^-8
11144
11145     local markers = {
11146         {lowercase_opening_roman_marker, lowercase_roman_marker},
11147         {uppercase_opening_roman_marker, uppercase_roman_marker},
11148         lowercase_letter_marker,
11149         uppercase_letter_marker,
11150         lowercase_roman_marker,
11151         uppercase_roman_marker,
11152         digit_marker
11153     }
11154
11155     local delims = {
11156         parsers.period,
11157         parsers.rparent
11158     }
11159
11160     local markers_table = combine_markers_and_delims(markers, delims)
11161
11162     local function enumerator(start_marker, _,
11163                             delimiter_type, interrupting)
11164         local delimiter_range
11165         local allowed_end
11166         if interrupting then
11167             delimiter_range = P("1")
11168             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
11169         else
11170             delimiter_range = start_marker
11171             allowed_end = C(parsers.spacechar^1)
11172                 + #(parsers.newline + parsers.eof)
11173         end
11174
11175         return parsers.check_trail
11176             * Ct(C(delimiter_range) * C(delimiter_type))
11177             * allowed_end
11178     end
11179
11180     local starter = join_table_with_func(enumerator, markers_table)
11181
11182     local TightListItem = function(starter)
11183         return parsers.add_indent(starter, "li")
11184             * parsers.indented_content_tight
11185     end
11186
11187     local LooseListItem = function(starter)
11188         return parsers.add_indent(starter, "li")
11189             * parsers.indented_content_loose

```

```

11190             * remove_indent("li")
11191     end
11192
11193     local function roman2number(roman)
11194         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
11195                         ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
11196         local numeral = 0
11197
11198         local i = 1
11199         local len = string.len(roman)
11200         while i < len do
11201             local z1, z2 = romans[ string.sub(roman, i, i) ],
11202                             romans[ string.sub(roman, i+1, i+1) ]
11203             if z1 < z2 then
11204                 numeral = numeral + (z2 - z1)
11205                 i = i + 2
11206             else
11207                 numeral = numeral + z1
11208                 i = i + 1
11209             end
11210         end
11211         if i <= len then
11212             numeral = numeral + romans[ string.sub(roman,i,i) ]
11213         end
11214         return numeral
11215     end
11216
11217     local function sniffstyle(numstr, delimend)
11218         local numdelim
11219         if delimend == ")" then
11220             numdelim = "OneParen"
11221         elseif delimend == "." then
11222             numdelim = "Period"
11223         else
11224             numdelim = "Default"
11225         end
11226
11227         local num
11228         num = numstr:match("^([I])$")
11229         if num then
11230             return roman2number(num), "UpperRoman", numdelim
11231         end
11232         num = numstr:match("^([i])$")
11233         if num then
11234             return roman2number(string.upper(num)), "LowerRoman", numdelim
11235         end
11236         num = numstr:match("^([A-Z])$")

```

```

11237     if num then
11238         return string.byte(num) - string.byte("A") + 1,
11239             "UpperAlpha", numdelim
11240     end
11241     num = numstr:match("^([a-z])$")
11242     if num then
11243         return string.byte(num) - string.byte("a") + 1,
11244             "LowerAlpha", numdelim
11245     end
11246     num = numstr:match("^([IVXLCDM]+)")
11247     if num then
11248         return roman2number(num), "UpperRoman", numdelim
11249     end
11250     num = numstr:match("^([ivxlcdm]+)")
11251     if num then
11252         return roman2number(string.upper(num)), "LowerRoman", numdelim
11253     end
11254     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
11255 end
11256
11257 local function fancylist(items,tight,start)
11258     local startnum, numstyle, numdelim
11259     = sniffstyle(start[2][1], start[2][2])
11260     return writer.fancylist(items,tight,
11261                             options.startNumber and startnum or 1,
11262                             numstyle or "Decimal",
11263                             numdelim or "Default")
11264 end
11265
11266 local FancyListOfType
11267 = function(start_marker, continuation_marker, delimiter_type)
11268     local enumerator_start
11269     = enumerator(start_marker, continuation_marker,
11270                 delimiter_type)
11271     local enumerator_cont
11272     = enumerator(continuation_marker, continuation_marker,
11273                 delimiter_type)
11274     return Cg(enumerator_start, "listtype")
11275         * (Ct( TightListItem(Cb("listtype"))
11276             * ((parsers.check_minimal_indent / "")
11277             * TightListItem(enumerator_cont))^0)
11278         * Cc(true)
11279         * -#((parsers.conditionally_indented_blankline^0 / "")
11280             * parsers.check_minimal_indent * enumerator_cont)
11281         + Ct( LooseListItem(Cb("listtype"))
11282             * ((parsers.conditionally_indented_blankline^0 / "")
11283             * (parsers.check_minimal_indent / ""))

```

```

11284             * LooseListItem(enumerator_cont))^0)
11285         * Cc(false)
11286         ) * Ct(Cb("listtype")) / fancylist
11287     end
11288
11289     local FancyList
11290         = join_table_with_func(FancyListOfType, markers_table)
11291
11292     local ListStarter = starter
11293
11294     self.update_rule("OrderedList", FancyList)
11295     self.update_rule("ListStarter", ListStarter)
11296 end
11297 }
11298 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.

```

11299 M.extensions.fenced_code = function(blank_before_code_fence,
11300                                     allow_attributes,
11301                                     allow_raw_blocks)
11302     return {
11303         name = "built-in fenced_code syntax extension",
11304         extend_writer = function(self)
11305             local options = self.options
11306

```

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.

```

11307     function self.fencedCode(s, i, attr)
11308         if not self.is_writing then return "" end
11309         s = s:gsub("\n$", "")
11310         local buf = {}
11311         if attr ~= nil then
11312             table.insert(buf,
11313                 {"\markdownRendererFencedCodeAttributeContextBegin",
11314                 self.attributes(attr)})
11315         end
11316         local name = util.cache_verbatim(options.cacheDir, s)

```



```

11317     table.insert(buf,
11318         {"\\markdownRendererInputFencedCode{" ,
11319         name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}")
11320     if attr ~= nil then
11321         table.insert(buf,
11322             "\\markdownRendererFencedCodeAttributeContextEnd{")
11323     end
11324     return buf
11325 end
11326

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

11327     if allow_raw_blocks then
11328         function self.rawBlock(s, attr)
11329             if not self.is_writing then return "" end
11330             s = s:gsub("\n$", "")
11331             local name = util.cache_verbatim(options.cacheDir, s)
11332             return {"\\markdownRendererInputRawBlock{" ,
11333                 name,"}{" , self.string(attr),"}"}
11334         end
11335     end
11336 end, extend_reader = function(self)
11337     local parsers = self.parsers
11338     local writer = self.writer
11339
11340     local function captures_geq_length(_,i,a,b)
11341         return #a >= #b and i
11342     end
11343
11344     local function strip_enclosing_whitespaces(str)
11345         return str:gsub("^%s*(.)%s*$", "%1")
11346     end
11347
11348     local tilde_infostring = Cs(Cs((V("HtmlEntity")
11349         + parsers.anyescaped
11350         - parsers.newline)^0)
11351         / strip_enclosing_whitespaces)
11352
11353     local backtick_infostring
11354     = Cs( Cs((V("HtmlEntity")
11355         + ( -#(parsers.backslash * parsers.backtick)
11356         * parsers.anyescaped)
11357         - parsers.newline
11358         - parsers.backtick)^0)
11359         / strip_enclosing_whitespaces)
11360

```

```

11361     local fenceindent
11362
11363     local function has_trail(indent_table)
11364         return indent_table ~= nil and
11365             indent_table.trail ~= nil and
11366             next(indent_table.trail) ~= nil
11367     end
11368
11369     local function has_indents(indent_table)
11370         return indent_table ~= nil and
11371             indent_table.indents ~= nil and
11372             next(indent_table.indents) ~= nil
11373     end
11374
11375     local function get_last_indent_name(indent_table)
11376         if has_indents(indent_table) then
11377             return indent_table.indents[#indent_table.indents].name
11378         end
11379     end
11380
11381     local count_fenced_start_indent =
11382     function(_, _, indent_table, trail)
11383         local last_indent_name = get_last_indent_name(indent_table)
11384         fenceindent = 0
11385         if last_indent_name ~= "li" then
11386             fenceindent = #trail
11387         end
11388         return true
11389     end
11390
11391     local fencehead = function(char, infostring)
11392         return Cmt( Cb("indent_info")
11393             * parsers.check_trail, count_fenced_start_indent)
11394             * Cg(char^3, "fencelength")
11395             * parsers.optionalspace
11396             * infostring
11397             * (parsers.newline + parsers.eof)
11398     end
11399
11400     local fencetail = function(char)
11401         return parsers.check_trail_no_rem
11402             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11403             * parsers.optionalspace * (parsers.newline + parsers.eof)
11404             + parsers.eof
11405     end
11406
11407     local process_fenced_line =

```

```

11408     function(s, i, -- luacheck: ignore s i
11409             indent_table, line_content, is_blank)
11410         local remainder = ""
11411         if has_trail(indent_table) then
11412             remainder = indent_table.trail.internal_remainder
11413         end
11414
11415         if is_blank
11416             and get_last_indent_name(indent_table) == "li" then
11417             remainder = ""
11418         end
11419
11420         local str = remainder .. line_content
11421         local index = 1
11422         local remaining = fenceindent
11423
11424         while true do
11425             local c = str:sub(index, index)
11426             if c == " " and remaining > 0 then
11427                 remaining = remaining - 1
11428                 index = index + 1
11429             elseif c == "\t" and remaining > 3 then
11430                 remaining = remaining - 4
11431                 index = index + 1
11432             else
11433                 break
11434             end
11435         end
11436
11437         return true, str:sub(index)
11438     end
11439
11440     local fencedline = function(char)
11441         return Cmt( Cb("indent_info")
11442                   * C(parsers.line - fencetail(char))
11443                   * Cc(false), process_fenced_line)
11444     end
11445
11446     local blankfencedline
11447         = Cmt( Cb("indent_info")
11448              * C(parsers.blankline)
11449              * Cc(true), process_fenced_line)
11450
11451     local TildeFencedCode
11452         = fencehead(parsers.tilde, tilde_infostring)
11453         * Cs(( parsers.check_minimal_blank_indent / ""))
11454             * blankfencedline

```

```

11455         + ( parsers.check_minimal_indent / "" )
11456         * fencedline(parsers.tilde)^0)
11457     * ( (parsers.check_minimal_indent / "" )
11458     * fencetail(parsers.tilde) + parsers.succeed)
11459
11460     local BacktickFencedCode
11461         = fencehead(parsers.backtick, backtick_infostring)
11462         * Cs(( (parsers.check_minimal_blank_indent / "" )
11463         * blankfencedline
11464         + (parsers.check_minimal_indent / "" )
11465         * fencedline(parsers.backtick))^0)
11466         * ( (parsers.check_minimal_indent / "" )
11467         * fencetail(parsers.backtick) + parsers.succeed)
11468
11469     local infostring_with_attributes
11470         = Ct(C((parsers.linechar
11471         - ( parsers.optionalspace
11472         * parsers.attributes))^0)
11473         * parsers.optionalspace
11474         * Ct(parsers.attributes))
11475
11476     local FencedCode
11477         = ((TildeFencedCode + BacktickFencedCode)
11478         / function(infostring, code)
11479         local expanded_code = self.expandtabs(code)
11480
11481         if allow_raw_blocks then
11482             local raw_attr = lpeg.match(parsers.raw_attribute,
11483             infostring)
11484             if raw_attr then
11485                 return writer.rawBlock(expanded_code, raw_attr)
11486             end
11487         end
11488
11489         local attr = nil
11490         if allow_attributes then
11491             local match = lpeg.match(infostring_with_attributes,
11492             infostring)
11493             if match then
11494                 infostring, attr = table.unpack(match)
11495             end
11496         end
11497         return writer.fencedCode(expanded_code, infostring, attr)
11498     end)
11499
11500     self.insert_pattern("Block after Verbatim",
11501         FencedCode, "FencedCode")

```

```

11502
11503     local fencestart
11504     if blank_before_code_fence then
11505         fencestart = parsers.fail
11506     else
11507         fencestart = fencehead(parsers.backtick, backtick_infostring)
11508                     + fencehead(parsers.tilde, tilde_infostring)
11509     end
11510
11511     self.update_rule("EndlineExceptions", function(previous_pattern)
11512         if previous_pattern == nil then
11513             previous_pattern = parsers.EndlineExceptions
11514         end
11515         return previous_pattern + fencestart
11516     end)
11517
11518     self.add_special_character("`")
11519     self.add_special_character("~")
11520 end
11521 }
11522 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.

```

11523 M.extensions.fenced_divs = function(blank_before_div_fence)
11524     return {
11525         name = "built-in fenced_divs syntax extension",
11526         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.

```

11527         function self.div_begin(attributes)
11528             local start_output
11529             = {"\\markdownRendererFencedDivAttributeContextBegin\n",
11530              self.attributes(attributes)}
11531             local end_output
11532             = {"\\markdownRendererFencedDivAttributeContextEnd{}}"}
11533             return self.push_attributes(
11534                 "div", attributes, start_output, end_output)
11535         end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

11536     function self.div_end()
11537         return self.pop_attributes("div")
11538     end
11539 end, extend_reader = function(self)
11540     local parsers = self.parsers
11541     local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

11542     local fenced_div_infostring
11543         = C((parsers.linechar
11544             - ( parsers.spacechar^1
11545               * parsers.colon^1))^1)
11546
11547     local fenced_div_begin = parsers.nonindentspace
11548         * parsers.colon^3
11549         * parsers.optionalspace
11550         * fenced_div_infostring
11551         * ( parsers.spacechar^1
11552           * parsers.colon^1)^0
11553         * parsers.optionalspace
11554         * (parsers.newline + parsers.eof)
11555
11556     local fenced_div_end = parsers.nonindentspace
11557         * parsers.colon^3
11558         * parsers.optionalspace
11559         * (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.

```

11560     self.initialize_named_group("fenced_div_level", "0")
11561     self.initialize_named_group("fenced_div_num_opening_indents")
11562
11563     local function increment_div_level()
11564         local push_indent_table =
11565             function(s, i, indent_table, -- luacheck: ignore s i
11566                 fenced_div_num_opening_indents, fenced_div_level)
11567                 fenced_div_level = tonumber(fenced_div_level) + 1
11568                 local num_opening_indents = 0
11569                 if indent_table.indents ~= nil then
11570                     num_opening_indents = #indent_table.indents
11571                 end
11572                 fenced_div_num_opening_indents[fenced_div_level]
11573                 = num_opening_indents

```

```

11574         return true, fenced_div_num_opening_indents
11575     end
11576
11577     local increment_level =
11578         function(s, i, fenced_div_level) -- luacheck: ignore s i
11579             fenced_div_level = tonumber(fenced_div_level) + 1
11580             return true, tostring(fenced_div_level)
11581         end
11582
11583     return Cg( Cmt( Cb("indent_info")
11584                 * Cb("fenced_div_num_opening_indents")
11585                 * Cb("fenced_div_level"), push_indent_table)
11586             , "fenced_div_num_opening_indents")
11587         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
11588             , "fenced_div_level")
11589     end
11590
11591     local function decrement_div_level()
11592         local pop_indent_table =
11593             function(s, i, -- luacheck: ignore s i
11594                 fenced_div_indent_table, fenced_div_level)
11595                 fenced_div_level = tonumber(fenced_div_level)
11596                 fenced_div_indent_table[fenced_div_level] = nil
11597                 return true, tostring(fenced_div_level - 1)
11598             end
11599
11600     return Cg( Cmt( Cb("fenced_div_num_opening_indents")
11601                 * Cb("fenced_div_level"), pop_indent_table)
11602             , "fenced_div_level")
11603     end
11604
11605
11606     local non_fenced_div_block
11607         = parsers.check_minimal_indent * V("Block")
11608         - parsers.check_minimal_indent_and_trail * fenced_div_end
11609
11610     local non_fenced_div_paragraph
11611         = parsers.check_minimal_indent * V("Paragraph")
11612         - parsers.check_minimal_indent_and_trail * fenced_div_end
11613
11614     local blank = parsers.minimally_indented_blank
11615
11616     local block_separated = parsers.block_sep_group(blank)
11617         * non_fenced_div_block
11618
11619     local loop_body_pair
11620         = parsers.create_loop_body_pair(block_separated,

```

```

11621             non_fenced_div_paragraph,
11622             parsers.block_sep_group(blank),
11623             parsers.par_sep_group(blank))
11624
11625     local content_loop = ( non_fenced_div_block
11626                         * loop_body_pair.block^0
11627                         + non_fenced_div_paragraph
11628                         * block_separated
11629                         * loop_body_pair.block^0
11630                         + non_fenced_div_paragraph
11631                         * loop_body_pair.par^0)
11632     * blank^0
11633
11634     local FencedDiv = fenced_div_begin
11635                     / function (infostring)
11636                         local attr
11637                         = lpeg.match(Ct(parsers.attributes),
11638                                     infostring)
11639                         if attr == nil then
11640                             attr = {"." .. infostring}
11641                         end
11642                         return attr
11643                     end
11644     / writer.div_begin
11645     * increment_div_level()
11646     * parsers.skipblanklines
11647     * Ct(content_loop)
11648     * parsers.minimally_indented_blank^0
11649     * parsers.check_minimal_indent_and_trail
11650     * fenced_div_end
11651     * decrement_div_level()
11652     * (Cc("") / writer.div_end)
11653
11654     self.insert_pattern("Block after Verbatim",
11655                       FencedDiv, "FencedDiv")
11656
11657     self.add_special_character(":")
11658

```

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.

```

11659     local function is_inside_div()
11660         local check_div_level =
11661             function(s, i, fenced_div_level) -- luacheck: ignore s i
11662                 fenced_div_level = tonumber(fenced_div_level)
11663                 return fenced_div_level > 0

```



```

11664         end
11665
11666         return Cmt(Cb("fenced_div_level"), check_div_level)
11667     end
11668
11669     local function check_indent()
11670         local compare_indent =
11671             function(s, i, indent_table, -- luacheck: ignore s i
11672                 fenced_div_num_opening_indents, fenced_div_level)
11673                 fenced_div_level = tonumber(fenced_div_level)
11674                 local num_current_indents
11675                     = ( indent_table.current_line_indents ~= nil and
11676                       #indent_table.current_line_indents) or 0
11677                 local num_opening_indents
11678                     = fenced_div_num_opening_indents[fenced_div_level]
11679                 return num_current_indents == num_opening_indents
11680             end
11681
11682         return Cmt( Cb("indent_info")
11683                   * Cb("fenced_div_num_opening_indents")
11684                   * Cb("fenced_div_level"), compare_indent)
11685     end
11686
11687     local fencestart = is_inside_div()
11688                       * fenced_div_end
11689                       * check_indent()
11690
11691     if not blank_before_div_fence then
11692         self.update_rule("EndlineExceptions", function(previous_pattern)
11693             if previous_pattern == nil then
11694                 previous_pattern = parsers.EndlineExceptions
11695             end
11696             return previous_pattern + fencestart
11697         end)
11698     end
11699 end
11700 }
11701 end

```

3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11702 M.extensions.header_attributes = function()
11703     return {
11704         name = "built-in header_attributes syntax extension",
11705         extend_writer = function()

```

```

11706 end, extend_reader = function(self)
11707     local parsers = self.parsers
11708     local writer = self.writer
11709
11710     local function strip_atx_end(s)
11711         return s:gsub("%s+##%s*$", "")
11712     end
11713
11714     local AtxHeading = Cg(parsers.heading_start, "level")
11715         * parsers.optionalspace
11716         * (C(((parsers.linechar
11717             - (parsers.attributes
11718                 * parsers.optionalspace
11719                 * parsers.newline))
11720             * (parsers.linechar
11721                 - parsers.lbrace)^0)^1)
11722             / strip_atx_end
11723             / parsers.parse_heading_text)
11724         * Cg(Ct(parsers.newline
11725             + (parsers.attributes
11726                 * parsers.optionalspace
11727                 * parsers.newline)), "attributes")
11728         * Cb("level")
11729         * Cb("attributes")
11730         / writer.heading
11731
11732     local function strip_trailing_spaces(s)
11733         return s:gsub("%s*$", "")
11734     end
11735
11736     local heading_line = (parsers.linechar
11737         - (parsers.attributes
11738             * parsers.optionalspace
11739             * parsers.newline))^1
11740         - parsers.thematic_break_lines
11741
11742     local heading_text
11743         = heading_line
11744         * ( (V("Endline") / "\n")
11745             * (heading_line - parsers.heading_level))^0
11746         * parsers.newline^-1
11747
11748     local SetextHeading
11749         = parsers.freeze_trail * parsers.check_trail_no_rem
11750         * #(heading_text
11751             * (parsers.attributes
11752                 * parsers.optionalspace

```

```

11753         * parsers.newline)^-1
11754         * parsers.check_minimal_indent
11755         * parsers.check_trail
11756         * parsers.heading_level)
11757     * Cs(heading_text) / strip_trailing_spaces
11758     / parsers.parse_heading_text
11759     * Cg(Ct((parsers.attributes
11760         * parsers.optionalspace
11761         * parsers.newline)^-1), "attributes")
11762     * parsers.check_minimal_indent_and_trail * parsers.heading_level
11763     * Cb("attributes")
11764     * parsers.newline
11765     * parsers.unfreeze_trail
11766     / writer.heading
11767
11768     local Heading = AtxHeading + SetextHeading
11769     self.update_rule("Heading", Heading)
11770 end
11771 }
11772 end

```

3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

11773 M.extensions.inline_code_attributes = function()
11774   return {
11775     name = "built-in inline_code_attributes syntax extension",
11776     extend_writer = function()
11777     end, extend_reader = function(self)
11778       local writer = self.writer
11779
11780       local CodeWithAttributes = parsers.inticks
11781         * Ct(parsers.attributes)
11782         / writer.code
11783
11784       self.insert_pattern("Inline before Code",
11785         CodeWithAttributes,
11786         "CodeWithAttributes")
11787     end
11788   }
11789 end

```

3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

11790 M.extensions.line_blocks = function()
11791   return {
11792     name = "built-in line_blocks syntax extension",
11793     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.

```

11794     function self.lineblock(lines)
11795       if not self.is_writing then return "" end
11796       local buffer = {}
11797       for i = 1, #lines - 1 do
11798         buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11799       end
11800       buffer[#buffer + 1] = lines[#lines]
11801
11802       return {"\\markdownRendererLineBlockBegin\n"
11803             ,buffer,
11804             "\n\\markdownRendererLineBlockEnd "}
11805     end
11806   end, extend_reader = function(self)
11807     local parsers = self.parsers
11808     local writer = self.writer
11809
11810     local LineBlock
11811     = Ct((Cs(( parsers.pipe * parsers.space) / ""
11812             * ((parsers.space)/entities.char_entity("nbsp"))^0
11813             * parsers.linechar^0 * (parsers.newline/""))
11814             * (-parsers.pipe
11815             * (parsers.space^1/" ")
11816             * parsers.linechar^1
11817             * (parsers.newline/"")
11818             )^0
11819             * (parsers.blankline/"")^0)
11820           / self.parser_functions.parse_inlines)^1)
11821     / writer.lineblock
11822
11823     self.insert_pattern("Block after Blockquote",
11824                       LineBlock, "LineBlock")
11825   end
11826 }
11827 end

```

3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11828 M.extensions.mark = function()
11829   return {
11830     name = "built-in mark syntax extension",

```

```

11831     extend_writer = function(self)
Define writer->mark as a function that will transform an input marked text s to
the output format.
11832         function self.mark(s)
11833             if self.flatten_inlines then return s end
11834             return {"\\markdownRendererMark{" , s, "}" }
11835         end
11836     end, extend_reader = function(self)
11837         local parsers = self.parsers
11838         local writer = self.writer
11839
11840         local doubleequals = P("==")
11841
11842         local Mark
11843             = parsers.between(V("Inline"), doubleequals, doubleequals)
11844             / function (inlines) return writer.mark(inlines) end
11845
11846         self.add_special_character("=")
11847         self.insert_pattern("Inline before LinkAndEmph",
11848                             Mark, "Mark")
11849     end
11850 }
11851 end

```

3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

11852 M.extensions.link_attributes = function()
11853     return {
11854         name = "built-in link_attributes syntax extension",
11855         extend_writer = function()
11856             end, extend_reader = function(self)
11857                 local parsers = self.parsers
11858                 local options = self.options
11859

```

The following patterns define link reference definitions with attributes.

```

11860     local define_reference_parser
11861         = (parsers.check_trail / "")
11862         * parsers.link_label
11863         * parsers.colon
11864         * parsers.spnlc * parsers.url
11865         * ( parsers.spnlc_sep * parsers.title
11866           * (parsers.spnlc * Ct(parsers.attributes))
11867           * parsers.only_blank
11868         + parsers.spnlc_sep * parsers.title * parsers.only_blank

```

```

11869         + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11870         * parsers.only_blank
11871         + Cc("") * parsers.only_blank)
11872
11873     local ReferenceWithAttributes = define_reference_parser
11874                                   / self.register_link
11875
11876     self.update_rule("Reference", ReferenceWithAttributes)
11877

```

The following patterns define direct and indirect links with attributes.

```

11878
11879     local LinkWithAttributesAndEmph
11880         = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11881         "match_link_attributes"))
11882         / self.defer_link_and_emphasis_processing
11883
11884     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11885

```

The following patterns define autolinks with attributes.

```

11886     local AutoLinkUrlWithAttributes
11887         = parsers.auto_link_url
11888         * Ct(parsers.attributes)
11889         / self.auto_link_url
11890
11891     self.insert_pattern("Inline before AutoLinkUrl",
11892                       AutoLinkUrlWithAttributes,
11893                       "AutoLinkUrlWithAttributes")
11894
11895     local AutoLinkEmailWithAttributes
11896         = parsers.auto_link_email
11897         * Ct(parsers.attributes)
11898         / self.auto_link_email
11899
11900     self.insert_pattern("Inline before AutoLinkEmail",
11901                       AutoLinkEmailWithAttributes,
11902                       "AutoLinkEmailWithAttributes")
11903
11904     if options.relativeReferences then
11905
11906         local AutoLinkRelativeReferenceWithAttributes
11907             = parsers.auto_link_relative_reference
11908             * Ct(parsers.attributes)
11909             / self.auto_link_url
11910
11911         self.insert_pattern(
11912             "Inline before AutoLinkRelativeReference",

```

```

11913         AutoLinkRelativeReferenceWithAttributes,
11914         "AutoLinkRelativeReferenceWithAttributes")
11915
11916     end
11917
11918 end
11919 }
11920 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.

```

11921 M.extensions.notes = function(notes, inline_notes)
11922   assert(notes or inline_notes)
11923   return {
11924     name = "built-in notes syntax extension",
11925     extend_writer = function(self)

```

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

```

11926     function self.note(s)
11927       if self.flatten_inlines then return "" end
11928       return {"\\markdownRendererNote{",s,"}"}
11929     end
11930 end, extend_reader = function(self)
11931   local parsers = self.parsers
11932   local writer = self.writer
11933
11934   local rawnotes = parsers.rawnotes
11935
11936   if inline_notes then
11937     local InlineNote
11938     = parsers.circumflex
11939     * ( parsers.link_label
11940       / self.parser_functions.parse_inlines_no_inline_note)
11941     / writer.note
11942
11943     self.insert_pattern("Inline after LinkAndEmph",
11944                       InlineNote, "InlineNote")
11945   end
11946   if notes then
11947     local function strip_first_char(s)
11948       return s:sub(2)
11949     end

```

```

11950
11951     local RawNoteRef
11952         = #(parsers.lbracket * parsers.circumflex)
11953         * parsers.link_label / strip_first_char
11954
11955     -- like indirect_link
11956     local function lookup_note(ref)
11957         return writer.defer_call(function()
11958             local found = rawnotes[self.normalize_tag(ref)]
11959             if found then
11960                 return writer.note(
11961                     self.parser_functions.parse_blocks_nested(found))
11962             else
11963                 return {"[",
11964                     self.parser_functions.parse_inlines("^" .. ref), "]" }
11965             end
11966         end)
11967     end
11968
11969     local function register_note(ref,rawnote)
11970         local normalized_tag = self.normalize_tag(ref)
11971         if rawnotes[normalized_tag] == nil then
11972             rawnotes[normalized_tag] = rawnote
11973         end
11974         return ""
11975     end
11976
11977     local NoteRef = RawNoteRef / lookup_note
11978
11979     local optionally_indented_line
11980         = parsers.check_optional_indent_and_any_trail * parsers.line
11981
11982     local blank
11983         = parsers.check_optional_blank_indent_and_any_trail
11984         * parsers.optionalspace * parsers.newline
11985
11986     local chunk
11987         = Cs(parsers.line
11988             * (optionally_indented_line - blank)^0)
11989
11990     local indented_blocks = function(bl)
11991         return Cs( bl
11992             * ( blank^1 * (parsers.check_optional_indent / "")
11993             * parsers.check_code_trail
11994             * -parsers.blankline * bl)^0)
11995     end
11996

```



```

11997     local NoteBlock
11998         = parsers.check_trail_no_rem
11999         * RawNoteRef * parsers.colon
12000         * parsers.spnlc * indented_blocks(chunk)
12001         / register_note
12002
12003     local Reference = NoteBlock + parsers.Reference
12004
12005     self.update_rule("Reference", Reference)
12006     self.insert_pattern("Inline before LinkAndEmph",
12007         NoteRef, "NoteRef")
12008 end
12009
12010     self.add_special_character("^")
12011 end
12012 }
12013 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.

```

12014 M.extensions.pipe_tables = function(table_captions, table_attributes)
12015
12016     local function make_pipe_table_rectangular(rows)
12017         local num_columns = #rows[2]
12018         local rectangular_rows = {}
12019         for i = 1, #rows do
12020             local row = rows[i]
12021             local rectangular_row = {}
12022             for j = 1, num_columns do
12023                 rectangular_row[j] = row[j] or ""
12024             end
12025             table.insert(rectangular_rows, rectangular_row)
12026         end
12027         return rectangular_rows
12028     end
12029
12030     local function pipe_table_row(allow_empty_first_column
12031         , nonempty_column
12032         , column_separator
12033         , column)
12034         local row_beginning

```

```

12035     if allow_empty_first_column then
12036         row_beginning = -- empty first column
12037             #(parsers.spacechar^4
12038                 * column_separator)
12039             * parsers.optionalspace
12040             * column
12041             * parsers.optionalspace
12042         -- non-empty first column
12043         + parsers.nonindentSPACE
12044         * nonempty_column^-1
12045         * parsers.optionalspace
12046     else
12047         row_beginning = parsers.nonindentSPACE
12048             * nonempty_column^-1
12049             * parsers.optionalspace
12050     end
12051
12052     return Ct(row_beginning
12053         * (-- single column with no leading pipes
12054             #(column_separator
12055                 * parsers.optionalspace
12056                 * parsers.newline)
12057             * column_separator
12058             * parsers.optionalspace
12059             -- single column with leading pipes or
12060             -- more than a single column
12061             + (column_separator
12062                 * parsers.optionalspace
12063                 * column
12064                 * parsers.optionalspace)^1
12065             * (column_separator
12066                 * parsers.optionalspace)^-1))
12067 end
12068
12069 return {
12070     name = "built-in pipe_tables syntax extension",
12071     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.

```

12072     function self.table(rows, caption, attributes)
12073         if not self.is_writing then return "" end
12074         local buffer = {}
12075         if attributes ~= nil then
12076             table.insert(buffer,
12077                 "\\markdownRendererTableAttributeContextBegin\n")
12078             table.insert(buffer, self.attributes(attributes))

```

```

12079     end
12080     table.insert(buffer,
12081                 {"\\markdownRendererTable{" ,
12082                 caption or "", "}" , #rows - 1, "}" ,
12083                 #rows[1], "}")
12084     local temp = rows[2] -- put alignments on the first row
12085     rows[2] = rows[1]
12086     rows[1] = temp
12087     for i, row in ipairs(rows) do
12088         table.insert(buffer, "{")
12089         for _, column in ipairs(row) do
12090             if i > 1 then -- do not use braces for alignments
12091                 table.insert(buffer, "{")
12092             end
12093             table.insert(buffer, column)
12094             if i > 1 then
12095                 table.insert(buffer, "}")
12096             end
12097         end
12098         table.insert(buffer, "}")
12099     end
12100     if attributes ~= nil then
12101         table.insert(buffer,
12102                     "\\markdownRendererTableAttributeContextEnd{")
12103     end
12104     return buffer
12105 end
12106 end, extend_reader = function(self)
12107     local parsers = self.parsers
12108     local writer = self.writer
12109
12110     local table_hline_separator = parsers.pipe + parsers.plus
12111
12112     local table_hline_column = (parsers.dash
12113                                - #(parsers.dash
12114                                   * (parsers.spacechar
12115                                      + table_hline_separator
12116                                      + parsers.newline)))^1
12117                                * (parsers.colon * Cc("r")
12118                                   + parsers.dash * Cc("d"))
12119                                + parsers.colon
12120                                * (parsers.dash
12121                                   - #(parsers.dash
12122                                      * (parsers.spacechar
12123                                         + table_hline_separator
12124                                         + parsers.newline)))^1
12125                                * (parsers.colon * Cc("c"))

```

```

12126             + parsers.dash * Cc("l"))
12127
12128     local table_hline = pipe_table_row(false
12129                                     , table_hline_column
12130                                     , table_hline_separator
12131                                     , table_hline_column)
12132
12133     local table_caption_beginning
12134     = ( parsers.check_minimal_blank_indent_and_any_trail_no_rem
12135       * parsers.optionalspace * parsers.newline)^0
12136       * parsers.check_minimal_indent_and_trail
12137       * (P("Table")^-1 * parsers.colon)
12138       * parsers.optionalspace
12139
12140     local function strip_trailing_spaces(s)
12141       return s:gsub("%s*$","")
12142     end
12143
12144     local table_row
12145     = pipe_table_row(true
12146                     , (C((parsers.linechar - parsers.pipe)^1)
12147                       / strip_trailing_spaces
12148                       / self.parser_functions.parse_inlines)
12149                     , parsers.pipe
12150                     , (C((parsers.linechar - parsers.pipe)^0)
12151                       / strip_trailing_spaces
12152                       / self.parser_functions.parse_inlines))
12153
12154     local table_caption
12155     if table_captions then
12156       table_caption = #table_caption_beginning
12157                     * table_caption_beginning
12158       if table_attributes then
12159         table_caption = table_caption
12160                       * (C(((( parsers.linechar
12161                               - (parsers.attributes
12162                                 * parsers.optionalspace
12163                                 * parsers.newline
12164                                 * -( parsers.optionalspace
12165                                   * parsers.linechar))))
12166                          + ( parsers.newline
12167                            * #( parsers.optionalspace
12168                              * parsers.linechar)
12169                              * C(parsers.optionalspace)
12170                              / writer.space))
12171                          * (parsers.linechar
12172                            - parsers.lbrace)^0)^1)

```

```

12173         / self.parser_functions.parse_inlines)
12174     * (parsers.newline
12175     + ( Ct(parsers.attributes)
12176     * parsers.optionalspace
12177     * parsers.newline))
12178     else
12179         table_caption = table_caption
12180         * C(( parsers.linechar
12181         + ( parsers.newline
12182         * #( parsers.optionalspace
12183         * parsers.linechar)
12184         * C(parsers.optionalspace)
12185         / writer.space))^1)
12186         / self.parser_functions.parse_inlines
12187         * parsers.newline
12188     end
12189     else
12190         table_caption = parsers.fail
12191     end
12192
12193     local PipeTable
12194     = Ct( table_row * parsers.newline
12195     * (parsers.check_minimal_indent_and_trail / {})
12196     * table_hline * parsers.newline
12197     * ( (parsers.check_minimal_indent / {})
12198     * table_row * parsers.newline)^0)
12199     / make_pipe_table_rectangular
12200     * table_caption^-1
12201     / writer.table
12202
12203     self.insert_pattern("Block after Blockquote",
12204         PipeTable, "PipeTable")
12205     end
12206 }
12207 end

```

3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

12208 M.extensions.raw_inline = function()
12209     return {
12210         name = "built-in raw_inline syntax extension",
12211         extend_writer = function(self)
12212             local options = self.options
12213

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```
12214     function self.rawInline(s, attr)
12215         if not self.is_writing then return "" end
12216         if self.flatten_inlines then return s end
12217         local name = util.cache_verbatim(options.cacheDir, s)
12218         return {"\\markdownRendererInputRawInline{" ,
12219             name,"}{" , self.string(attr),""}
12220     end
12221 end, extend_reader = function(self)
12222     local writer = self.writer
12223
12224     local RawInline = parsers.inticks
12225         * parsers.raw_attribute
12226         / writer.rawInline
12227
12228     self.insert_pattern("Inline before Code",
12229         RawInline, "RawInline")
12230 end
12231 }
12232 end
```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
12233 M.extensions.strike_through = function()
12234     return {
12235         name = "built-in strike_through syntax extension",
12236         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.

```
12237     function self.strike_through(s)
12238         if self.flatten_inlines then return s end
12239         return {"\\markdownRendererStrikeThrough{" ,s,""}
12240     end
12241 end, extend_reader = function(self)
12242     local parsers = self.parsers
12243     local writer = self.writer
12244
12245     local StrikeThrough = (
12246         parsers.between(parsers.Inline, parsers.doubletildes,
12247             parsers.doubletildes)
12248     ) / writer.strike_through
12249
12250     self.insert_pattern("Inline after LinkAndEmph",
```

```

12251             StrikeThrough, "StrikeThrough")
12252
12253         self.add_special_character("~")
12254     end
12255 }
12256 end

```

3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

12257 M.extensions.subscripts = function()
12258   return {
12259     name = "built-in subscripts syntax extension",
12260     extend_writer = function(self)

```

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

```

12261         function self.subscript(s)
12262           if self.flatten_inlines then return s end
12263           return {"\\markdownRendererSubscript{" ,s,"}"}
12264         end
12265     end, extend_reader = function(self)
12266       local parsers = self.parsers
12267       local writer = self.writer
12268
12269       local Subscript = (
12270         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
12271       ) / writer.subscript
12272
12273       self.insert_pattern("Inline after LinkAndEmph",
12274         Subscript, "Subscript")
12275
12276       self.add_special_character("~")
12277     end
12278   }
12279 end

```

3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

12280 M.extensions.superscripts = function()
12281   return {
12282     name = "built-in superscripts syntax extension",
12283     extend_writer = function(self)

```

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

```
12284     function self.superscript(s)
12285         if self.flatten_inlines then return s end
12286         return {"\\markdownRendererSuperscript{" ,s,"}"}
12287     end
12288 end, extend_reader = function(self)
12289     local parsers = self.parsers
12290     local writer = self.writer
12291
12292     local Superscript = (
12293         parsers.between(parsers.Str, parsers.circumflex,
12294             parsers.circumflex)
12295     ) / writer.superscript
12296
12297     self.insert_pattern("Inline after LinkAndEmph",
12298         Superscript, "Superscript")
12299
12300     self.add_special_character("^")
12301 end
12302 }
12303 end
```

3.1.7.19 T_EX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
12304 M.extensions.tex_math = function(tex_math_dollars,
12305     tex_math_single_backslash,
12306     tex_math_double_backslash)
12307     return {
12308         name = "built-in tex_math syntax extension",
12309         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.

```
12310     function self.display_math(s)
12311         if self.flatten_inlines then return s end
12312         return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
12313     end
```

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

```
12314     function self.inline_math(s)
12315         if self.flatten_inlines then return s end
12316         return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
12317     end
```



```

12318     end, extend_reader = function(self)
12319         local parsers = self.parsers
12320         local writer = self.writer
12321
12322         local function between(p, starter, ender)
12323             return (starter * Cs(p * (p - ender)^0) * ender)
12324         end
12325
12326         local function strip_preceding_whitespaces(str)
12327             return str:gsub("^%s*(.-)$", "%1")
12328         end
12329
12330         local allowed_before_closing
12331             = B( parsers.backslash * parsers.any
12332                 + parsers.any * (parsers.any - parsers.backslash))
12333
12334         local allowed_before_closing_no_space
12335             = B( parsers.backslash * parsers.any
12336                 + parsers.any * (parsers.nonspacechar - parsers.backslash))
12337

```

The following patterns implement the Pandoc dollar math syntax extension.

```

12338     local dollar_math_content
12339         = (parsers.newline * (parsers.check_optional_indent / "")
12340           + parsers.backslash^-1
12341           * parsers.linechar)
12342         - parsers.blankline^2
12343         - parsers.dollar
12344
12345     local inline_math_opening_dollars = parsers.dollar
12346                                         * #(parsers.nonspacechar)
12347
12348     local inline_math_closing_dollars
12349         = allowed_before_closing_no_space
12350         * parsers.dollar
12351         * -#(parsers.digit)
12352
12353     local inline_math_dollars = between(Cs( dollar_math_content),
12354                                         inline_math_opening_dollars,
12355                                         inline_math_closing_dollars)
12356
12357     local display_math_opening_dollars = parsers.dollar
12358                                         * parsers.dollar
12359
12360     local display_math_closing_dollars = parsers.dollar
12361                                         * parsers.dollar
12362
12363     local display_math_dollars = between(Cs( dollar_math_content),

```

```

12364             display_math_opening_dollars,
12365             display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12366     local backslash_math_content
12367     = (parsers.newline * (parsers.check_optional_indent / ""))
12368     + parsers.linechar)
12369     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12370     local inline_math_opening_double = parsers.backslash
12371     * parsers.backslash
12372     * parsers.lparent
12373
12374     local inline_math_closing_double = allowed_before_closing
12375     * parsers.spacechar^0
12376     * parsers.backslash
12377     * parsers.backslash
12378     * parsers.rparent
12379
12380     local inline_math_double = between(Cs( backslash_math_content),
12381     inline_math_opening_double,
12382     inline_math_closing_double)
12383     / strip_preceding_whitespaces
12384
12385     local display_math_opening_double = parsers.backslash
12386     * parsers.backslash
12387     * parsers.lbracket
12388
12389     local display_math_closing_double = allowed_before_closing
12390     * parsers.spacechar^0
12391     * parsers.backslash
12392     * parsers.backslash
12393     * parsers.rbracket
12394
12395     local display_math_double = between(Cs( backslash_math_content),
12396     display_math_opening_double,
12397     display_math_closing_double)
12398     / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

12399     local inline_math_opening_single = parsers.backslash
12400     * parsers.lparent
12401
12402     local inline_math_closing_single = allowed_before_closing
12403     * parsers.spacechar^0

```

```

12404             * parsers.backslash
12405             * parsers.rparent
12406
12407     local inline_math_single = between(Cs( backslash_math_content),
12408                                     inline_math_opening_single,
12409                                     inline_math_closing_single)
12410                                     / strip_preceding_whitespaces
12411
12412     local display_math_opening_single = parsers.backslash
12413                                     * parsers.lbracket
12414
12415     local display_math_closing_single = allowed_before_closing
12416                                     * parsers.spacechar^0
12417                                     * parsers.backslash
12418                                     * parsers.rbracket
12419
12420     local display_math_single = between(Cs( backslash_math_content),
12421                                     display_math_opening_single,
12422                                     display_math_closing_single)
12423                                     / strip_preceding_whitespaces
12424
12425     local display_math = parsers.fail
12426
12427     local inline_math = parsers.fail
12428
12429     if tex_math_dollars then
12430         display_math = display_math + display_math_dollars
12431         inline_math = inline_math + inline_math_dollars
12432     end
12433
12434     if tex_math_double_backslash then
12435         display_math = display_math + display_math_double
12436         inline_math = inline_math + inline_math_double
12437     end
12438
12439     if tex_math_single_backslash then
12440         display_math = display_math + display_math_single
12441         inline_math = inline_math + inline_math_single
12442     end
12443
12444     local TexMath = display_math / writer.display_math
12445                   + inline_math / writer.inline_math
12446
12447     self.insert_pattern("Inline after LinkAndEmph",
12448                       TexMath, "TexMath")
12449
12450     if tex_math_dollars then

```

```

12451     self.add_special_character("$")
12452   end
12453
12454   if tex_math_single_backslash or tex_math_double_backslash then
12455     self.add_special_character("\\")
12456     self.add_special_character("[")
12457     self.add_special_character("]")
12458     self.add_special_character("(")
12459     self.add_special_character("(")
12460   end
12461 end
12462 }
12463 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.

```

12464 M.extensions.jekyll_data = function(expect_jekyll_data,
12465                                     ensure_jekyll_data)
12466   return {
12467     name = "built-in jekyll_data syntax extension",
12468     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.

```

12469     function self.jekyllData(d, t, p)
12470       if not self.is_writing then return "" end
12471
12472       local buf = {}
12473
12474       local keys = {}
12475       for k, _ in pairs(d) do
12476         table.insert(keys, k)
12477       end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

12478     table.sort(keys, function(first, second)

```

```

12479         if type(first) ~= type(second) then
12480             return type(first) < type(second)
12481         else
12482             return first < second
12483         end
12484     end)
12485
12486     if not p then
12487         table.insert(buf, "\\markdownRendererJekyllDataBegin")
12488     end
12489
12490     local is_sequence = false
12491     if #d > 0 and #d == #keys then
12492         for i=1, #d do
12493             if d[i] == nil then
12494                 goto not_a_sequence
12495             end
12496         end
12497         is_sequence = true
12498     end
12499     ::not_a_sequence::
12500
12501     if is_sequence then
12502         table.insert(buf,
12503             "\\markdownRendererJekyllDataSequenceBegin{")
12504         table.insert(buf, self.identifier(p or "null"))
12505         table.insert(buf, "}{")
12506         table.insert(buf, #keys)
12507         table.insert(buf, "}")
12508     else
12509         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12510         table.insert(buf, self.identifier(p or "null"))
12511         table.insert(buf, "}{")
12512         table.insert(buf, #keys)
12513         table.insert(buf, "}")
12514     end
12515
12516     for _, k in ipairs(keys) do
12517         local v = d[k]
12518         local typ = type(v)
12519         k = tostring(k or "null")
12520         if typ == "table" and next(v) ~= nil then
12521             table.insert(
12522                 buf,
12523                 self.jekyllData(v, t, k)
12524             )
12525         else

```

```

12526         k = self.identifier(k)
12527         v = tostring(v)
12528         if typ == "boolean" then
12529             table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12530             table.insert(buf, k)
12531             table.insert(buf, "}{")
12532             table.insert(buf, v)
12533             table.insert(buf, "}")
12534         elseif typ == "number" then
12535             table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12536             table.insert(buf, k)
12537             table.insert(buf, "}{")
12538             table.insert(buf, v)
12539             table.insert(buf, "}")
12540         elseif typ == "string" then
12541             table.insert(buf,
12542                 "\\markdownRendererJekyllDataProgrammaticString{")
12543             table.insert(buf, k)
12544             table.insert(buf, "}{")
12545             table.insert(buf, self.identifier(v))
12546             table.insert(buf, "}")
12547             table.insert(buf,
12548                 "\\markdownRendererJekyllDataTypographicString{")
12549             table.insert(buf, k)
12550             table.insert(buf, "}{")
12551             table.insert(buf, t(v))
12552             table.insert(buf, "}")
12553         elseif typ == "table" then
12554             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12555             table.insert(buf, k)
12556             table.insert(buf, "}")
12557         else
12558             local error = self.error(format(
12559                 "Unexpected type %s for value of "
12560                 .. "YAML key %s.", typ, k))
12561             table.insert(buf, error)
12562         end
12563     end
12564 end
12565
12566 if is_sequence then
12567     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12568 else
12569     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12570 end
12571
12572 if not p then

```

```

12573         table.insert(buf, "\\markdownRendererJekyllDataEnd")
12574     end
12575
12576     return buf
12577 end
12578 end, extend_reader = function(self)
12579     local parsers = self.parsers
12580     local writer = self.writer
12581
12582     local JekyllData
12583     = Cmt( C((parsers.line - P("----") - P("..."))^0)
12584         , function(s, i, text) -- luacheck: ignore s i
12585             local data
12586             local ran_ok, _ = pcall(function()
12587                 local tinyyaml = require("tinyyaml")
12588                 data = tinyyaml.parse(text, {timestamps=false})
12589             end)
12590             if ran_ok and data ~= nil then
12591                 return true, writer.jekyllData(data, function(s)
12592                     return self.parser_functions.parse_blocks_nested(s)
12593                 end, nil)
12594             else
12595                 return false
12596             end
12597         end
12598     )
12599
12600     local UnexpectedJekyllData
12601     = P("----")
12602     * parsers.blankline / 0
12603     -- if followed by blank, it's thematic break
12604     * #(-parsers.blankline)
12605     * JekyllData
12606     * (P("----") + P("..."))
12607
12608     local ExpectedJekyllData
12609     = ( P("----")
12610         * parsers.blankline / 0
12611         -- if followed by blank, it's thematic break
12612         * #(-parsers.blankline)
12613         )^-1
12614     * JekyllData
12615     * (P("----") + P("..."))^-1
12616
12617 if ensure_jekyll_data then
12618     ExpectedJekyllData = ExpectedJekyllData
12619     * parsers.eof

```

```

12620     else
12621         ExpectedJekyllData = ( ExpectedJekyllData
12622                               * (V("Blank")^0 / writer.interblocksep)
12623                               )^-1
12624     end
12625
12626     self.insert_pattern("Block before Blockquote",
12627                       UnexpectedJekyllData, "UnexpectedJekyllData")
12628     if expect_jekyll_data then
12629         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12630     end
12631 end
12632 }
12633 end

```

3.1.8 Conversion from Markdown to Plain T_EX

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`.

```
12634 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```

12635     options = options or {}
12636     setmetatable(options, { __index = function (_, key)
12637         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.

```

12638     local parser_convert = nil
12639     return function(input)
12640         local function convert(input)
12641             if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

12642             local parser = require("markdown-parser")
12643             if metadata.version ~= parser.metadata.version then
12644                 warn("markdown.lua " .. metadata.version .. " used with " ..
12645                     "markdown-parser.lua " .. parser.metadata.version .. ".")
12646             end
12647             parser_convert = parser.new(options)
12648         end
12649         return parser_convert(input)
12650     end

```


If we cache markdown documents, produce the cache file and transform its filename to plain T_EX 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).

```
12651     local output
12652     if options.eagerCache or options.finalizeCache then
12653         local salt = util.salt(options)
12654         local name = util.cache(options.cacheDir, input, salt, convert,
12655                                 ".md.tex")
12656         output = [[\input{]} .. name .. [{}\relax]]
```

Otherwise, return the result of the conversion directly.

```
12657     else
12658         output = convert(input)
12659     end
```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```
12660     if options.finalizeCache then
12661         local file, mode
12662         if options.frozenCacheCounter > 0 then
12663             mode = "a"
12664         else
12665             mode = "w"
12666         end
12667         file = assert(io.open(options.frozenCacheFileName, mode),
12668                       [[Could not open file ]] .. options.frozenCacheFileName
12669                       .. [{" for writing"}])
12670         assert(file:write(
12671             [[\expandafter\global\expandafter\def\csname ]]
12672             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12673             .. [[\endcsname{]} .. output .. [{}]] .. "\n"))
12674         assert(file:close())
12675     end
12676     return output
12677 end
12678 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_EX output. See Section 2.1.1.

```
12679 function M.new(options)
```

Make the `options` table inherit from the `defaultOptions` table.

```
12680     options = options or {}
12681     setmetatable(options, { __index = function (_, key)
12682         return defaultOptions[key] end })
```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```
12683   if options.singletonCache and singletonCache.convert then
12684     for k, v in pairs(defaultOptions) do
12685       if type(v) == "table" then
12686         for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12687           if singletonCache.options[k][i] ~= options[k][i] then
12688             goto miss
12689           end
12690         end
```

The `cacheDir` option is disregarded.

```
12691       elseif k ~= "cacheDir"
12692         and singletonCache.options[k] ~= options[k] then
12693         goto miss
12694       end
12695     end
12696     return singletonCache.convert
12697   end
12698   ::miss::
```

Apply built-in syntax extensions based on `options`.

```
12699   local extensions = {}
12700
12701   if options.bracketedSpans then
12702     local bracketed_spans_extension = M.extensions.bracketed_spans()
12703     table.insert(extensions, bracketed_spans_extension)
12704   end
12705
12706   if options.contentBlocks then
12707     local content_blocks_extension = M.extensions.content_blocks(
12708       options.contentBlocksLanguageMap)
12709     table.insert(extensions, content_blocks_extension)
12710   end
12711
12712   if options.definitionLists then
12713     local definition_lists_extension = M.extensions.definition_lists(
12714       options.tightLists)
12715     table.insert(extensions, definition_lists_extension)
12716   end
12717
12718   if options.fencedCode then
12719     local fenced_code_extension = M.extensions.fenced_code(
12720       options.blankBeforeCodeFence,
12721       options.fencedCodeAttributes,
12722       options.rawAttribute)
12723     table.insert(extensions, fenced_code_extension)
12724   end
12725
```

```

12726 if options.fencedDivs then
12727   local fenced_div_extension = M.extensions.fenced_divs(
12728     options.blankBeforeDivFence)
12729   table.insert(extensions, fenced_div_extension)
12730 end
12731
12732 if options.headerAttributes then
12733   local header_attributes_extension = M.extensions.header_attributes()
12734   table.insert(extensions, header_attributes_extension)
12735 end
12736
12737 if options.inlineCodeAttributes then
12738   local inline_code_attributes_extension =
12739     M.extensions.inline_code_attributes()
12740   table.insert(extensions, inline_code_attributes_extension)
12741 end
12742
12743 if options.jekyllData then
12744   local jekyll_data_extension = M.extensions.jekyll_data(
12745     options.expectJekyllData, options.ensureJekyllData)
12746   table.insert(extensions, jekyll_data_extension)
12747 end
12748
12749 if options.linkAttributes then
12750   local link_attributes_extension =
12751     M.extensions.link_attributes()
12752   table.insert(extensions, link_attributes_extension)
12753 end
12754
12755 if options.lineBlocks then
12756   local line_block_extension = M.extensions.line_blocks()
12757   table.insert(extensions, line_block_extension)
12758 end
12759
12760 if options.mark then
12761   local mark_extension = M.extensions.mark()
12762   table.insert(extensions, mark_extension)
12763 end
12764
12765 if options.pipeTables then
12766   local pipe_tables_extension = M.extensions.pipe_tables(
12767     options.tableCaptions, options.tableAttributes)
12768   table.insert(extensions, pipe_tables_extension)
12769 end
12770
12771 if options.rawAttribute then
12772   local raw_inline_extension = M.extensions.raw_inline()

```

```

12773     table.insert(extensions, raw_inline_extension)
12774 end
12775
12776 if options.strikeThrough then
12777     local strike_through_extension = M.extensions.strike_through()
12778     table.insert(extensions, strike_through_extension)
12779 end
12780
12781 if options.subscripts then
12782     local subscript_extension = M.extensions.subscripts()
12783     table.insert(extensions, subscript_extension)
12784 end
12785
12786 if options.superscripts then
12787     local superscript_extension = M.extensions.superscripts()
12788     table.insert(extensions, superscript_extension)
12789 end
12790
12791 if options.texMathDollars or
12792     options.texMathSingleBackslash or
12793     options.texMathDoubleBackslash then
12794     local tex_math_extension = M.extensions.tex_math(
12795         options.texMathDollars,
12796         options.texMathSingleBackslash,
12797         options.texMathDoubleBackslash)
12798     table.insert(extensions, tex_math_extension)
12799 end
12800
12801 if options.notes or options.inlineNotes then
12802     local notes_extension = M.extensions.notes(
12803         options.notes, options.inlineNotes)
12804     table.insert(extensions, notes_extension)
12805 end
12806
12807 if options.citations then
12808     local citations_extension
12809         = M.extensions.citations(options.citationNbsps)
12810     table.insert(extensions, citations_extension)
12811 end
12812
12813 if options.fancyLists then
12814     local fancy_lists_extension = M.extensions.fancy_lists()
12815     table.insert(extensions, fancy_lists_extension)
12816 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12817 for _, user_extension_filename in ipairs(options.extensions) do
12818     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```
12819     local pathname = assert(kpse.find_file(filename),
12820         [[Could not locate user-defined syntax extension "]]
12821         .. filename)
12822     local input_file = assert(io.open(pathname, "r"),
12823         [[Could not open user-defined syntax extension "]]
12824         .. pathname .. [{" for reading}])
12825     local input = assert(input_file:read("*a"))
12826     assert(input_file:close())
12827     local user_extension, err = load([[
12828         local sandbox = {}
12829         setmetatable(sandbox, {_index = _G})
12830         _ENV = sandbox
12831     ]]) .. input)()
12832     assert(user_extension,
12833         [[Failed to compile user-defined syntax extension "]]
12834         .. pathname .. [{": }]] .. (err or [{"}]))
```

Then, validate the user-defined syntax extension.

```
12835     assert(user_extension.api_version ~= nil,
12836         [[User-defined syntax extension "]] .. pathname
12837         .. [{" does not specify mandatory field "api_version"}])
12838     assert(type(user_extension.api_version) == "number",
12839         [[User-defined syntax extension "]] .. pathname
12840         .. [{" specifies field "api_version" of type "]]
12841         .. type(user_extension.api_version)
12842         .. [{" but "number" was expected}])
12843     assert(user_extension.api_version > 0
12844         and user_extension.api_version
12845         <= metadata.user_extension_api_version,
12846         [[User-defined syntax extension "]] .. pathname
12847         .. [{" uses syntax extension API version "]]
12848         .. user_extension.api_version .. [{" but markdown.lua "]]
12849         .. metadata.version .. [{" uses API version "]]
12850         .. metadata.user_extension_api_version
12851         .. [{" , which is incompatible}])
12852
12853     assert(user_extension.grammar_version ~= nil,
12854         [[User-defined syntax extension "]] .. pathname
12855         .. [{" does not specify mandatory field "grammar_version"}])
12856     assert(type(user_extension.grammar_version) == "number",
12857         [[User-defined syntax extension "]] .. pathname
12858         .. [{" specifies field "grammar_version" of type "]]
12859         .. type(user_extension.grammar_version)
12860         .. [{" but "number" was expected}])
12861     assert(user_extension.grammar_version == metadata.grammar_version,
12862         [[User-defined syntax extension "]] .. pathname
```

```

12863     .. [" uses grammar version "]
12864     .. user_extension.grammar_version
12865     .. [[ but markdown.lua ]] .. metadata.version
12866     .. [[ uses grammar version ]] .. metadata.grammar_version
12867     .. [[, which is incompatible]])
12868
12869     assert(user_extension.finalize_grammar ~= nil,
12870           [[User-defined syntax extension "]] .. pathname
12871           .. [" does not specify mandatory "finalize_grammar" field]])
12872     assert(type(user_extension.finalize_grammar) == "function",
12873           [[User-defined syntax extension "]] .. pathname
12874           .. [" specifies field "finalize_grammar" of type "]]
12875           .. type(user_extension.finalize_grammar)
12876           .. [" 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.)

```

12877     local extension = {
12878       name = [[user-defined "]] .. pathname .. [" syntax extension]],
12879       extend_reader = user_extension.finalize_grammar,
12880       extend_writer = function() end,
12881     }
12882     return extension
12883   end)(user_extension_filename)
12884   table.insert(extensions, user_extension)
12885 end

```

Produce a conversion function from markdown to plain TeX.

```

12886 local writer = M.writer.new(options)
12887 local reader = M.reader.new(writer, options)
12888 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`.

```

12889 collectgarbage("collect")

```

Update the singleton cache.

```

12890 if options.singletonCache then
12891   local singletonCacheOptions = {}
12892   for k, v in pairs(options) do
12893     singletonCacheOptions[k] = v
12894   end
12895   setmetatable(singletonCacheOptions,
12896     { __index = function (_, key)
12897       return defaultOptions[key] end })
12898   singletonCache.options = singletonCacheOptions
12899   singletonCache.convert = convert
12900 end

```

Return the conversion function from markdown to plain \TeX .

```
12901 return convert
12902 end
12903 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.

```
12904
12905 local input
12906 if input_filename then
12907   local input_file = assert(io.open(input_filename, "r"),
12908     [[Could not open file ]] .. input_filename .. [[ for reading]])
12909   input = assert(input_file:read("*a"))
12910   assert(input_file:close())
12911 else
12912   input = assert(io.read("*a"))
12913 end
12914
```

First, ensure that the `options.cacheDir` directory exists.

```
12915 local lfs = require("lfs")
12916 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12917   assert(lfs.mkdir(options["cacheDir"]))
12918 end
```

If Kpathsea has not been loaded before or if Lua \TeX has not yet been initialized, configure Kpathsea on top of loading it.

```
12919 local kpse
12920 (function()
12921   local should_initialize = package.loaded.kpse == nil
12922     or tex.initialize ~= nil
12923   kpse = require("kpse")
12924   if should_initialize then
12925     kpse.set_program_name("luatex")
12926   end
12927 end)()
12928 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.

```
12929 if metadata.version ~= md.metadata.version then
12930   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12931     "markdown.lua " .. md.metadata.version .. ".")
12932 end
12933 local convert = md.new(options)
```

```

12934 local output = convert(input)
12935
12936 if output_filename then
12937   local output_file = assert(io.open(output_filename, "w"),
12938     [[Could not open file "]] .. output_filename .. [[ for writing]])
12939   assert(output_file:write(output))
12940   assert(output_file:close())
12941 else
12942   assert(io.write(output))
12943 end

```

Remove the `options.cacheDir` directory if it is empty.

```

12944 if options.cacheDir then
12945   lfs.rmdir(options.cacheDir)
12946 end

```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```

12947 \ExplSyntaxOn
12948 \cs_if_free:NT
12949   \markdownInfo
12950   {
12951     \cs_new:Npn
12952       \markdownInfo #1
12953       {
12954         \msg_info:nne
12955           { markdown }
12956           { generic-message }
12957           { #1 }
12958       }
12959   }
12960 \cs_if_free:NT
12961   \markdownWarning
12962   {
12963     \cs_new:Npn
12964       \markdownWarning #1
12965       {
12966         \msg_warning:nne
12967           { markdown }
12968           { generic-message }

```



```

12969         { #1 }
12970     }
12971 }
12972 \cs_if_free:NT
12973 \markdownError
12974 {
12975     \cs_new:Npn
12976         \markdownError #1 #2
12977     {
12978         \msg_error:nnee
12979             { markdown }
12980             { generic-message-with-help-text }
12981             { #1 }
12982             { #2 }
12983     }
12984 }
12985 \msg_new:nnn
12986 { markdown }
12987 { generic-message }
12988 { #1 }
12989 \msg_new:nnnn
12990 { markdown }
12991 { generic-message-with-help-text }
12992 { #1 }
12993 { #2 }
12994 \cs_generate_variant:Nn
12995 \msg_info:nnn
12996 { nne }
12997 \cs_generate_variant:Nn
12998 \msg_warning:nnn
12999 { nne }
13000 \cs_generate_variant:Nn
13001 \msg_error:nnnn
13002 { nnee }
13003 \ExplSyntaxOff

```

3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain T_EX themes provided with the Markdown package.

```

13004 \ExplSyntaxOn
13005 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
13006 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
13007 \cs_new:Nn
13008 \@@_plain_tex_load_theme:nnn
13009 {

```

```

13010 \prop_get:NnNTF
13011   \g_@@_plain_tex_loaded_themes_linenos_prop
13012   { #1 }
13013   \l_tmpa_tl
13014   {
13015     \prop_get:NnN
13016     \g_@@_plain_tex_loaded_themes_versions_prop
13017     { #1 }
13018     \l_tmpb_tl
13019     \str_if_eq:nVTF
13020     { #2 }
13021     \l_tmpb_tl
13022     {
13023       \msg_warning:nnnVn
13024       { markdown }
13025       { repeatedly-loaded-plain-tex-theme }
13026       { #1 }
13027       \l_tmpa_tl
13028       { #2 }
13029     }
13030     {
13031       \msg_error:nnnnVV
13032       { markdown }
13033       { different-versions-of-plain-tex-theme }
13034       { #1 }
13035       { #2 }
13036       \l_tmpb_tl
13037       \l_tmpa_tl
13038     }
13039   }
13040   {
13041     \prop_gput:Nnx
13042     \g_@@_plain_tex_loaded_themes_linenos_prop
13043     { #1 }
13044     { \tex_the:D \tex_inputlineno:D }
13045     \prop_gput:Nnn
13046     \g_@@_plain_tex_loaded_themes_versions_prop
13047     { #1 }
13048     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_plain_tex_built_in_themes_prop` and from the filesystem otherwise.

```

13049   \prop_if_in:NnTF
13050   \g_@@_plain_tex_built_in_themes_prop
13051   { #1 }
13052   {
13053     \msg_info:nnnn

```

```

13054         { markdown }
13055         { loading-built-in-plain-tex-theme }
13056         { #1 }
13057         { #2 }
13058         \prop_item:Nn
13059         \g_@@_plain_tex_built_in_themes_prop
13060         { #1 }
13061     }
13062     {
13063         \msg_info:nnnn
13064         { markdown }
13065         { loading-plain-tex-theme }
13066         { #1 }
13067         { #2 }
13068         \file_input:n
13069         { markdown theme #3 }
13070     }
13071 }
13072 }
13073 \msg_new:nnn
13074 { markdown }
13075 { loading-plain-tex-theme }
13076 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
13077 \msg_new:nnn
13078 { markdown }
13079 { loading-built-in-plain-tex-theme }
13080 { Loading~version~#2~of~built-in-plain-TeX~Markdown~theme~#1 }
13081 \msg_new:nnn
13082 { markdown }
13083 { repeatedly-loaded-plain-tex-theme }
13084 {
13085     Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
13086     loaded~on~line~#2,~not~loading~it~again
13087 }
13088 \msg_new:nnn
13089 { markdown }
13090 { different-versions-of-plain-tex-theme }
13091 {
13092     Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
13093     but~version~#3~has~already~been~loaded~on~line~#4
13094 }
13095 \cs_generate_variant:Nn
13096 \prop_gput:Nnn
13097 { Nnx }
13098 \cs_gset_eq:NN
13099 \@@_load_theme:nnn
13100 \@@_plain_tex_load_theme:nnn

```

```

13101 \cs_generate_variant:Nn
13102   \@@_load_theme:nnn
13103   { VeV }
13104 \cs_generate_variant:Nn
13105   \msg_error:nnnnnn
13106   { nnnnVV }
13107 \cs_generate_variant:Nn
13108   \msg_warning:nnnnn
13109   { nnnVn }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L^AT_EX and ConT_EXt.

```

13110 \cs_new:Npn
13111   \markdownLoadPlainTeXTheme
13112   {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

13113   \tl_set:NV
13114     \l_tmpa_tl
13115     \g_@@_current_theme_tl
13116   \tl_reverse:N
13117     \l_tmpa_tl
13118   \tl_set:Ne
13119     \l_tmpb_tl
13120     {
13121       \tl_tail:V
13122         \l_tmpa_tl
13123     }
13124   \tl_reverse:N
13125     \l_tmpb_tl

```

Next, we munge the theme name.

```

13126   \str_set:NV
13127     \l_tmpa_str
13128     \l_tmpb_tl
13129   \str_replace_all:Nnn
13130     \l_tmpa_str
13131     { / }
13132     { _ }

```

Finally, we load the plain TeX theme.

```

13133   \@@_plain_tex_load_theme:VeV
13134     \l_tmpb_tl
13135     { \markdownThemeVersion }
13136     \l_tmpa_str
13137   }

```

```

13138 \cs_generate_variant:Nn
13139   \tl_set:Nn
13140   { Ne }
13141 \cs_generate_variant:Nn
13142   \@@_plain_tex_load_theme:nnn
13143   { VeV }

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```

13144 \prop_gput:Nnn
13145   \g_@@_plain_tex_built_in_themes_prop
13146   { witiko / dot }
13147   {
13148     \markdownSetup{fencedCode}

```

We store the previous definition of the fenced code token renderer prototype:

```

13149   \cs_set_eq:NN
13150     \@@_dot_previous_definition:nnn
13151     \markdownRendererInputFencedCodePrototype

```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T_EX option is disabled and the code block has not been previously typeset:

```

13152   \regex_const:Nn
13153     \c_@@_dot_infostring_regex
13154     { ^dot(\s+(.+)?)? }
13155   \seq_new:N
13156     \l_@@_dot_matches_seq
13157   \markdownSetup {
13158     rendererPrototypes = {
13159       inputFencedCode = {
13160         \regex_extract_once:NnNTF
13161           \c_@@_dot_infostring_regex
13162           { #2 }
13163         \l_@@_dot_matches_seq
13164         {
13165           \@@_if_option:nF
13166             { frozenCache }
13167             {
13168               \sys_shell_now:n
13169               {
13170                 if~!~test~-e~#1.pdf.source~
13171                 ||~!~diff~#1~#1.pdf.source;
13172                 then~
13173                   dot~-Tpdf~-o~#1.pdf~#1;
13174                   cp~#1~#1.pdf.source;
13175                 fi
13176               }

```

```
13177         }
```

We include the typeset image using the image token renderer:

```
13178         \exp_args:NNne
13179         \exp_last_unbraced:No
13180         \markdownRendererImage
13181         {
13182             { Graphviz~image }
13183             { #1.pdf }
13184             { #1.pdf }
13185         }
13186         {
13187             \seq_item:Nn
13188             \l_@@_dot_matches_seq
13189             { 3 }
13190         }
13191     }
```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```
13192     {
13193         \@@_dot_previous_definition:nnn
13194         { #1 }
13195         { #2 }
13196         { #3 }
13197     }
13198 },
13199 },
13200 }
13201 }
```

We locally change the category code of percent signs, so that we can use them in the shell code:

```
13202 \group_begin:
13203 \char_set_catcode_other:N \%
```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```
13204 \prop_gput:Nnn
13205 \g_@@_plain_tex_built_in_themes_prop
13206 { witiko / graphicx / http }
13207 {
13208     \cs_set_eq:NN
13209     \@@_graphicx_http_previous_definition:nnnn
13210     \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.

```
13211     \int_new:N
```

```

13212     \g_@@_graphicx_http_image_number_int
13213 \int_gset:Nn
13214     \g_@@_graphicx_http_image_number_int
13215     { 0 }
13216 \cs_new:Nn
13217     \@@_graphicx_http_filename:
13218     {
13219     \markdownOptionCacheDir
13220     / witiko_graphicx_http .
13221     \int_use:N
13222     \g_@@_graphicx_http_image_number_int
13223     }

```

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.

```

13224 \cs_new:Nn
13225     \@@_graphicx_http_download:nn
13226     {
13227     wget~--0~#2~#1~
13228     ||~curl~--location~--o~#2~#1~
13229     ||~rm~--f~#2
13230     }

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

13231 \str_new:N
13232     \l_@@_graphicx_http_filename_str
13233 \ior_new:N
13234     \g_@@_graphicx_http_filename_ior
13235 \markdownSetup {
13236     rendererPrototypes = {
13237     image = {
13238     \@@_if_option:nF
13239     { frozenCache }
13240     {

```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```

13241     \sys_shell_now:e
13242     {
13243     mkdir~--p~" \markdownOptionCacheDir ";
13244     if~printf~'%s'~"#3"~|~grep~--q~--E~'\^https?:';
13245     then~

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

13246         OUTPUT_PREFIX=" \markdownOptionCacheDir ";
13247         OUTPUT_BODY="$(printf~'%s'~'#3'
13248             |~md5sum~|~cut~-d'~'~-f1)";
13249         OUTPUT_SUFFIX="$(printf~'%s'~'#3'
13250             |~sed~'s/.*[.]//')";
13251         OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

13252         if~!~[~-e~"$OUTPUT"~];
13253         then~
13254             \@@_graphicx_http_download:nn
13255             { '#3' }
13256             { "$OUTPUT" } ;
13257         printf~'%s'~"$OUTPUT"~
13258             >~" \@@_graphicx_http_filename: ";
13259         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:

```

13260             else~
13261                 printf~'%s'~'#3'~
13262                 >~" \@@_graphicx_http_filename: ";
13263             fi
13264         }
13265     }

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

13266         \ior_open:Ne
13267         \g_@@_graphicx_http_filename_ior
13268         { \@@_graphicx_http_filename: }
13269         \ior_str_get:NN
13270         \g_@@_graphicx_http_filename_ior
13271         \l_@@_graphicx_http_filename_str
13272         \ior_close:N
13273         \g_@@_graphicx_http_filename_ior
13274         \@@_graphicx_http_previous_definition:nnVn
13275         { #1 }
13276         { #2 }
13277         \l_@@_graphicx_http_filename_str
13278         { #4 }
13279         \int_gincr:N
13280         \g_@@_graphicx_http_image_number_int
13281     }
13282 }
13283 }
13284 \cs_generate_variant:Nn
13285     \ior_open:Nn

```



```

13286     { Ne }
13287     \cs_generate_variant:Nn
13288     \@@_graphicx_http_previous_definition:nnnn
13289     { nnVn }
13290   }
13291 \group_end:

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

13292 \prop_gput:Nnn
13293 \g_@@_plain_tex_built_in_themes_prop
13294 { witiko / tilde }
13295 {
13296   \markdownSetup {
13297     rendererPrototypes = {
13298       tilde = {-},
13299     },
13300   }
13301 }
13302 \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.

```

13303 \def\markdownRendererInterblockSeparatorPrototype{\par}%
13304 \def\markdownRendererParagraphSeparatorPrototype{%
13305   \markdownRendererInterblockSeparator}%
13306 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
13307 \def\markdownRendererSoftLineBreakPrototype{ }%
13308 \let\markdownRendererEllipsisPrototype\dots
13309 \def\markdownRendererNbspPrototype{~}%
13310 \def\markdownRendererLeftBracePrototype{\char`\{}%
13311 \def\markdownRendererRightBracePrototype{\char`\}}%
13312 \def\markdownRendererDollarSignPrototype{\char`\$}%
13313 \def\markdownRendererPercentSignPrototype{\char`\}%
13314 \def\markdownRendererAmpersandPrototype{\&}%
13315 \def\markdownRendererUnderscorePrototype{\char`\_}%
13316 \def\markdownRendererHashPrototype{\char`\#}%
13317 \def\markdownRendererCircumflexPrototype{\char`\^}%
13318 \def\markdownRendererBackslashPrototype{\char`\}%
13319 \def\markdownRendererTildePrototype{\char`\~}%
13320 \def\markdownRendererPipePrototype{|}%
13321 \def\markdownRendererCodeSpanPrototype#1{\tt#1}%
13322 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
13323 \def\markdownRendererContentBlockPrototype#1#2#3#4{#

```

```

13324 \markdownInput{#3}}%
13325 \def\markdownRendererContentBlockOnlineImagePrototype{%
13326 \markdownRendererImage}%
13327 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
13328 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
13329 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
13330 \def\markdownRendererUlBeginPrototype{}%
13331 \def\markdownRendererUlBeginTightPrototype{}%
13332 \def\markdownRendererUlItemPrototype{}%
13333 \def\markdownRendererUlItemEndPrototype{}%
13334 \def\markdownRendererUlEndPrototype{}%
13335 \def\markdownRendererUlEndTightPrototype{}%
13336 \def\markdownRendererOlBeginPrototype{}%
13337 \def\markdownRendererOlBeginTightPrototype{}%
13338 \def\markdownRendererFancyOlBeginPrototype#1#2{%
13339 \markdownRendererOlBegin}%
13340 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
13341 \markdownRendererOlBeginTight}%
13342 \def\markdownRendererOlItemPrototype{}%
13343 \def\markdownRendererOlItemWithNumberPrototype#1{}%
13344 \def\markdownRendererOlItemEndPrototype{}%
13345 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
13346 \def\markdownRendererFancyOlItemWithNumberPrototype{%
13347 \markdownRendererOlItemWithNumber}%
13348 \def\markdownRendererFancyOlItemEndPrototype{}%
13349 \def\markdownRendererOlEndPrototype{}%
13350 \def\markdownRendererOlEndTightPrototype{}%
13351 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
13352 \def\markdownRendererFancyOlEndTightPrototype{%
13353 \markdownRendererOlEndTight}%
13354 \def\markdownRendererDlBeginPrototype{}%
13355 \def\markdownRendererDlBeginTightPrototype{}%
13356 \def\markdownRendererDlItemPrototype#1{#1}%
13357 \def\markdownRendererDlItemEndPrototype{}%
13358 \def\markdownRendererDlDefinitionBeginPrototype{}%
13359 \def\markdownRendererDlDefinitionEndPrototype{\par}%
13360 \def\markdownRendererDlEndPrototype{}%
13361 \def\markdownRendererDlEndTightPrototype{}%
13362 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
13363 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
13364 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
13365 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
13366 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
13367 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
13368 \def\markdownRendererInputVerbatimPrototype#1{%
13369 \par{\tt\input#1\relax{}}\par}%
13370 \def\markdownRendererInputFencedCodePrototype#1#2#3{%

```

```

13371 \markdownRendererInputVerbatim{#1}}%
13372 \def\markdownRendererHeadingOnePrototype#1{#1}%
13373 \def\markdownRendererHeadingTwoPrototype#1{#1}%
13374 \def\markdownRendererHeadingThreePrototype#1{#1}%
13375 \def\markdownRendererHeadingFourPrototype#1{#1}%
13376 \def\markdownRendererHeadingFivePrototype#1{#1}%
13377 \def\markdownRendererHeadingSixPrototype#1{#1}%
13378 \def\markdownRendererThematicBreakPrototype{}%
13379 \def\markdownRendererNotePrototype#1{#1}%
13380 \def\markdownRendererCitePrototype#1{}%
13381 \def\markdownRendererTextCitePrototype#1{}%
13382 \def\markdownRendererTickedBoxPrototype{[X]}%
13383 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
13384 \def\markdownRendererUntickedBoxPrototype{[ ]}%
13385 \def\markdownRendererStrikeThroughPrototype#1{#1}%
13386 \def\markdownRendererSuperscriptPrototype#1{#1}%
13387 \def\markdownRendererSubscriptPrototype#1{#1}%
13388 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
13389 \def\markdownRendererInlineMathPrototype#1{$#1$}%
13390 \ExplSyntaxOn
13391 \cs_gset:Npn
13392 \markdownRendererHeaderAttributeContextBeginPrototype
13393 {
13394 \group_begin:
13395 \color_group_begin:
13396 }
13397 \cs_gset:Npn
13398 \markdownRendererHeaderAttributeContextEndPrototype
13399 {
13400 \color_group_end:
13401 \group_end:
13402 }
13403 \cs_gset_eq:NN
13404 \markdownRendererBracketedSpanAttributeContextBeginPrototype
13405 \markdownRendererHeaderAttributeContextBeginPrototype
13406 \cs_gset_eq:NN
13407 \markdownRendererBracketedSpanAttributeContextEndPrototype
13408 \markdownRendererHeaderAttributeContextEndPrototype
13409 \cs_gset_eq:NN
13410 \markdownRendererFencedDivAttributeContextBeginPrototype
13411 \markdownRendererHeaderAttributeContextBeginPrototype
13412 \cs_gset_eq:NN
13413 \markdownRendererFencedDivAttributeContextEndPrototype
13414 \markdownRendererHeaderAttributeContextEndPrototype
13415 \cs_gset_eq:NN
13416 \markdownRendererFencedCodeAttributeContextBeginPrototype
13417 \markdownRendererHeaderAttributeContextBeginPrototype

```

```

13418 \cs_gset_eq:NN
13419 \markdownRendererFencedCodeAttributeContextEndPrototype
13420 \markdownRendererHeaderAttributeContextEndPrototype
13421 \cs_gset:Npn
13422 \markdownRendererReplacementCharacterPrototype
13423 { \codepoint_str_generate:n { fffd } }
13424 \ExplSyntaxOff
13425 \def\markdownRendererSectionBeginPrototype{ }%
13426 \def\markdownRendererSectionEndPrototype{ }%
13427 \ExplSyntaxOn
13428 \cs_gset:Npn
13429 \markdownRendererWarningPrototype
13430 #1#2#3#4
13431 {
13432   \tl_set:Nn
13433     \l_tmpa_tl
13434     { #2 }
13435   \tl_if_empty:nF
13436     { #4 }
13437     {
13438       \tl_put_right:Nn
13439         \l_tmpa_tl
13440         { \iow_newline: #4 }
13441     }
13442   \exp_args:NV
13443     \markdownWarning
13444     \l_tmpa_tl
13445   }
13446 \ExplSyntaxOff
13447 \def\markdownRendererErrorPrototype#1#2#3#4{%
13448   \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.

```

13449 \ExplSyntaxOn
13450 \cs_new:Nn
13451   \@@_plain_tex_default_input_raw_inline:nn
13452   {
13453     \str_case:nn
13454       { #2 }
13455       {
13456         { md } { \markdownInput{#1} }
13457         { tex } { \markdownEscape{#1} \unskip }
13458       }

```

```

13459 }
13460 \cs_new:Nn
13461 \@@_plain_tex_default_input_raw_block:nn
13462 {
13463   \str_case:nn
13464     { #2 }
13465     {
13466       { md } { \markdownInput{#1} }
13467       { tex } { \markdownEscape{#1} }
13468     }
13469 }
13470 \cs_gset:Npn
13471 \markdownRendererInputRawInlinePrototype#1#2
13472 {
13473   \@@_plain_tex_default_input_raw_inline:nn
13474     { #1 }
13475     { #2 }
13476 }
13477 \cs_gset:Npn
13478 \markdownRendererInputRawBlockPrototype#1#2
13479 {
13480   \@@_plain_tex_default_input_raw_block:nn
13481     { #1 }
13482     { #2 }
13483 }
13484 \ExplSyntaxOff

```

3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position p :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth p .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth p .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth p .

```

13485 \ExplSyntaxOn
13486 \seq_new:N \g_@@_jekyll_data_datatypes_seq
13487 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
13488 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
13489 \tl_const:Nn \c_@@_jekyll_data_scalar_tl { scalar }

```

To keep track of our current place when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_wildcard_absolute_address_seq` stack of keys using the `\markdown_jekyll_data_push_address_segment:n` macro.

```

13490 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
13491 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
13492   {
13493     \seq_if_empty:NF
13494       \g_@@_jekyll_data_datatypes_seq
13495       {
13496         \seq_get_right:NN
13497         \g_@@_jekyll_data_datatypes_seq
13498         \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.

```

13499     \str_if_eq:NNTF
13500       \l_tmpa_tl
13501       \c_@@_jekyll_data_sequence_tl
13502       {
13503         \seq_put_right:Nn
13504           \g_@@_jekyll_data_wildcard_absolute_address_seq
13505           { * }
13506       }
13507       {
13508         \seq_put_right:Nn
13509           \g_@@_jekyll_data_wildcard_absolute_address_seq
13510           { #1 }
13511       }
13512     }
13513   }

```

Out of `\g_@@_jekyll_data_wildcard_absolute_address_seq`, we will construct the following two token lists:

`\g_@@_jekyll_data_wildcard_absolute_address_tl` An *absolute wildcard*: The wildcard from the root of the document prefixed with a slash (/) with individual keys and asterisks also delimited by slashes. Allows the users to react to complex context-sensitive structures with ease.

For example, the `name` key in the following YAML document would correspond to the `/*/person/name` absolute wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

`\g_@@_jekyll_data_wildcard_relative_address_tl` A *relative wildcard*: The rightmost segment of the wildcard. Allows the users to react to simple context-free structures.

For example, the `name` key in the following YAML document would correspond to the `name` relative wildcard:

```
[{person: {name: Elon, surname: Musk}}]
```

We will construct `\g_@@_jekyll_data_wildcard_absolute_address_tl` using the `\markdown_jekyll_data_concatenate_address:NN` macro and we will construct both token lists using the `\markdown_jekyll_data_update_address_tls:` macro.

```
13514 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
13515 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
13516 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
13517 {
13518   \seq_pop_left:NN #1 \l_tmpa_tl
13519   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
13520   \seq_put_left:NV #1 \l_tmpa_tl
13521 }
13522 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
13523 {
13524   \markdown_jekyll_data_concatenate_address:NN
13525   \g_@@_jekyll_data_wildcard_absolute_address_seq
13526   \g_@@_jekyll_data_wildcard_absolute_address_tl
13527   \seq_get_right:NN
13528   \g_@@_jekyll_data_wildcard_absolute_address_seq
13529   \g_@@_jekyll_data_wildcard_relative_address_tl
13530 }
```

To make sure that the stacks and token lists stay in sync, we will use the `\markdown_jekyll_data_push:nN` and `\markdown_jekyll_data_pop:` macros.

```
13531 \cs_new:Nn \markdown_jekyll_data_push:nN
13532 {
13533   \markdown_jekyll_data_push_address_segment:n
13534   { #1 }
13535   \seq_put_right:NV
13536   \g_@@_jekyll_data_datatypes_seq
13537   #2
13538   \markdown_jekyll_data_update_address_tls:
13539 }
13540 \cs_new:Nn \markdown_jekyll_data_pop:
13541 {
13542   \seq_pop_right:NN
13543   \g_@@_jekyll_data_wildcard_absolute_address_seq
13544   \l_tmpa_tl
```

```

13545     \seq_pop_right:NN
13546     \g_@@_jekyll_data_datatypes_seq
13547     \l_tmpa_tl
13548     \markdown_jekyll_data_update_address_tls:
13549   }

```

To set a single key–value, we will use the `\markdown_jekyll_data_set_keyval:Nn` macro, ignoring unknown keys. To set key–values for both absolute and relative wildcards, we will use the `\markdown_jekyll_data_set_keyvals:nn` macro.

```

13550 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
13551   {
13552     \keys_set_known:nn
13553     { markdown/jekyllData }
13554     { { #1 } = { #2 } }
13555   }
13556 \cs_generate_variant:Nn
13557   \markdown_jekyll_data_set_keyval:nn
13558   { Vn }
13559 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
13560   {
13561     \markdown_jekyll_data_push:nN
13562     { #1 }
13563     \c_@@_jekyll_data_scalar_tl
13564     \markdown_jekyll_data_set_keyval:Vn
13565     \g_@@_jekyll_data_wildcard_absolute_address_tl
13566     { #2 }
13567     \markdown_jekyll_data_set_keyval:Vn
13568     \g_@@_jekyll_data_wildcard_relative_address_tl
13569     { #2 }
13570     \markdown_jekyll_data_pop:
13571   }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

13572 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13573   \markdown_jekyll_data_push:nN
13574   { #1 }
13575   \c_@@_jekyll_data_sequence_tl
13576 }
13577 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13578   \markdown_jekyll_data_push:nN
13579   { #1 }
13580   \c_@@_jekyll_data_mapping_tl
13581 }
13582 \def\markdownRendererJekyllDataSequenceEndPrototype{
13583   \markdown_jekyll_data_pop:
13584 }
13585 \def\markdownRendererJekyllDataMappingEndPrototype{

```



```

13586 \markdown_jekyll_data_pop:
13587 }
13588 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13589 \markdown_jekyll_data_set_keyvals:nn
13590 { #1 }
13591 { #2 }
13592 }
13593 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13594 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13595 \markdown_jekyll_data_set_keyvals:nn
13596 { #1 }
13597 { #2 }
13598 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13599 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13600 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13601 \markdown_jekyll_data_set_keyvals:nn
13602 { #1 }
13603 { #2 }
13604 }
13605 \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).

```

13606 \ExplSyntaxOn
13607 \str_if_eq:VVT
13608 \c_@@_top_layer_tl
13609 \c_@@_option_layer_plain_tex_tl
13610 {
13611 \ExplSyntaxOff
13612 \@@_if_option:nF
13613 { noDefaults }
13614 {
13615 \@@_if_option:nTF
13616 { experimental }
13617 {
13618 \@@_setup:n
13619 { theme = witiko/markdown/defaults@experimental }
13620 }
13621 {
13622 \@@_setup:n
13623 { theme = witiko/markdown/defaults }
13624 }
13625 }
13626 \ExplSyntaxOn

```

```

13627 }
13628 \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).

```

13629 \ExplSyntaxOn
13630 \tl_new:N \g_@@_formatted_lua_options_tl
13631 \cs_new:Nn \@@_format_lua_options:
13632 {
13633   \tl_gclear:N
13634   \g_@@_formatted_lua_options_tl
13635   \seq_map_function:NN
13636   \g_@@_lua_options_seq
13637   \@@_format_lua_option:n
13638 }
13639 \cs_new:Nn \@@_format_lua_option:n
13640 {
13641   \@@_typecheck_option:n
13642   { #1 }
13643   \@@_get_option_type:nN
13644   { #1 }
13645   \l_tmpa_tl
13646   \bool_case_true:nF
13647   {
13648     {
13649       \str_if_eq_p:VV
13650       \l_tmpa_tl
13651       \c_@@_option_type_boolean_tl ||
13652       \str_if_eq_p:VV
13653       \l_tmpa_tl
13654       \c_@@_option_type_number_tl ||
13655       \str_if_eq_p:VV
13656       \l_tmpa_tl
13657       \c_@@_option_type_counter_tl
13658     }
13659     {
13660       \@@_get_option_value:nN
13661       { #1 }
13662       \l_tmpa_tl
13663       \tl_gput_right:Nx
13664       \g_@@_formatted_lua_options_tl
13665       { #1~::~ \l_tmpa_tl ,~ }
13666     }

```

```

13667 {
13668   \str_if_eq_p:VV
13669   \l_tmpa_tl
13670   \c_@@_option_type_clist_tl
13671 }
13672 {
13673   \@@_get_option_value:nN
13674   { #1 }
13675   \l_tmpa_tl
13676   \tl_gput_right:Nx
13677   \g_@@_formatted_lua_options_tl
13678   { #1~::~\c_left_brace_str }
13679   \clist_map_inline:Vn
13680   \l_tmpa_tl
13681   {
13682     \@@_lua_escape:xN
13683     { ##1 }
13684     \l_tmpb_tl
13685     \tl_gput_right:Nn
13686     \g_@@_formatted_lua_options_tl
13687     { " }
13688     \tl_gput_right:NV
13689     \g_@@_formatted_lua_options_tl
13690     \l_tmpb_tl
13691     \tl_gput_right:Nn
13692     \g_@@_formatted_lua_options_tl
13693     { " ,~ }
13694   }
13695   \tl_gput_right:Nx
13696   \g_@@_formatted_lua_options_tl
13697   { \c_right_brace_str ,~ }
13698 }
13699 }
13700 {
13701   \@@_get_option_value:nN
13702   { #1 }
13703   \l_tmpa_tl
13704   \@@_lua_escape:xN
13705   { \l_tmpa_tl }
13706   \l_tmpb_tl
13707   \tl_gput_right:Nn
13708   \g_@@_formatted_lua_options_tl
13709   { #1~::~ " }
13710   \tl_gput_right:NV
13711   \g_@@_formatted_lua_options_tl
13712   \l_tmpb_tl
13713   \tl_gput_right:Nn

```

```

13714         \g_@@_formatted_lua_options_tl
13715         { " ,~ }
13716     }
13717 }
13718 \cs_generate_variant:Nn
13719   \clist_map_inline:nn
13720   { Vn }
13721 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
13722 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
13723 \sys_if_engine luatex:TF
13724 {
13725   \cs_new:Nn
13726     \@@_lua_escape:nN
13727     {
13728       \tl_set:Nx
13729         #2
13730         {
13731           \lua_escape:n
13732             { #1 }
13733         }
13734     }
13735 }
13736 {
13737   \regex_const:Nn
13738     \c_@@_lua_escape_regex
13739     { [\\"' ] }
13740   \cs_new:Nn
13741     \@@_lua_escape:nN
13742     {
13743       \tl_set:Nn
13744         #2
13745         { #1 }
13746       \regex_replace_all:NnN
13747         \c_@@_lua_escape_regex
13748         { \u { c_backslash_str } \0 }
13749         #2
13750     }
13751 }
13752 \cs_generate_variant:Nn
13753   \@@_lua_escape:nN
13754   { 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.

```

13755 \tl_new:N
13756   \markdownInputFilename

```

```

13757 \cs_new:Npn
13758   \markdownPrepareInputFilename
13759   #1
13760   {
13761     \@_lua_escape:xN
13762     { #1 }
13763     \markdownInputFilename
13764     \tl_gset:Nx
13765     \markdownInputFilename
13766     { " \markdownInputFilename " }
13767   }

```

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.

```

13768 \cs_new:Npn
13769   \markdownPrepare
13770   {

```

First, ensure that the `cacheDir` directory exists.

```

13771     local~lfs = require("lfs")
13772     local~options = \markdownLuaOptions
13773     if~not~lfs.isdir(options.cacheDir) then~
13774         assert(lfs.mkdir(options.cacheDir))
13775     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.

```

13776     local~md = require("markdown")
13777     local~convert = md.new(options)
13778   }

```

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.

```

13779 \cs_new:Npn
13780   \markdownConvert
13781   {
13782     local~filename = \markdownInputFilename
13783     local~file = assert(io.open(filename, "r"),
13784       [[Could~not~open~file~]] .. filename .. [[~for~reading]])
13785     local~input = assert(file:read("*a"))
13786     assert(file:close())
13787     print(convert(input))
13788   }
13789 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain \TeX .

```

13790 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
13791   if options.cacheDir then
13792     lfs.rmdir(options.cacheDir)
13793   end
13794 }%
```

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.

```

13795 \csname newread\endcsname\markdownInputFileStream
13796 \csname newwrite\endcsname\markdownOutputFileStream
```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

13797 \begingroup
13798   \catcode\^^I=12%
13799   \gdef\markdownReadAndConvertTab{^^I}%
13800 \endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the $\LaTeX 2_{\epsilon}$ `\filecontents` macro to plain \TeX .

```

13801 \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.
13802   \catcode\^^M=13%
13803   \catcode\^^I=13%
13804   \catcode|=0%
13805   \catcode\|=12%
13806   |catcode@=14%
13807   |catcode|=12@
13808   |gdef|markdownReadAndConvert#1#2{@
13809     |begingroup@
```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

13810   |markdownIfOption{frozenCache}{-}{@
13811     |immediate|openout|markdownOutputFileStream@
13812     |markdownOptionInputTempFileName|relax@
13813     |markdownInfo{@
13814     Buffering block-level markdown input into the temporary @
13815     input file "|markdownOptionInputTempFileName" and scanning @
```

```

13816         for the closing token sequence "#1"}@
13817     }@

```

Locally change the category of the special plain T_EX 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.

```

13818     |def|do##1{|catcode`##1=12}|dospecials@
13819     |catcode`| =12@
13820     |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `stripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (\sim M) are produced.

```

13821     |def|markdownReadAndConvertStripPercentSign##1{@
13822     |markdownIfOption{stripPercentSigns}{@
13823     |if##1%@
13824     |expandafter|expandafter|expandafter@
13825     |markdownReadAndConvertProcessLine@
13826     |else@
13827     |expandafter|expandafter|expandafter@
13828     |markdownReadAndConvertProcessLine@
13829     |expandafter|expandafter|expandafter##1@
13830     |fi@
13831     }{@
13832     |expandafter@
13833     |markdownReadAndConvertProcessLine@
13834     |expandafter##1@
13835     }@
13836     }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (\sim M) are produced.

```

13837     |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

13838     |ifx|relax##3|relax@
13839     |markdownIfOption{frozenCache}{-}{@
13840     |immediate|write|markdownOutputFileStream{##1}@
13841     }@
13842     |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 TeX, `\input` the result of the conversion, and expand the ending control sequence.

```

13843     |def^^M{@
13844         |markdownInfo{The ending token sequence was found}@
13845         |markdownIfOption{frozenCache}{-}{@
13846             |immediate|closeout|markdownOutputFileStream@
13847         }@
13848     |endgroup@
13849     |markdownInput{@
13850         |markdownOptionOutputDir@
13851         /|markdownOptionInputTempFileName@
13852     }@
13853     #2}@
13854     |fi@

```

Repeat with the next line.

```

13855     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

13856     |catcode`\^^I=13@
13857     |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.

```

13858     |catcode`\^^M=13@
13859     |def^^M##1^^M{@
13860         |def^^M###1^^M{@
13861             |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
13862         ^^M}@
13863     ^^M}@

```

Reset the character categories back to the former state.

```

13864 |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.

```

13865 \ExplSyntaxOn
13866 \cs_new:Npn
13867   \markdownLuaExecute
13868   #1
13869   {
13870     \int_compare:nNt
13871     { \g_luabridge_method_int }
13872     =
13873     { \c_luabridge_method_shell_int }
13874     {

```



```

13875     \sys_if_shell_unrestricted:F
13876     {
13877         \sys_if_shell:TF
13878         {
13879             \msg_error:nn
13880             { markdown }
13881             { restricted-shell-access }
13882         }
13883         {
13884             \msg_error:nn
13885             { markdown }
13886             { disabled-shell-access }
13887         }
13888     }
13889 }
13890 \str_gset:NV
13891   \g_luabridge_output_dirname_str
13892   \markdownOptionOutputDir
13893   \luabridge_now:e
13894   { #1 }
13895 }
13896 \cs_generate_variant:Nn
13897   \msg_new:nnnn
13898   { nnnV }
13899 \tl_set:Nn
13900   \l_tmpa_tl
13901   {
13902     You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
13903     --enable-write18~flag,~or~write~shell_escape=t~in~the~
13904     texmf.cnf~file.
13905   }
13906 \msg_new:nnnV
13907   { markdown }
13908   { restricted-shell-access }
13909   { Shell~escape~is~restricted }
13910   \l_tmpa_tl
13911 \msg_new:nnnV
13912   { markdown }
13913   { disabled-shell-access }
13914   { Shell~escape~is~disabled }
13915   \l_tmpa_tl
13916 \ExplSyntaxOff

```

3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

13917 \ExplSyntaxOn

```

```

13918 \tl_new:N
13919   \g_@@_after_markinline_tl
13920 \tl_gset:Nn
13921   \g_@@_after_markinline_tl
13922   { \unskip }
13923 \cs_new:Npn
13924   \markinline
13925   {

```

Locally change the category of the special plain T_EX characters to *other* in order to prevent unwanted interpretation of the input markdown text as T_EX code.

```

13926   \group_begin:
13927   \cctab_select:N
13928   \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

13929   \@@_if_option:nF
13930   { frozenCache }
13931   {
13932     \immediate
13933     \openout
13934     \markdownOutputFileStream
13935     \markdownOptionInputTempFileName
13936     \relax
13937     \msg_info:nne
13938     { markdown }
13939     { buffering-markinline }
13940     { \markdownOptionInputTempFileName }
13941   }

```

Peek ahead and extract the inline markdown text.

```

13942   \peek_regex_replace_once:nnF
13943   { { (.*) } }
13944   {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

13945   \c { @@_if_option:nF }
13946   \cB { frozenCache \cE }
13947   \cB {
13948     \c { immediate }
13949     \c { write }
13950     \c { markdownOutputFileStream }
13951     \cB { \1 \cE }
13952     \c { immediate }
13953     \c { closeout }
13954     \c { markdownOutputFileStream }
13955     \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

13956     \c { group_end: }
13957     \c { group_begin: }
13958     \c { @@_setup:n }
13959     \cB { contentLevel = inline \cE }
13960     \c { markdownInput }
13961     \cB {
13962         \c { markdownOptionOutputDir } /
13963         \c { markdownOptionInputTempFileName }
13964     \cE }
13965     \c { group_end: }
13966     \c { tl_use:N }
13967     \c { g_@@_after_markinline_tl }
13968 }
13969 {
13970     \msg_error:nn
13971     { markdown }
13972     { markinline-peek-failure }
13973     \group_end:
13974     \tl_use:N
13975     \g_@@_after_markinline_tl
13976 }
13977 }
13978 \msg_new:nnn
13979 { markdown }
13980 { buffering-markinline }
13981 { Buffering~inline~markdown~input~into~
13982   the~temporary~input~file~"#1". }
13983 \msg_new:nnnn
13984 { markdown }
13985 { markinline-peek-failure }
13986 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
13987 { The~macro~should~be~followed~by~inline~
13988   markdown~text~in~curly~braces }
13989 \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_EX.

```

13990 \ExplSyntaxOn
13991 \cs_new:Npn
13992   \markdownInput
13993   #1
13994   {

```

```

13995 \@@_if_option:nTF
13996 { frozenCache }
13997 {
13998   \markdownInputRaw
13999   { #1 }
14000 }
14001 {

```

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 \LaTeX , this also includes the directories specified in `\input@path`.

```

14002   \tl_set:Nx
14003   \l_tmpa_tl
14004   { #1 }
14005   \file_get_full_name:VNTF
14006   \l_tmpa_tl
14007   \l_tmpb_tl
14008   {
14009     \exp_args:NV
14010     \markdownInputRaw
14011     \l_tmpb_tl
14012   }
14013   {
14014     \msg_error:nnV
14015     { markdown }
14016     { markdown-file-does-not-exist }
14017     \l_tmpa_tl
14018   }
14019 }
14020 }
14021 \msg_new:nnn
14022 { markdown }
14023 { markdown-file-does-not-exist }
14024 {
14025   Markdown~file~#1~does~not~exist
14026 }
14027 \ExplSyntaxOff
14028 \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.

```

14029 \catcode`|=0%
14030 \catcode`\|=12%
14031 \catcode`|&=6%
14032 |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.

```
14033     |begingroup
14034     |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.

```
14035     |catcode`\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
14036     |markdownIfOption{frozenCache}{%
14037         |ifnum|markdownOptionFrozenCacheCounter=0|relax
14038         |markdownInfo{Reading frozen cache from
14039             "|markdownOptionFrozenCacheFileName"}%
14040         |input|markdownOptionFrozenCacheFileName|relax
14041         |fi
14042         |markdownInfo{Including markdown document number
14043             "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
14044         |csname markdownFrozenCache%
14045             |the|markdownOptionFrozenCacheCounter|endcsname
14046         |global|advance|markdownOptionFrozenCacheCounter by 1|relax
14047     }{%
14048         |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L^AT_EX_Mk to track changes to the markdown document.

```
14049         |openin|markdownInputFileStream&1
14050         |closein|markdownInputFileStream
14051         |markdownPrepareLuaOptions
14052         |markdownPrepareInputFilename{&1}%
14053         |markdownLuaExecute{%
14054             |markdownPrepare
14055             |markdownConvert
14056             |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
14057         |markdownIfOption{finalizeCache}{%
14058             |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
14059     }%
14060     |endgroup
14061 }%
14062 |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.

```
14063 \gdef\markdownEscape#1{%
14064   \catcode`\%=14\relax
14065   \catcode`\#=6\relax
14066   \input #1\relax
14067   \catcode`\%=12\relax
14068   \catcode`\#=12\relax
14069 }%
```

3.3 \LaTeX Implementation

The \LaTeX implementation makes use of the fact that, apart from some subtle differences, \LaTeX implements the majority of the plain \TeX format [15, Section 9]. As a consequence, we can directly reuse the existing plain \TeX implementation.

```
14070 \def\markdownVersionSpace{ }%
14071 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
14072   \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).

```
14073 \ExplSyntaxOn
14074 \cs_gset_eq:NN
14075   \markinlinePlainTeX
14076   \markinline
14077 \cs_gset:Npn
14078   \markinline
14079   {
14080     \peek_regex_replace_once:nn
14081       { ( \[ (.*) \] ) ? }
14082     {
```

Apply the options locally.

```
14083       \c { group_begin: }
14084       \c { @@_setup:n }
14085       \cB { \2 \cE }
14086       \c { tl_put_right:Nn }
14087       \c { g_@@_after_markinline_tl }
14088       \cB { \c { group_end: } \cE }
```

```

14089     \c { markinlinePlainTeX }
14090   }
14091 }
14092 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the L^AT_EX interface (see Section 2.3.3).

```

14093 \let\markdownInputPlainTeX\markdownInput
14094 \renewcommand\markdownInput[2] [] {%
14095   \begingroup
14096     \markdownSetup{#1}%
14097     \markdownInputPlainTeX{#2}%
14098   \endgroup}%
14099 \renewcommand\yamlInput[2] [] {%
14100   \begingroup
14101     \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
14102     \markdownInputPlainTeX{#2}%
14103   \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L^AT_EX environments are implemented using the `\markdownReadAndConvert` macro.

```

14104 \ExplSyntaxOn
14105 \renewenvironment
14106   { markdown }
14107   {

```

In our implementation of the `markdown` L^AT_EX 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^AT_EX 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^AT_EX environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return

character will be produced by \TeX via the `\endlinechar` plain \TeX macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
14108     \group_begin:
14109     \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).

```
14110     \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
14111     \peek_regex_replace_once:nnF
14112     { \ *[\r*([^\r]*)\][^\r]* }
14113     {
```

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.

```
14114         \c { group_end: }
14115         \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
14116         \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}
```

```
14117         \c { exp_args:NV }
14118         \c { markdownReadAndConvert@ }
14119         \c { @currenvir }
14120     }
14121     {
14122         \group_end:
14123         \exp_args:NV
14124         \markdownReadAndConvert@
14125         \@currenvir
```



```

14126     }
14127   }
14128   { \markdownEnd }
14129 \renewenvironment
14130   { markdown* }
14131   [ 1 ]
14132   {
14133     \@@_if_option:nTF
14134       { experimental }
14135       {
14136         \msg_error:nnn
14137           { markdown }
14138           { latex-markdown-star-deprecated }
14139           { #1 }
14140       }
14141       {
14142         \msg_warning:nnn
14143           { markdown }
14144           { latex-markdown-star-deprecated }
14145           { #1 }
14146       }
14147     \@@_setup:n
14148       { #1 }
14149     \markdownReadAndConvert@
14150       { markdown* }
14151   }
14152   { \markdownEnd }
14153 \renewenvironment
14154   { yaml }
14155   {
14156     \group_begin:
14157     \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
14158     \markdown
14159   }
14160   { \yamlEnd }
14161 \msg_new:nnn
14162   { markdown }
14163   { latex-markdown-star-deprecated }
14164   {
14165     The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
14166     be~removed~in~the~next~major~version~of~the~Markdown~package.
14167   }
14168 \cs_generate_variant:Nn
14169   \@@_setup:n
14170   { V }
14171 \ExplSyntaxOff
14172 \beginingroup

```

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*.

```

14173 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
14174 \catcode`\=12\catcode`\{=12\catcode`\}=12%
14175 |gdef|markdownReadAndConvert@#1<%
14176     |markdownReadAndConvert<\end{#1}>%
14177         <|end<#1>>%
14178 |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.

```

14179 \ExplSyntaxOn
14180 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
14181 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
14182 \cs_gset:Nn
14183   \@@_load_theme:nnn
14184   {

```

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.

```

14185   \ifmarkdownLaTeXLoaded
14186     \ifx\@onlypreamble\@notprerr

```

If both conditions are true, end with an error, since we cannot load \LaTeX themes after the preamble.

```

14187     \bool_if:nTF
14188     {
14189       \bool_lazy_or_p:nn
14190       {
14191         \prop_if_in_p:Nn
14192         \g_@@_latex_built_in_themes_prop
14193         { #1 }
14194       }
14195       {
14196         \file_if_exist_p:n
14197         { markdown theme #3.sty }
14198       }
14199     }
14200     {

```

```

14201         \msg_error:nnn
14202         { markdown }
14203         { latex-theme-after-preamble }
14204         { #1 }
14205     }

```

Otherwise, try loading a plain \TeX theme instead.

```

14206     {
14207         \@@_plain_tex_load_theme:nnn
14208         { #1 }
14209         { #2 }
14210         { #3 }
14211     }
14212     \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a \LaTeX theme if it exists or load a plain \TeX theme otherwise.

```

14213     \bool_if:nTF
14214     {
14215         \bool_lazy_or_p:nn
14216         {
14217             \prop_if_in_p:Nn
14218             \g_@@_latex_built_in_themes_prop
14219             { #1 }
14220         }
14221         {
14222             \file_if_exist_p:n
14223             { markdown theme #3.sty }
14224         }
14225     }
14226     {
14227         \prop_get:NnNTF
14228         \g_@@_latex_loaded_themes_linenos_prop
14229         { #1 }
14230         \l_tmpa_tl
14231         {
14232             \prop_get:NnN
14233             \g_@@_latex_loaded_themes_versions_prop
14234             { #1 }
14235             \l_tmpb_tl
14236             \str_if_eq:nVTF
14237             { #2 }
14238             \l_tmpb_tl
14239             {
14240                 \msg_warning:nnnVn
14241                 { markdown }
14242                 { repeatedly-loaded-latex-theme }
14243                 { #1 }

```

```

14244         \l_tmpa_tl
14245         { #2 }
14246     }
14247     {
14248         \msg_error:nnnnVV
14249         { markdown }
14250         { different-versions-of-latex-theme }
14251         { #1 }
14252         { #2 }
14253         \l_tmpb_tl
14254         \l_tmpa_tl
14255     }
14256 }
14257 {
14258     \prop_gput:Nnx
14259     \g_@@_latex_loaded_themes_linenos_prop
14260     { #1 }
14261     { \tex_the:D \tex_inputlineno:D }
14262     \prop_gput:Nnn
14263     \g_@@_latex_loaded_themes_versions_prop
14264     { #1 }
14265     { #2 }

```

Load built-in plain TeX themes from the prop `\g_@@_latex_built_in_themes_prop` and from the filesystem otherwise.

```

14266     \prop_if_in:NnTF
14267     \g_@@_latex_built_in_themes_prop
14268     { #1 }
14269     {
14270         \msg_info:nnnn
14271         { markdown }
14272         { loading-built-in-latex-theme }
14273         { #1 }
14274         { #2 }
14275         \prop_item:Nn
14276         \g_@@_latex_built_in_themes_prop
14277         { #1 }
14278     }
14279     {
14280         \msg_info:nnnn
14281         { markdown }
14282         { loading-latex-theme }
14283         { #1 }
14284         { #2 }
14285         \RequirePackage
14286         { markdown theme #3 }
14287     }

```

```

14288         }
14289     }
14290     {
14291         \@_plain_tex_load_theme:nnn
14292         { #1 }
14293         { #2 }
14294         { #3 }
14295     }
14296     \fi
14297     \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

14298     \msg_info:nnnn
14299     { markdown }
14300     { theme-loading-postponed }
14301     { #1 }
14302     { #2 }
14303     \AtEndOfPackage
14304     {
14305         \@_set_theme:n
14306         { #1 @ #2 }
14307     }
14308     \fi
14309 }
14310 \msg_new:nnn
14311 { markdown }
14312 { theme-loading-postponed }
14313 {
14314     Postponing~loading~version~#2~of~Markdown~theme~#1~until~
14315     Markdown~package~has~finished~loading
14316 }
14317 \msg_new:nnn
14318 { markdown }
14319 { loading-built-in-latex-theme }
14320 { Loading~version~#2~of~built-in-LaTeX-Markdown~theme~#1 }
14321 \msg_new:nnn
14322 { markdown }
14323 { loading-latex-theme }
14324 { Loading~version~#2~of~LaTeX-Markdown~theme~#1 }
14325 \msg_new:nnn
14326 { markdown }
14327 { repeatedly-loaded-latex-theme }
14328 {
14329     Version~#3~of~LaTeX-Markdown~theme~#1~was~previously~
14330     loaded~on~line~#2,~not~loading~it~again
14331 }

```

```

14332 \msg_new:nnn
14333   { markdown }
14334   { different-versions-of-latex-theme }
14335   {
14336     Tried-to-load-version-#2-of-LaTeX-Markdown-theme-#1~
14337     but-version-#3-has-already-been-loaded-on-line-#4
14338   }
14339 \cs_generate_variant:Nn
14340   \msg_new:nnnn
14341   { nnVV }
14342 \tl_set:Nn
14343   \l_tmpa_tl
14344   { Cannot-load-LaTeX-Markdown-theme-#1~after~ }
14345 \tl_put_right:NV
14346   \l_tmpa_tl
14347   \c_backslash_str
14348 \tl_put_right:Nn
14349   \l_tmpa_tl
14350   { begin{document} }
14351 \tl_set:Nn
14352   \l_tmpb_tl
14353   { Load-Markdown-theme-#1~before~ }
14354 \tl_put_right:NV
14355   \l_tmpb_tl
14356   \c_backslash_str
14357 \tl_put_right:Nn
14358   \l_tmpb_tl
14359   { begin{document} }
14360 \msg_new:nnVV
14361   { markdown }
14362   { latex-theme-after-preamble }
14363   \l_tmpa_tl
14364   \l_tmpb_tl

```

The [witiko/dot](#) and [witiko/graphicx/http](#) L^AT_EX themes load the package `graphicx`, see also Section 1.1.3. Then, they load the corresponding plain T_EX themes.

```

14365 \tl_set:Nn
14366   \l_tmpa_tl
14367   {
14368     \RequirePackage
14369     { graphicx }
14370     \markdownLoadPlainTeXTheme
14371   }
14372 \prop_gput:NnV
14373   \g_@@_latex_built_in_themes_prop
14374   { witiko / dot }
14375   \l_tmpa_tl

```

```

14376 \prop_gput:NnV
14377   \g_@@_latex_built_in_themes_prop
14378   { witiko / graphicx / http }
14379   \l_tmpa_tl
14380 \ExplSyntaxOff

```

The `witiko/markdown/defaults` L^AT_EX theme also loads the corresponding plain T_EX theme.

```
14381 \markdownLoadPlainTeXTheme
```

Next, the L^AT_EX theme overrides some of the plain T_EX definitions. See Section 3.3.4 for the actual definitions.

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

14382 \DeclareOption*{%
14383   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
14384 \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.

```
14385 \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`.

```

14386 \ExplSyntaxOn
14387 \bool_new:N
14388   \g_@@_tight_or_fancy_lists_bool
14389 \bool_gset_false:N
14390   \g_@@_tight_or_fancy_lists_bool
14391 \@@_if_option:nTF
14392 { tightLists }
14393 {
14394   \bool_gset_true:N
14395     \g_@@_tight_or_fancy_lists_bool
14396 }
14397 {
14398   \@@_if_option:nT

```

```

14399     { fancyLists }
14400     {
14401         \bool_gset_true:N
14402         \g_@@_tight_or_fancy_lists_bool
14403     }
14404 }
14405 \bool_new:N
14406 \g_@@_beamer_paralist_or_enumitem_bool
14407 \bool_gset_true:N
14408 \g_@@_beamer_paralist_or_enumitem_bool
14409 \@ifclassloaded
14410 { beamer }
14411 { }
14412 {
14413     \@ifpackageloaded
14414     { paralist }
14415     { }
14416     {
14417         \@ifpackageloaded
14418         { enumitem }
14419         { }
14420         {
14421             \bool_gset_false:N
14422             \g_@@_beamer_paralist_or_enumitem_bool
14423         }
14424     }
14425 }
14426 \bool_if:nT
14427 {
14428     \g_@@_tight_or_fancy_lists_bool &&
14429     ! \g_@@_beamer_paralist_or_enumitem_bool
14430 }
14431 {
14432     \bool_if:nTF
14433     {
14434         \bool_lazy_or_p:nn
14435         {
14436             \str_if_eq_p:en
14437             { \markdownThemeVersion }
14438             { experimental }
14439         }
14440         {
14441             \bool_lazy_and_p:nn
14442             {
14443                 \prop_if_exist_p:N
14444                 \g__pdfmanagement_documentproperties_prop
14445             }

```



```

14446     {
14447         \bool_lazy_any_p:n
14448     {
14449         {
14450             \prop_if_in_p:Nn
14451             \g__pdfmanagement_documentproperties_prop
14452             { document / testphase / phase-I }
14453         }
14454         {
14455             \prop_if_in_p:Nn
14456             \g__pdfmanagement_documentproperties_prop
14457             { document / testphase / phase-II }
14458         }
14459         {
14460             \prop_if_in_p:Nn
14461             \g__pdfmanagement_documentproperties_prop
14462             { document / testphase / phase-III }
14463         }
14464         {
14465             \prop_if_in_p:Nn
14466             \g__pdfmanagement_documentproperties_prop
14467             { document / testphase / phase-IV }
14468         }
14469         {
14470             \prop_if_in_p:Nn
14471             \g__pdfmanagement_documentproperties_prop
14472             { document / testphase / phase-V }
14473         }
14474         {
14475             \prop_if_in_p:Nn
14476             \g__pdfmanagement_documentproperties_prop
14477             { document / testphase / phase-VI }
14478         }
14479     }
14480     }
14481 }
14482 }
14483 {
14484     \RequirePackage
14485     { enumitem }
14486 }
14487 {
14488     \RequirePackage
14489     { paralist }
14490 }
14491 }
14492 \ExplSyntaxOff

```

If we loaded the `enumitem` package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

14493 \ExplSyntaxOn
14494 \cs_new:Nn
14495   \@@_latex_fancy_list_item_label_number:nn
14496   {
14497     \str_case:nn
14498       { #1 }
14499       {
14500         { Decimal } { #2 }
14501         { LowerRoman } { \int_to_roman:n { #2 } }
14502         { UpperRoman } { \int_to_Roman:n { #2 } }
14503         { LowerAlpha } { \int_to_alph:n { #2 } }
14504         { UpperAlpha } { \int_to_Alph:n { #2 } }
14505       }
14506   }
14507 \cs_new:Nn
14508   \@@_latex_fancy_list_item_label_delimiter:n
14509   {
14510     \str_case:nn
14511       { #1 }
14512       {
14513         { Default } { . }
14514         { OneParen } { ) }
14515         { Period } { . }
14516       }
14517   }
14518 \cs_new:Nn
14519   \@@_latex_fancy_list_item_label:nnn
14520   {
14521     \@@_latex_fancy_list_item_label_number:nn
14522       { #1 }
14523       { #3 }
14524     \@@_latex_fancy_list_item_label_delimiter:n
14525       { #2 }
14526   }
14527 \cs_generate_variant:Nn
14528   \@@_latex_fancy_list_item_label:nnn
14529   { VVn }
14530 \tl_new:N
14531   \l_@@_latex_fancy_list_item_label_number_style_tl
14532 \tl_new:N
14533   \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14534 \ifpackageloaded{enumitem}{
14535   \markdownSetup{rendererPrototypes={

```

First, let's define the tight list item renderer prototypes.

```

14536     ulBeginTight = {
14537         \begin
14538             { itemize }
14539             [ noitemsep ]
14540     },
14541     ulEndTight = {
14542         \end
14543             { itemize }
14544     },
14545     olBeginTight = {
14546         \begin
14547             { enumerate }
14548             [ noitemsep ]
14549     },
14550     olEndTight = {
14551         \end
14552             { enumerate }
14553     },
14554     dlBeginTight = {
14555         \begin
14556             { description }
14557             [ noitemsep ]
14558     },
14559     dlEndTight = {
14560         \end
14561             { description }
14562     },

```

Second, let's define the fancy list item renderer prototypes.

```

14563     fancyOlBegin = {
14564         \group_begin:
14565         \tl_set:Nn
14566             \l_@@_latex_fancy_list_item_label_number_style_tl
14567             { #1 }
14568         \tl_set:Nn
14569             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14570             { #2 }
14571         \begin
14572             { enumerate }
14573     },
14574     fancyOlBeginTight = {
14575         \group_begin:
14576         \tl_set:Nn
14577             \l_@@_latex_fancy_list_item_label_number_style_tl
14578             { #1 }
14579         \tl_set:Nn
14580             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14581             { #2 }

```

```

14582     \begin
14583     { enumerate }
14584     [ noitemsep ]
14585   },
14586   fancyO1End(|Tight) = {
14587     \end { enumerate }
14588     \group_end:
14589   },
14590   fancyO1ItemWithNumber = {
14591     \item
14592     [
14593       \@@_latex_fancy_list_item_label:VVn
14594       \l_@@_latex_fancy_list_item_label_number_style_tl
14595       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14596       { #1 }
14597     ]
14598   },
14599   }}

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

14600 }{\@ifpackageloaded{paralist}{
14601   \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

14602   ulBeginTight = {%
14603     \group_begin:
14604     \pltopsep=\topsep
14605     \plpartopsep=\partopsep
14606     \begin{compactitem}
14607   },
14608   ulEndTight = {
14609     \end{compactitem}
14610     \group_end:
14611   },
14612   fancyO1Begin = {
14613     \group_begin:
14614     \tl_set:Nn
14615     \l_@@_latex_fancy_list_item_label_number_style_tl
14616     { #1 }
14617     \tl_set:Nn
14618     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14619     { #2 }
14620     \begin{enumerate}
14621   },
14622   fancyO1End = {
14623     \end{enumerate}

```

```

14624     \group_end:
14625     },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

14626     olBeginTight = {%
14627         \group_begin:
14628         \plpartopsep=\partopsep
14629         \pltopsep=\topsep
14630         \begin{compactenum}
14631     },
14632     olEndTight = {
14633         \end{compactenum}
14634         \group_end:
14635     },
14636     fancyOlBeginTight = {
14637         \group_begin:
14638         \tl_set:Nn
14639             \l_@@_latex_fancy_list_item_label_number_style_tl
14640             { #1 }
14641         \tl_set:Nn
14642             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14643             { #2 }
14644         \plpartopsep=\partopsep
14645         \pltopsep=\topsep
14646         \begin{compactenum}
14647     },
14648     fancyOlEndTight = {
14649         \end{compactenum}
14650         \group_end:
14651     },
14652     fancyOlItemWithNumber = {
14653         \item
14654         [
14655             \@@_latex_fancy_list_item_label:VVn
14656             \l_@@_latex_fancy_list_item_label_number_style_tl
14657             \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14658             { #1 }
14659         ]
14660     },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

14661     dlBeginTight = {
14662         \group_begin:
14663         \plpartopsep=\partopsep
14664         \pltopsep=\topsep
14665         \begin{compactdesc}

```

```

14666     },
14667     dlEndTight = {
14668         \end{compactdesc}
14669         \group_end:
14670     }
14671 }}
14672 }{

```

Otherwise, if we loaded neither the enunitem 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.

```

14673 \markdownSetup
14674 {
14675     rendererPrototypes = {
14676         ulBeginTight = \markdownRendererUlBegin,
14677         ulEndTight = \markdownRendererUlEnd,
14678         fancyOlBegin = \markdownRendererOlBegin,
14679         fancyOlEnd = \markdownRendererOlEnd,
14680         olBeginTight = \markdownRendererOlBegin,
14681         olEndTight = \markdownRendererOlEnd,
14682         fancyOlBeginTight = \markdownRendererOlBegin,
14683         fancyOlEndTight = \markdownRendererOlEnd,
14684         dlBeginTight = \markdownRendererDlBegin,
14685         dlEndTight = \markdownRendererDlEnd,
14686     },
14687 }
14688 }}
14689 \ExplSyntaxOff
14690 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

14691 \@ifpackageloaded{unicode-math}{
14692     \markdownSetup{rendererPrototypes={
14693         untickedBox = {\$ \mdlgwhtsquare$},
14694     }}
14695 }{
14696     \RequirePackage{amssymb}
14697     \markdownSetup{rendererPrototypes={
14698         untickedBox = {\$ \square$},
14699     }}
14700 }
14701 \RequirePackage{csvsimple}
14702 \RequirePackage{fancyvrb}
14703 \RequirePackage{graphicx}
14704 \markdownSetup{rendererPrototypes={
14705     hardLineBreak = {\},
14706     leftBrace = {\textbraceleft},

```

```

14707 rightBrace = {\textbraceright},
14708 dollarSign = {\textdollar},
14709 underscore = {\textunderscore},
14710 circumflex = {\textasciicircum},
14711 backslash = {\textbackslash},
14712 tilde = {\textasciitilde},
14713 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T_EX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,³⁴ we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

14714 codeSpan = {%
14715   \ifmmode
14716     \text{#1}%
14717   \else
14718     \texttt{#1}%
14719   \fi
14720 }}

```

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.

```

14721 \ExplSyntaxOn
14722 \markdownSetup{
14723   rendererPrototypes = {
14724     contentBlock = {
14725       \str_case:nnF
14726         { #1 }
14727         {
14728           { csv }
14729           {
14730             \begin{table}
14731               \begin{center}
14732                 \csvautotabular{#3}
14733               \end{center}
14734             \tl_if_empty:nF

```

³⁴This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

14735         { #4 }
14736         { \caption{#4} }
14737     \end{table}
14738     }
14739 { html }
14740     {

```

If we are using $\text{T}_{\text{E}}\text{X}4\text{ht}$ ³⁵, we will pass HTML elements to the output HTML document unchanged.

```

14741     \cs_if_exist:NTF
14742     \HCode
14743     {
14744         \if_mode_vertical:
14745         \IgnorePar
14746         \fi:
14747         \EndP
14748         \special
14749         { t4ht* < #3 }
14750         \par
14751         \ShowPar
14752     }
14753     {
14754         \@_luaxml_print_html:n
14755         { #3 }
14756     }
14757 }
14758 { tex }
14759 {
14760     \markdownEscape
14761     { #3 }
14762 }
14763 }
14764 {
14765     \markdownInput
14766     { #3 }
14767 }
14768 },
14769 },
14770 }
14771 \ExplSyntaxOff
14772 \markdownSetup{rendererPrototypes={
14773     ulBegin = {\begin{itemize}},
14774     ulEnd = {\end{itemize}},
14775     olBegin = {\begin{enumerate}},
14776     olItem = {\item{}},
14777     olItemWithNumber = {\item[#1.]},

```

³⁵See <https://tug.org/tex4ht/>.


```

14778 olEnd = {\end{enumerate}},
14779 dlBegin = {\begin{description}},
14780 dlItem = {\item[#1]},
14781 dlEnd = {\end{description}},
14782 emphasis = {\emph{#1}},
14783 tickedBox = {\$\boxtimes$},
14784 halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

14785 \ExplSyntaxOn
14786 \seq_new:N
14787   \l_@@_header_identifiers_seq
14788 \markdownSetup
14789   {
14790     rendererPrototypes = {
14791       headerAttributeContextBegin = {
14792         \markdownSetup
14793         {
14794           rendererPrototypes = {
14795             attributeIdentifier = {
14796               \seq_put_right:Nn
14797                 \l_@@_header_identifiers_seq
14798                 { ##1 }
14799             },
14800           },
14801         },
14802       },
14803       headerAttributeContextEnd = {
14804         \seq_map_inline:Nn
14805           \l_@@_header_identifiers_seq
14806           { \label { ##1 } }
14807         \seq_clear:N
14808           \l_@@_header_identifiers_seq
14809       },
14810     },
14811   }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

14812 \bool_new:N
14813   \l_@@_header_unnumbered_bool
14814 \markdownSetup
14815   {
14816     rendererPrototypes = {
14817       headerAttributeContextBegin += {
14818         \markdownSetup
14819         {
14820           rendererPrototypes = {

```

```

14821         attributeClassName = {
14822             \bool_if:nT
14823             {
14824                 \str_if_eq_p:nn
14825                     { ##1 }
14826                     { unnumbered } &&
14827                 ! \l_@@_header_unnumbered_bool
14828             }
14829             {
14830                 \group_begin:
14831                 \bool_set_true:N
14832                     \l_@@_header_unnumbered_bool
14833                 \c@secnumdepth = 0
14834                 \markdownSetup
14835                     {
14836                         rendererPrototypes = {
14837                             sectionBegin = {
14838                                 \group_begin:
14839                                 },
14840                             sectionEnd = {
14841                                 \group_end:
14842                                 },
14843                             },
14844                         }
14845                     },
14846                 },
14847             },
14848         },
14849     },
14850 },
14851 }
14852 \ExplSyntaxOff
14853 \markdownSetup{rendererPrototypes={
14854     superscript = {\textsuperscript{#1}},
14855     subscript = {\textsubscript{#1}},
14856     blockQuoteBegin = {\begin{quotation}},
14857     blockQuoteEnd = {\end{quotation}},
14858     inputVerbatim = {\VerbatimInput{#1}},
14859     thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
14860     note = {\footnote{#1}}}}

```

3.3.4.3 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

14861 \RequirePackage{ltxcmds}
14862 \ExplSyntaxOn
14863 \cs_gset_protected:Npn

```

```

14864 \markdownRendererInputFencedCodePrototype#1#2#3
14865 {
14866   \tl_if_empty:nTF
14867     { #2 }
14868     { \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.

```

14869   {
14870     \regex_extract_once:nnN
14871       { \w* }
14872       { #2 }
14873     \l_tmpa_seq
14874     \seq_pop_left:NN
14875     \l_tmpa_seq
14876     \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

14877   \ltx@ifpackageloaded
14878     { minted }
14879     {
14880       \catcode`\%=14\relax
14881       \catcode`\#=6\relax
14882       \exp_args:NV
14883         \inputminted
14884         \l_tmpa_tl
14885         { #1 }
14886       \catcode`\%=12\relax
14887       \catcode`\#=12\relax
14888     }
14889     {

```

When the listings package is loaded, use it for syntax highlighting.

```

14890   \ltx@ifpackageloaded
14891     { listings }
14892     { \lstinputlisting[language=\l_tmpa_tl]{#1} }

```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```

14893       { \markdownRendererInputFencedCode{#1}{}} }
14894     }
14895   }
14896 }
14897 \ExplSyntaxOff

```

Support the nesting of strong emphasis.

```

14898 \ExplSyntaxOn
14899 \def\markdownLATEXStrongEmphasis#1{%
14900   \str_if_in:NnTF

```

```

14901 \f@series
14902 { b }
14903 { \textnormal{#1} }
14904 { \textbf{#1} }
14905 }
14906 \ExplSyntaxOff
14907 \markdownSetup{rendererPrototypes={strongEmphasis={%
14908 \protect\markdownLATEXStrongEmphasis{#1}}}}

```

Support L^AT_EX document classes that do not provide chapters.

```

14909 \@ifundefined{chapter}{%
14910 \markdownSetup{rendererPrototypes = {
14911 headingOne = {\section{#1}},
14912 headingTwo = {\subsection{#1}},
14913 headingThree = {\subsubsection{#1}},
14914 headingFour = {\paragraph{#1}},
14915 headingFive = {\subparagraph{#1}}}}
14916 }{%
14917 \markdownSetup{rendererPrototypes = {
14918 headingOne = {\chapter{#1}},
14919 headingTwo = {\section{#1}},
14920 headingThree = {\subsection{#1}},
14921 headingFour = {\subsubsection{#1}},
14922 headingFive = {\paragraph{#1}},
14923 headingSix = {\subparagraph{#1}}}}
14924 }%

```

3.3.4.4 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

14925 \markdownSetup{
14926   rendererPrototypes = {
14927     ulItem = {%
14928       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUItem
14929     },
14930   },
14931 }
14932 \def\markdownLaTeXUItem{%
14933   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
14934     \item[\markdownLaTeXCheckbox]%
14935     \expandafter\@gobble
14936   \else
14937     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
14938       \item[\markdownLaTeXCheckbox]%
14939       \expandafter\expandafter\expandafter\@gobble
14940     \else
14941       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox

```

```

14942     \item[\markdownLaTeXCheckbox]%
14943     \expandafter\expandafter\expandafter\expandafter
14944     \expandafter\expandafter\expandafter\@gobble
14945     \else
14946     \item{}%
14947     \fi
14948 \fi
14949 \fi
14950 }

```

3.3.4.5 HTML elements

If the `html` option is enabled and we are using `TeX4ht`³⁶, we will pass HTML elements to the output HTML document unchanged.

```

14951 \@ifundefined{HCode}{}{
14952   \markdownSetup{
14953     rendererPrototypes = {
14954       inlineHtmlTag = {%
14955         \ifvmode
14956         \IgnorePar
14957         \EndP
14958         \fi
14959         \HCode{#1}%
14960       },
14961       inputBlockHtmlElement = {%
14962         \ifvmode
14963         \IgnorePar
14964         \fi
14965         \EndP
14966         \special{t4ht*<#1}%
14967         \par
14968         \ShowPar
14969       },
14970     },
14971   }
14972 }

```

3.3.4.6 Citations

Here is a basic implementation for citations that uses the `LATEX` `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `BibLATEX` `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

14973 \newcount\markdownLaTeXCitationsCounter
14974

```

³⁶See <https://tug.org/tex4ht/>.

```

14975 % Basic implementation
14976 \long\def@gobblethree#1#2#3{}%
14977 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
14978   \advance\markdownLaTeXCitationsCounter by 1\relax
14979   \ifx\relax#4\relax
14980     \ifx\relax#5\relax
14981       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14982         \relax
14983         \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
14984         \expandafter\expandafter\expandafter
14985         \expandafter\expandafter\expandafter\expandafter
14986         \@gobblethree
14987       \fi
14988     \else% Before a postnote (#5), dump the accumulator
14989       \ifx\relax#1\relax\else
14990         \cite{#1}%
14991       \fi
14992       \cite[#5]{#6}%
14993       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14994         \relax
14995       \else
14996         \expandafter\expandafter\expandafter
14997         \expandafter\expandafter\expandafter\expandafter
14998         \expandafter\expandafter\expandafter
14999         \expandafter\expandafter\expandafter\expandafter
15000         \markdownLaTeXBasicCitations
15001       \fi
15002       \expandafter\expandafter\expandafter
15003       \expandafter\expandafter\expandafter\expandafter{%
15004       \expandafter\expandafter\expandafter
15005       \expandafter\expandafter\expandafter\expandafter}%
15006       \expandafter\expandafter\expandafter
15007       \expandafter\expandafter\expandafter\expandafter{%
15008       \expandafter\expandafter\expandafter
15009       \expandafter\expandafter\expandafter\expandafter}%
15010       \expandafter\expandafter\expandafter
15011       \@gobblethree
15012     \fi
15013   \else% Before a prenote (#4), dump the accumulator
15014     \ifx\relax#1\relax\else
15015       \cite{#1}%
15016     \fi
15017     \ifnum\markdownLaTeXCitationsCounter>1\relax
15018       \space % Insert a space before the prenote in later citations
15019     \fi
15020     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
15021     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal

```

```

15022 \relax
15023 \else
15024 \expandafter\expandafter\expandafter
15025 \expandafter\expandafter\expandafter\expandafter
15026 \markdownLaTeXBasicCitations
15027 \fi
15028 \expandafter\expandafter\expandafter{%
15029 \expandafter\expandafter\expandafter}%
15030 \expandafter\expandafter\expandafter{%
15031 \expandafter\expandafter\expandafter}%
15032 \expandafter
15033 \@gobblethree
15034 \fi\markdownLaTeXBasicCitations{#1#2#6},}
15035 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
15036
15037 % Natbib implementation
15038 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
15039 \advance\markdownLaTeXCitationsCounter by 1\relax
15040 \ifx\relax#3\relax
15041 \ifx\relax#4\relax
15042 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15043 \relax
15044 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
15045 \expandafter\expandafter\expandafter
15046 \expandafter\expandafter\expandafter\expandafter
15047 \@gobbletwo
15048 \fi
15049 \else% Before a postnote (#4), dump the accumulator
15050 \ifx\relax#1\relax\else
15051 \citep{#1}%
15052 \fi
15053 \citep[] [#4]{#5}%
15054 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15055 \relax
15056 \else
15057 \expandafter\expandafter\expandafter
15058 \expandafter\expandafter\expandafter\expandafter
15059 \expandafter\expandafter\expandafter
15060 \expandafter\expandafter\expandafter\expandafter
15061 \markdownLaTeXNatbibCitations
15062 \fi
15063 \expandafter\expandafter\expandafter
15064 \expandafter\expandafter\expandafter\expandafter{%
15065 \expandafter\expandafter\expandafter
15066 \expandafter\expandafter\expandafter\expandafter}%
15067 \expandafter\expandafter\expandafter
15068 \@gobbletwo

```

```

15069     \fi
15070 \else% Before a prenote (#3), dump the accumulator
15071     \ifx\relax#1\relax\relax\else
15072         \citep{#1}%
15073     \fi
15074     \citep[#3][#4]{#5}%
15075     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15076     \relax
15077     \else
15078         \expandafter\expandafter\expandafter
15079         \expandafter\expandafter\expandafter\expandafter
15080         \markdownLaTeXNatbibCitations
15081     \fi
15082     \expandafter\expandafter\expandafter{%
15083     \expandafter\expandafter\expandafter}%
15084     \expandafter
15085     \@gobbletwo
15086     \fi\markdownLaTeXNatbibCitations{#1,#5}}
15087 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
15088     \advance\markdownLaTeXCitationsCounter by 1\relax
15089     \ifx\relax#3\relax
15090     \ifx\relax#4\relax
15091         \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15092         \relax
15093         \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
15094         \expandafter\expandafter\expandafter
15095         \expandafter\expandafter\expandafter\expandafter
15096         \@gobbletwo
15097     \fi
15098     \else% After a prenote or a postnote, dump the accumulator
15099     \ifx\relax#1\relax\else
15100         \citet{#1}%
15101     \fi
15102     , \citet[#3][#4]{#5}%
15103     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15104     \relax
15105     ,
15106     \else
15107         \ifnum
15108             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15109         \relax
15110         ,
15111     \fi
15112     \fi
15113     \expandafter\expandafter\expandafter
15114     \expandafter\expandafter\expandafter\expandafter
15115     \markdownLaTeXNatbibTextCitations

```



```

15116     \expandafter\expandafter\expandafter
15117     \expandafter\expandafter\expandafter\expandafter{%
15118     \expandafter\expandafter\expandafter
15119     \expandafter\expandafter\expandafter\expandafter}%
15120     \expandafter\expandafter\expandafter
15121     \@gobbletwo
15122     \fi
15123 \else% After a prenote or a postnote, dump the accumulator
15124     \ifx\relax#1\relax\relax\else
15125         \citet{#1}%
15126     \fi
15127     , \citet[#3][#4]{#5}%
15128     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
15129     \relax
15130     ,
15131     \else
15132         \ifnum
15133             \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
15134         \relax
15135         ,
15136         \fi
15137     \fi
15138     \expandafter\expandafter\expandafter
15139     \markdownLaTeXNatbibTextCitations
15140     \expandafter\expandafter\expandafter{%
15141     \expandafter\expandafter\expandafter}%
15142     \expandafter
15143     \@gobbletwo
15144     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
15145
15146 % BibLaTeX implementation
15147 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
15148     \advance\markdownLaTeXCitationsCounter by 1\relax
15149     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15150     \relax
15151     \autocites#1[#3][#4]{#5}%
15152     \expandafter\@gobbletwo
15153     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
15154 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
15155     \advance\markdownLaTeXCitationsCounter by 1\relax
15156     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
15157     \relax
15158     \textcites#1[#3][#4]{#5}%
15159     \expandafter\@gobbletwo
15160     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
15161
15162 \markdownSetup{rendererPrototypes = {

```

```

15163 cite = {%
15164     \markdownLaTeXCitationsCounter=1%
15165     \def\markdownLaTeXCitationsTotal{#1}%
15166     \@ifundefined{autocites}{%
15167         \@ifundefined{citep}{%
15168             \expandafter\expandafter\expandafter
15169             \markdownLaTeXBasicCitations
15170             \expandafter\expandafter\expandafter{%
15171                 \expandafter\expandafter\expandafter}%
15172             \expandafter\expandafter\expandafter{%
15173                 \expandafter\expandafter\expandafter}%
15174         }{%
15175             \expandafter\expandafter\expandafter
15176             \markdownLaTeXNatbibCitations
15177             \expandafter\expandafter\expandafter{%
15178                 \expandafter\expandafter\expandafter}%
15179         }%
15180     }{%
15181         \expandafter\expandafter\expandafter
15182         \markdownLaTeXBibLaTeXCitations
15183         \expandafter{\expandafter}%
15184     }},
15185 textCite = {%
15186     \markdownLaTeXCitationsCounter=1%
15187     \def\markdownLaTeXCitationsTotal{#1}%
15188     \@ifundefined{autocites}{%
15189         \@ifundefined{citep}{%
15190             \expandafter\expandafter\expandafter
15191             \markdownLaTeXBasicTextCitations
15192             \expandafter\expandafter\expandafter{%
15193                 \expandafter\expandafter\expandafter}%
15194             \expandafter\expandafter\expandafter{%
15195                 \expandafter\expandafter\expandafter}%
15196         }{%
15197             \expandafter\expandafter\expandafter
15198             \markdownLaTeXNatbibTextCitations
15199             \expandafter\expandafter\expandafter{%
15200                 \expandafter\expandafter\expandafter}%
15201         }%
15202     }{%
15203         \expandafter\expandafter\expandafter
15204         \markdownLaTeXBibLaTeXTextCitations
15205         \expandafter{\expandafter}%
15206     }}}}

```

3.3.4.7 Links

Here is an implementation for hypertext links and relative references.

```

15207 \RequirePackage{url}
15208 \RequirePackage{expl3}
15209 \ExplSyntaxOn
15210 \cs_gset_protected:Npn
15211   \markdownRendererLinkPrototype
15212   #1#2#3#4
15213   {
15214     \tl_set:Nn \l_tmpa_tl { #1 }
15215     \tl_set:Nn \l_tmpb_tl { #2 }
15216     \bool_set:Nn
15217       \l_tmpa_bool
15218       {
15219         \tl_if_eq_p:NN
15220           \l_tmpa_tl
15221           \l_tmpb_tl
15222       }
15223     \tl_set:Nn \l_tmpa_tl { #4 }
15224     \bool_set:Nn
15225       \l_tmpb_bool
15226       {
15227         \tl_if_empty_p:N
15228           \l_tmpa_tl
15229       }

```

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.

```

15230   \bool_if:nTF
15231     {
15232       \l_tmpa_bool && \l_tmpb_bool
15233     }
15234     {
15235       \markdownLaTeXRendererAutolink { #2 } { #3 }
15236     }{
15237       \markdownLaTeXRendererDirectOrIndirectLink
15238         { #1 } { #2 } { #3 } { #4 }
15239     }
15240   }
15241 \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.

```

15242   \tl_set:Nn
15243     \l_tmpa_tl
15244     { #2 }
15245   \tl_trim_spaces:N

```

```

15246     \l_tmpa_tl
15247     \tl_set:Nx
15248     \l_tmpb_tl
15249     {
15250         \tl_range:Nnn
15251         \l_tmpa_tl
15252         { 1 }
15253         { 1 }
15254     }
15255     \str_if_eq:NNTF
15256     \l_tmpb_tl
15257     \c_hash_str
15258     {
15259         \tl_set:Nx
15260         \l_tmpb_tl
15261         {
15262             \tl_range:Nnn
15263             \l_tmpa_tl
15264             { 2 }
15265             { -1 }
15266         }
15267         \exp_args:NV
15268         \ref
15269         \l_tmpb_tl
15270     }{
15271         \url { #2 }
15272     }
15273 }
15274 \ExplSyntaxOff
15275 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
15276     #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.

```

15277 \newcount\markdownLaTeXRowCount
15278 \newcount\markdownLaTeXRowTotal
15279 \newcount\markdownLaTeXColumnCounter
15280 \newcount\markdownLaTeXColumnTotal
15281 \newtoks\markdownLaTeXTable
15282 \newtoks\markdownLaTeXTableAlignment
15283 \newtoks\markdownLaTeXTableEnd
15284 \AtBeginDocument{%
15285     \ifpackageloaded{booktabs}{%
15286         \def\markdownLaTeXTopRule{\toprule}%
15287         \def\markdownLaTeXMidRule{\midrule}%

```

```

15288     \def\markdownLaTeXBottomRule{\bottomrule}%
15289   }{%
15290     \def\markdownLaTeXTopRule{\hline}%
15291     \def\markdownLaTeXMidRule{\hline}%
15292     \def\markdownLaTeXBottomRule{\hline}%
15293   }%
15294 }
15295 \markdownSetup{rendererPrototypes={
15296   table = {%
15297     \markdownLaTeXTable={}%
15298     \markdownLaTeXTableAlignment={}%
15299     \markdownLaTeXTableEnd={%
15300       \markdownLaTeXBottomRule
15301       \end{tabular}}%
15302     \ifx\empty#1\empty\else
15303       \addto@hook\markdownLaTeXTable{%
15304         \begin{table}
15305         \centering}%
15306       \addto@hook\markdownLaTeXTableEnd{%
15307         \caption{#1}}%
15308     \fi
15309   }
15310 }}

```

If the `tableAttributes` option is enabled, we will register any identifiers, so that they can be used as L^AT_EX labels for referencing tables.

```

15311 \ExplSyntaxOn
15312 \seq_new:N
15313   \l_@@_table_identifiers_seq
15314 \markdownSetup {
15315   rendererPrototypes = {
15316     table += {
15317       \seq_map_inline:Nn
15318         \l_@@_table_identifiers_seq
15319         {
15320           \addto@hook
15321             \markdownLaTeXTableEnd
15322             { \label { ##1 } }
15323         }
15324     },
15325   }
15326 }
15327 \markdownSetup {
15328   rendererPrototypes = {
15329     tableAttributeContextBegin = {
15330       \group_begin:
15331       \markdownSetup {

```

```

15332     rendererPrototypes = {
15333         attributeIdentifier = {
15334             \seq_put_right:Nn
15335             \l_@@_table_identifiers_seq
15336             { ##1 }
15337         },
15338     },
15339 }
15340 },
15341 tableAttributeContextEnd = {
15342     \group_end:
15343 },
15344 },
15345 }
15346 \ExplSyntaxOff
15347 \markdownSetup{rendererPrototypes={
15348     table += {%
15349         \ifx\empty#1\empty\else
15350             \addto@hook\markdownLaTeXTableEnd{%
15351                 \end{table}}%
15352         \fi
15353         \addto@hook\markdownLaTeXTable{\begin{tabular}}%
15354         \markdownLaTeXRowCounter=0%
15355         \markdownLaTeXRowTotal=#2%
15356         \markdownLaTeXColumnTotal=#3%
15357         \markdownLaTeXRenderTableRow
15358     }
15359 }}
15360 \def\markdownLaTeXRenderTableRow#1{%
15361     \markdownLaTeXColumnCounter=0%
15362     \ifnum\markdownLaTeXRowCounter=0\relax
15363         \markdownLaTeXReadAlignments#1%
15364         \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
15365             \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
15366                 \the\markdownLaTeXTableAlignment}}%
15367         \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
15368     \else
15369         \markdownLaTeXRenderTableCell#1%
15370     \fi
15371     \ifnum\markdownLaTeXRowCounter=1\relax
15372         \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
15373     \fi
15374     \advance\markdownLaTeXRowCounter by 1\relax
15375     \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
15376         \the\markdownLaTeXTable
15377         \the\markdownLaTeXTableEnd
15378         \expandafter\@gobble

```

```

15379 \fi\markdownLaTeXRenderTableRow}
15380 \def\markdownLaTeXReadAlignments#1{%
15381 \advance\markdownLaTeXColumnCounter by 1\relax
15382 \if#1d%
15383 \addto@hook\markdownLaTeXTableAlignment{1}%
15384 \else
15385 \addto@hook\markdownLaTeXTableAlignment{#1}%
15386 \fi
15387 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
15388 \expandafter\@gobble
15389 \fi\markdownLaTeXReadAlignments}
15390 \def\markdownLaTeXRenderTableCell#1{%
15391 \advance\markdownLaTeXColumnCounter by 1\relax
15392 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
15393 \addto@hook\markdownLaTeXTable{#1&}%
15394 \else
15395 \addto@hook\markdownLaTeXTable{#1\\}%
15396 \expandafter\@gobble
15397 \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.

```

15398
15399 \markdownIfOption{lineBlocks}{%
15400 \RequirePackage{verse}
15401 \markdownSetup{rendererPrototypes={
15402 lineBlockBegin = {%
15403 \begingroup
15404 \def\markdownRendererHardLineBreak{\\}%
15405 \begin{verse}%
15406 },
15407 lineBlockEnd = {%
15408 \end{verse}%
15409 \endgroup
15410 },
15411 }}
15412 }{}
15413

```

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.

```

15414 \ExplSyntaxOn

```

```

15415 \keys_define:nn
15416   { markdown/jekyllData }
15417   {
15418     author .code:n = { \author{#1} },
15419     date   .code:n = { \date{#1}   },
15420     title  .code:n = { \title{#1}  },
15421   }

```

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.

```

15422 \markdownSetup{
15423   rendererPrototypes = {
15424     jekyllDataEnd = {
15425       \AddToHook{begindocument/end}{\maketitle}
15426     },
15427   },
15428 }

```

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.

```

15429 \@_if_option:nT
15430   { mark }
15431   {
15432     \sys_if_engine luatex:TF
15433     {
15434       \RequirePackage
15435         { luacolor }
15436       \RequirePackage
15437         { lua-ul }
15438       \markdownSetup
15439         {
15440           rendererPrototypes = {
15441             mark = {
15442               \highLight
15443                 { #1 }
15444             },
15445           }
15446         }
15447     }
15448     {
15449       \RequirePackage
15450         { xcolor }
15451       \RequirePackage

```



```

15452     { soul }
15453   \markdownSetup
15454   {
15455     rendererPrototypes = {
15456       mark = {
15457         \hl
15458         { #1 }
15459       },
15460     }
15461   }
15462 }
15463 }

```

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.

```

15464 \@@_if_option:nT
15465 { strikeThrough }
15466 {
15467   \sys_if_engine luatex:TF
15468   {
15469     \RequirePackage
15470     { lua-ul }
15471     \markdownSetup
15472     {
15473       rendererPrototypes = {
15474         strikeThrough = {
15475           \strikeThrough
15476           { #1 }
15477         },
15478       }
15479     }
15480   }
15481   {
15482     \RequirePackage
15483     { soul }
15484     \markdownSetup
15485     {
15486       rendererPrototypes = {
15487         strikeThrough = {
15488           \st
15489           { #1 }
15490         },
15491       }
15492     }
15493   }

```

```
15494 }
```

3.3.4.13 Images and their attributes

We define images to be rendered as floating figures using the command `\includegraphics`, where the image label is the alt text and the image title is the caption of the figure.

If the `linkAttributes` option is enabled, we will make attributes in the form `<key>=<value>` set the corresponding keys of the `graphicx` package to the corresponding values and we will register any identifiers, so that they can be used as \LaTeX labels for referencing figures.

```
15495 \ExplSyntaxOn
15496 \seq_new:N
15497   \l_@@_image_identifiers_seq
15498 \markdownSetup {
15499   rendererPrototypes = {
15500     image = {
15501       \begin { figure }
15502         \begin { center }
15503           \includegraphics
15504             [ alt = { #1 } ]
15505             { #3 }
15506           \tl_if_empty:nF
15507             { #4 }
15508             { \caption { #4 } }
15509           \seq_map_inline:Nn
15510             \l_@@_image_identifiers_seq
15511             { \label { ##1 } }
15512         \end { center }
15513       \end { figure }
15514     },
15515   }
15516 }
15517 \@@_if_option:nT
15518   { linkAttributes }
15519   {
15520     \RequirePackage { graphicx }
15521     \markdownSetup {
15522       rendererPrototypes = {
15523         imageAttributeContextBegin = {
15524           \group_begin:
15525           \markdownSetup {
15526             rendererPrototypes = {
15527               attributeIdentifier = {
15528                 \seq_put_right:Nn
15529                 \l_@@_image_identifiers_seq
15530                 { ##1 }

```

```

15531         },
15532         attributeKeyValue = {
15533             \setkeys
15534             { Gin }
15535             { { ##1 } = { ##2 } }
15536         },
15537     },
15538 }
15539 },
15540 imageAttributeContextEnd = {
15541     \group_end:
15542 },
15543 },
15544 }
15545 }
15546 \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`.

```

15547 \ExplSyntaxOn
15548 \cs_new:Nn
15549   \@@_luaxml_print_html:n
15550   {
15551     \luabridge_now:n
15552     {
15553       local~input_file = assert(io.open(" #1 ", "r"))
15554       local~input = assert(input_file:read("*a"))
15555       assert(input_file:close())
15556       input = "<body>" .. input .. "</body>"
15557       local~dom = require("luaxml-domobject").html_parse(input)
15558       local~output = require("luaxml-htmltemplates"):process_dom(dom)
15559       print(output)
15560     }
15561   }
15562 \cs_gset_protected:Npn
15563   \markdownRendererInputRawInlinePrototype#1#2
15564   {
15565     \str_case:nnF
15566     { #2 }
15567     {
15568       { latex }
15569       {
15570         \@@_plain_tex_default_input_raw_inline:nn

```

```

15571         { #1 }
15572         { tex }
15573     }
15574     { html }
15575     {

```

If we are using $\text{T}\text{E}\text{X}4\text{ht}$ ³⁷, we will pass HTML elements to the output HTML document unchanged.

```

15576         \cs_if_exist:NTF
15577         \HCode
15578         {
15579             \if_mode_vertical:
15580             \IgnorePar
15581             \EndP
15582             \fi:
15583             \special
15584             { t4ht* < #1 }
15585         }
15586         {
15587             \@@_luaxml_print_html:n
15588             { #1 }
15589         }
15590     }
15591 }
15592 {
15593     \@@_plain_tex_default_input_raw_inline:nn
15594     { #1 }
15595     { #2 }
15596 }
15597 }
15598 \cs_gset_protected:Npn
15599 \markdownRendererInputRawBlockPrototype#1#2
15600 {
15601     \str_case:nnF
15602     { #2 }
15603     {
15604         { latex }
15605         {
15606             \@@_plain_tex_default_input_raw_block:nn
15607             { #1 }
15608             { tex }
15609         }
15610     { html }
15611     {

```

³⁷See <https://tug.org/tex4ht/>.

If we are using $\text{\TeX}4\text{ht}$ ³⁸, we will pass HTML elements to the output HTML document unchanged.

```

15612         \cs_if_exist:NTF
15613         \HCode
15614         {
15615             \if_mode_vertical:
15616             \IgnorePar
15617             \fi:
15618             \EndP
15619             \special
15620             { t4ht* < #1 }
15621             \par
15622             \ShowPar
15623         }
15624         {
15625             \@@_luaxml_print_html:n
15626             { #1 }
15627         }
15628     }
15629 }
15630 {
15631     \@@_plain_tex_default_input_raw_block:nn
15632     { #1 }
15633     { #2 }
15634 }
15635 }

```

3.3.4.15 Bracketed spans

If the `bracketedSpans` option is enabled, we will register any identifiers, so that they can be used as \LaTeX labels for referencing the last \LaTeX counter that has been incremented in e.g. ordered lists.

```

15636 \seq_new:N
15637 \l_@@_bracketed_span_identifiers_seq
15638 \markdownSetup {
15639     rendererPrototypes = {
15640         bracketedSpanAttributeContextBegin = {
15641             \group_begin:
15642             \markdownSetup {
15643                 rendererPrototypes = {
15644                     attributeIdentifier = {
15645                         \seq_put_right:Nn
15646                         \l_@@_bracketed_span_identifiers_seq
15647                         { ##1 }
15648                     },

```

³⁸See <https://tug.org/tex4ht/>.

```

15649     },
15650   }
15651 },
15652 bracketedSpanAttributeContextEnd = {
15653   \seq_map_inline:Nn
15654   \l_@@_bracketed_span_identifiers_seq
15655   { \label { ##1 } }
15656   \group_end:
15657 },
15658 },
15659 }
15660 \ExplSyntaxOff
15661 \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.

```

15662 \newcommand\markdownMakeOther{%
15663   \count0=128\relax
15664   \loop
15665     \catcode\count0=11\relax
15666     \advance\count0 by 1\relax
15667   \ifnum\count0<256\repeat}%

```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain T_EX implementation. As a consequence, we can directly reuse the existing plain T_EX implementation after supplying the missing plain T_EX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the `filecontents` L^AT_EX package.

```

15668 \def\markdownMakeOther{%
15669   \count0=128\relax
15670   \loop
15671     \catcode\count0=11\relax
15672     \advance\count0 by 1\relax
15673   \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.

```
15674 \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).

```
15675 \long\def\inputmarkdown{%
15676   \dosingleempty
15677   \doinputmarkdown}%
15678 \long\def\doinputmarkdown[#1]#2{%
15679   \begingroup
15680     \iffirstargument
15681       \setupmarkdown[#1]%
15682       \fi
15683     \markdownInput{#2}%
15684   \endgroup}%
15685 \long\def\inputyaml{%
15686   \dosingleempty
15687   \doinputyaml}%
15688 \long\def\doinputyaml[#1]#2{%
15689   \doinputmarkdown
15690   [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%
```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s TeX, trailing spaces are removed very early on when a line is being put to the input buffer. [16, sec. 31]. According to Eijkhout [17, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to suppress this behavior in (Lua)TeX, but ConTeXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTeXt MkIV and therefore to insert hard line breaks into markdown text.

```
15691 \startluacode
15692   document.markdown_buffering = false
15693   local function preserve_trailing_spaces(line)
15694     if document.markdown_buffering then
15695       line = line:gsub("[ \t][ \t]$", "\t\t")
15696     end
15697     return line
15698   end
15699   resolvers.installinputlinehandler(preserve_trailing_spaces)
15700 \stopluacode
15701 \begingroup
15702   \catcode`\|=0%
15703   \catcode`\|=12%
```

```

15704 |gdef|startmarkdown{%
15705   |ctxlua{document.markdown_buffering = true}%
15706   |markdownReadAndConvert{\stopmarkdown}%
15707   {|\stopmarkdown}}%
15708 |gdef|stopmarkdown{%
15709   |ctxlua{document.markdown_buffering = false}%
15710   |markdownEnd}%
15711 |gdef|startyaml{%
15712   |begingroup
15713   |ctxlua{document.markdown_buffering = true}%
15714   |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
15715   |markdownReadAndConvert{\stopyaml}%
15716   {|\stopyaml}}%
15717 |gdef|stopyaml{%
15718   |ctxlua{document.markdown_buffering = false}%
15719   |yamlEnd}%
15720 |endgroup

```

3.4.2 Themes

This section overrides the plain \TeX implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in Con \TeX t themes provided with the Markdown package.

```

15721 \ExplSyntaxOn
15722 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
15723 \prop_new:N \g_@@_context_loaded_themes_versions_prop
15724 \cs_gset:Nn
15725   \@@_load_theme:nnn
15726   {

```

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.

```

15727   \bool_if:nTF
15728     {
15729     \bool_lazy_or_p:nn
15730       {
15731         \prop_if_in_p:Nn
15732           \g_@@_context_built_in_themes_prop
15733           { #1 }
15734       }
15735     {
15736       \file_if_exist_p:n
15737         { t - markdown theme #3.tex }
15738     }

```



```

15739     }
15740     {
15741       \prop_get:NnNTF
15742       \g_@@_context_loaded_themes_linenos_prop
15743       { #1 }
15744       \l_tmpa_tl
15745       {
15746         \prop_get:NnN
15747         \g_@@_context_loaded_themes_versions_prop
15748         { #1 }
15749         \l_tmpb_tl
15750         \str_if_eq:nVTF
15751         { #2 }
15752         \l_tmpb_tl
15753         {
15754           \msg_warning:nnnVn
15755           { markdown }
15756           { repeatedly-loaded-context-theme }
15757           { #1 }
15758           \l_tmpa_tl
15759           { #2 }
15760         }
15761         {
15762           \msg_error:nnnnVV
15763           { markdown }
15764           { different-versions-of-context-theme }
15765           { #1 }
15766           { #2 }
15767           \l_tmpb_tl
15768           \l_tmpa_tl
15769         }
15770       }
15771     {
15772       \prop_gput:Nnx
15773       \g_@@_context_loaded_themes_linenos_prop
15774       { #1 }
15775       { \tex_the:D \tex_inputlineno:D }
15776       \prop_gput:Nnn
15777       \g_@@_context_loaded_themes_versions_prop
15778       { #1 }
15779       { #2 }

```

Load built-in plain T_EX themes from the prop `\g_@@_context_built_in_themes_prop` and from the filesystem otherwise.

```

15780     \prop_if_in:NnTF
15781     \g_@@_context_built_in_themes_prop
15782     { #1 }

```

```

15783         {
15784             \msg_info:nnnn
15785             { markdown }
15786             { loading-built-in-context-theme }
15787             { #1 }
15788             { #2 }
15789             \prop_item:Nn
15790             \g_@@_context_built_in_themes_prop
15791             { #1 }
15792         }
15793         {
15794             \msg_info:nnnn
15795             { markdown }
15796             { loading-context-theme }
15797             { #1 }
15798             { #2 }
15799             \usemodule
15800             [ t ]
15801             [ markdown theme #3 ]
15802         }
15803     }
15804 }
15805 {
15806     \@@_plain_tex_load_theme:nnn
15807     { #1 }
15808     { #2 }
15809     { #3 }
15810 }
15811 }
15812 \msg_new:nnn
15813 { markdown }
15814 { loading-built-in-context-theme }
15815 { Loading~version~#2~of~built-in~ConTeXt~Markdown~theme~#1 }
15816 \msg_new:nnn
15817 { markdown }
15818 { loading-context-theme }
15819 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
15820 \msg_new:nnn
15821 { markdown }
15822 { repeatedly-loaded-context-theme }
15823 {
15824     Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
15825     loaded~on~line~#2,~not~loading~it~again
15826 }
15827 \msg_new:nnn
15828 { markdown }
15829 { different-versions-of-context-theme }

```

```

15830 {
15831   Tried-to-load-version-#2-of-ConTeXt-Markdown-theme-#1~
15832   but-version-#3-has-already-been-loaded-on-line-#4
15833 }
15834 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain T_EX theme with the default definitions for plain T_EX:

```
15835 \markdownLoadPlainTeXTheme
```

Next, the ConTeXt theme overrides some of the plain T_EX 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.

```

15836 \markdownIfOption{plain}{\iffalse}{\iftrue}
15837 \def\markdownRendererHardLineBreakPrototype{\blank}%
15838 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
15839 \def\markdownRendererRightBracePrototype{\textbraceright}%
15840 \def\markdownRendererDollarSignPrototype{\textdollar}%
15841 \def\markdownRendererPercentSignPrototype{\percent}%
15842 \def\markdownRendererUnderscorePrototype{\textunderscore}%
15843 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
15844 \def\markdownRendererBackslashPrototype{\textbackslash}%
15845 \def\markdownRendererTildePrototype{\textasciitilde}%
15846 \def\markdownRendererPipePrototype{\char`|}%
15847 \def\markdownRendererLinkPrototype#1#2#3#4{%
15848   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
15849   \fi\tt<\hyphenatedurl{#3}>}}%
15850 \usemodule[database]
15851 \defineseparatedlist
15852   [MarkdownConTeXtCSV]
15853   [separator={,},
15854   before=\bTABLE,after=\eTABLE,
15855   first=\bTR,last=\eTR,
15856   left=\bTD,right=\eTD]
15857 \def\markdownConTeXtCSV{csv}
15858 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
15859   \def\markdownConTeXtCSV@arg{#1}%
15860   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
15861     \placetable[] [tab:#1]{#4}{%
15862       \processseparatedfile[MarkdownConTeXtCSV][#3]}%
15863   \else
15864     \markdownInput{#3}%

```

```

15865 \fi}%
15866 \def\markdownRendererImagePrototype#1#2#3#4{%
15867 \placefigure [] []{#4}{\externalfigure[#3]}}%
15868 \def\markdownRendererUlBeginPrototype{\startitemize}%
15869 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
15870 \def\markdownRendererUlItemPrototype{\item}%
15871 \def\markdownRendererUlEndPrototype{\stopitemize}%
15872 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
15873 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
15874 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
15875 \def\markdownRendererOlItemPrototype{\item}%
15876 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
15877 \def\markdownRendererOlEndPrototype{\stopitemize}%
15878 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
15879 \definedescription
15880 [MarkdownConTeXtDlItemPrototype]
15881 [location=hanging,
15882 margin=standard,
15883 headstyle=bold]%
15884 \definestartstop
15885 [MarkdownConTeXtDlPrototype]
15886 [before=\blank,
15887 after=\blank]%
15888 \definestartstop
15889 [MarkdownConTeXtDlTightPrototype]
15890 [before=\blank\startpacked,
15891 after=\stoppacked\blank]%
15892 \def\markdownRendererDlBeginPrototype{%
15893 \startMarkdownConTeXtDlPrototype}%
15894 \def\markdownRendererDlBeginTightPrototype{%
15895 \startMarkdownConTeXtDlTightPrototype}%
15896 \def\markdownRendererDlItemPrototype#1{%
15897 \startMarkdownConTeXtDlItemPrototype{#1}}%
15898 \def\markdownRendererDlItemEndPrototype{%
15899 \stopMarkdownConTeXtDlItemPrototype}%
15900 \def\markdownRendererDlEndPrototype{%
15901 \stopMarkdownConTeXtDlPrototype}%
15902 \def\markdownRendererDlEndTightPrototype{%
15903 \stopMarkdownConTeXtDlTightPrototype}%
15904 \def\markdownRendererEmphasisPrototype#1{\em#1}%
15905 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
15906 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
15907 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
15908 \def\markdownRendererLineBlockBeginPrototype{%
15909 \begingroup
15910 \def\markdownRendererHardLineBreak{
15911 }%

```

```

15912     \startlines
15913 }%
15914 \def\markdownRendererLineBlockEndPrototype{%
15915     \stoptlines
15916     \endgroup
15917 }%
15918 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

15919 \ExplSyntaxOn
15920 \cs_gset:Npn
15921   \markdownRendererInputFencedCodePrototype#1#2#3
15922   {
15923     \tl_if_empty:nTF
15924       { #2 }
15925     { \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

```

```

15926   {
15927     \regex_extract_once:nnN
15928       { \w* }
15929       { #2 }
15930     \l_tmpa_seq
15931     \seq_pop_left:NN
15932     \l_tmpa_seq

```

```

15933         \l_tmpa_tl
15934         \typefile[\l_tmpa_tl][\l]{#1}
15935     }
15936 }
15937 \ExplSyntaxOff
15938 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
15939 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
15940 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
15941 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
15942 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
15943 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
15944 \def\markdownRendererThematicBreakPrototype{%
15945     \blackrule[height=1pt, width=\hsize]}%
15946 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
15947 \def\markdownRendererTickedBoxPrototype{\$ \boxtimes $}
15948 \def\markdownRendererHalfTickedBoxPrototype{\$ \boxdot $}
15949 \def\markdownRendererUntickedBoxPrototype{\$ \square $}
15950 \def\markdownRendererStrikeThroughPrototype#1{\overstrides{#1}}
15951 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
15952 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
15953 \def\markdownRendererDisplayMathPrototype#1{%
15954     \startformula#1\stopformula}%

```

3.4.3.2 Tables

There is a basic implementation of tables.

```

15955 \newcount\markdownConTeXtRowCounter
15956 \newcount\markdownConTeXtRowTotal
15957 \newcount\markdownConTeXtColumnCounter
15958 \newcount\markdownConTeXtColumnTotal
15959 \newtoks\markdownConTeXtTable
15960 \newtoks\markdownConTeXtTableFloat
15961 \def\markdownRendererTablePrototype#1#2#3{%
15962     \markdownConTeXtTable={}%
15963     \ifx\empty#1\empty
15964         \markdownConTeXtTableFloat={%
15965             \the\markdownConTeXtTable}%
15966     \else
15967         \markdownConTeXtTableFloat={%
15968             \placetable{#1}{\the\markdownConTeXtTable}}%
15969     \fi
15970     \begingroup
15971     \setupTABLE[r][each][topframe=off, bottomframe=off,
15972         leftframe=off, rightframe=off]
15973     \setupTABLE[c][each][topframe=off, bottomframe=off,
15974         leftframe=off, rightframe=off]
15975     \setupTABLE[r][1][topframe=on, bottomframe=on]

```

```

15976 \setupTABLE[r] [#1] [bottomframe=on]
15977 \markdownConTeXtRowCounter=0%
15978 \markdownConTeXtRowTotal=#2%
15979 \markdownConTeXtColumnTotal=#3%
15980 \markdownConTeXtRenderTableRow}
15981 \def\markdownConTeXtRenderTableRow#1{%
15982 \markdownConTeXtColumnCounter=0%
15983 \ifnum\markdownConTeXtRowCounter=0\relax
15984 \markdownConTeXtReadAlignments#1%
15985 \markdownConTeXtTable={\bTABLE}%
15986 \else
15987 \markdownConTeXtTable=\expandafter{%
15988 \the\markdownConTeXtTable\bTR}%
15989 \markdownConTeXtRenderTableCell#1%
15990 \markdownConTeXtTable=\expandafter{%
15991 \the\markdownConTeXtTable\eTR}%
15992 \fi
15993 \advance\markdownConTeXtRowCounter by 1\relax
15994 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
15995 \markdownConTeXtTable=\expandafter{%
15996 \the\markdownConTeXtTable\eTABLE}%
15997 \the\markdownConTeXtTableFloat
15998 \endgroup
15999 \expandafter\gobbleoneargument
16000 \fi\markdownConTeXtRenderTableRow}
16001 \def\markdownConTeXtReadAlignments#1{%
16002 \advance\markdownConTeXtColumnCounter by 1\relax
16003 \if#1d%
16004 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=right]
16005 \fi\if#1l%
16006 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=right]
16007 \fi\if#1c%
16008 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=middle]
16009 \fi\if#1r%
16010 \setupTABLE[c] [\the\markdownConTeXtColumnCounter] [align=left]
16011 \fi
16012 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16013 \else
16014 \expandafter\gobbleoneargument
16015 \fi\markdownConTeXtReadAlignments}
16016 \def\markdownConTeXtRenderTableCell#1{%
16017 \advance\markdownConTeXtColumnCounter by 1\relax
16018 \markdownConTeXtTable=\expandafter{%
16019 \the\markdownConTeXtTable\bTD#1\eTD}%
16020 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
16021 \else
16022 \expandafter\gobbleoneargument

```

```
16023 \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`.

```
16024 \ExplSyntaxOn
16025 \cs_gset:Npn
16026 \markdownRendererInputRawInlinePrototype#1#2
16027 {
16028   \str_case:nnF
16029     { #2 }
16030     {
16031       { latex }
16032       {
16033         \@@_plain_tex_default_input_raw_inline:nn
16034           { #1 }
16035           { context }
16036       }
16037     }
16038     {
16039       \@@_plain_tex_default_input_raw_inline:nn
16040         { #1 }
16041         { #2 }
16042     }
16043 }
16044 \cs_gset:Npn
16045 \markdownRendererInputRawBlockPrototype#1#2
16046 {
16047   \str_case:nnF
16048     { #2 }
16049     {
16050       { context }
16051       {
16052         \@@_plain_tex_default_input_raw_block:nn
16053           { #1 }
16054           { tex }
16055       }
16056     }
16057     {
16058       \@@_plain_tex_default_input_raw_block:nn
16059         { #1 }
16060         { #2 }
16061     }
16062 }
16063 \cs_gset_eq:NN
16064 \markdownRendererInputRawBlockPrototype
```



```

16065 \markdownRendererInputRawInlinePrototype
16066 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}`
16067 \ExplSyntaxOff
16068 \stopmodule
16069 \protect

```

At the end of the ConT_EXt module, we load the `witiko/markdown/defaults` ConT_EXt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

16070 \ExplSyntaxOn
16071 \str_if_eq:VVT
16072 \c_@@_top_layer_tl
16073 \c_@@_option_layer_context_tl
16074 {
16075   \ExplSyntaxOff
16076   \@@_if_option:nF
16077   { noDefaults }
16078   {
16079     \@@_if_option:nTF
16080     { experimental }
16081     {
16082       \@@_setup:n
16083       { theme = witiko/markdown/defaults@experimental }
16084     }
16085     {
16086       \@@_setup:n
16087       { theme = witiko/markdown/defaults }
16088     }
16089   }
16090   \ExplSyntaxOn
16091 }
16092 \ExplSyntaxOff
16093 \stopmodule
16094 \protect

```

References

- [1] LuaT_EX development team. *LuaT_EX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Frank Mittelbach, Ulrike Fischer, and L^AT_EX Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).

- [3] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [4] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [5] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [6] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [7] Donald Ervin Knuth. *The T_EXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [8] Frank Mittelbach. *The doc and shortverb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [9] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [10] Vít Starý Novotný. *Versioned Themes*. Markdown Enhancement Proposal. Oct. 13, 2024. URL: <https://github.com/Witiko/markdown/discussions/514> (visited on 10/21/2024).
- [11] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [12] Frank Mittelbach. *L^AT_EX’s hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [13] Geoffrey M. Poore. *The minted Package. Highlighted source code in L^AT_EX*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [14] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [15] Johannes Braams et al. *The L^AT_EX_{2 ϵ} Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [16] Donald Ervin Knuth. *T_EX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.

- [17] Victor Eijkhout. *TEX by Topic. A T_EXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

Index

autoIdentifiers	21, 33, 82, 98
blankBeforeBlockquote	21
blankBeforeCodeFence	22
blankBeforeDivFence	22
blankBeforeHeading	22
blankBeforeList	22
bracketedSpans	23, 84, 445
breakableBlockquotes	23
cacheDir	4, 17, 19, 59, 156, 168, 370, 383, 397
citationNbsps	23
citations	24, 87, 88
codeSpans	24
contentBlocks	20, 25, 34
contentBlocksLanguageMap	20
contentLevel	25
debugExtensions	9, 20, 26, 314
debugExtensionsFileName	20, 26
defaultOptions	10, 51, 368, 369
definitionLists	26, 92
eagerCache	17, 368
ensureJekyllData	27
entities.char_entity	215
entities.dec_entity	214
entities.hex_entity	214
entities.hex_entity_with_x_char	215
escape_minimal	219
escape_programmatic_text	219
escape_typographic_text	219
expandtabs	274
expectJekyllData	27, 27
experimental	5, 17, 415
extensions	28, 163, 319
extensions.bracketed_spans	319
extensions.citations	320

<code>extensions.content_blocks</code>	325
<code>extensions.definition_lists</code>	328
<code>extensions.fancy_lists</code>	330
<code>extensions.fenced_code</code>	336
<code>extensions.fenced_divs</code>	341
<code>extensions.header_attributes</code>	345
<code>extensions.inline_code_attributes</code>	347
<code>extensions.jekyll_data</code>	364
<code>extensions.line_blocks</code>	347
<code>extensions.link_attributes</code>	349
<code>extensions.mark</code>	348
<code>extensions.notes</code>	351
<code>extensions.pipe_table</code>	353
<code>extensions.raw_inline</code>	357
<code>extensions.strike_through</code>	358
<code>extensions.subscripts</code>	359
<code>extensions.superscripts</code>	359
<code>extensions.tex_math</code>	360
<code>fancyLists</code>	30, 109–114, 415
<code>fencedCode</code>	30, 39, 89, 96, 115, 381
<code>fencedCodeAttributes</code>	31, 82, 96
<code>fencedDiv</code>	97
<code>fencedDivs</code>	31, 41
<code>finalizeCache</code>	17, 20, 32, 32, 59, 155, 368, 369
<code>frozenCache</code>	20, 32, 59, 74, 155, 381, 383
<code>frozenCacheCounter</code>	32, 369, 405
<code>frozenCacheFileName</code>	20, 32, 59, 369
<code>gfmAutoIdentifiers</code>	21, 32, 82, 98
<code>hashEnumerators</code>	33
<code>headerAttributes</code>	33, 41, 82, 98
<code>html</code>	34, 101, 429
<code>hybrid</code>	34, 34, 40, 46, 48, 62, 75, 116, 156, 219, 275, 405
<code>inlineCodeAttributes</code>	35, 82, 90
<code>inlineNotes</code>	36
<code>\input</code>	55, 56
<code>\inputmarkdown</code>	159, 160, 161, 447
<code>inputTempFileName</code>	60, 62, 398–400, 402
<code>\inputyaml</code>	159, 161, 447
<code>iterlines</code>	273

jeekyllData	3, 27, 28, 36, 125–128, 131
\l_file_search_path_seq	404
languages_json	325, 325
lineBlocks	37, 104
linkAttributes	37, 82, 102, 106, 295, 442
mark	38, 107, 440
\markdown	151, 152, 408
markdown	150, 150, 151, 407
markdown*	150, 150, 155, 407
\markdown_jeekyll_data_concatenate_address:NN	391
\markdown_jeekyll_data_pop:	391
\markdown_jeekyll_data_push:nN	391
\markdown_jeekyll_data_push_address_segment:n	390
\markdown_jeekyll_data_set_keyval:NN	392
\markdown_jeekyll_data_set_keyvals:nn	392
\markdown_jeekyll_data_update_address_tls:	391
\markdownBegin	53, 53–56, 148, 150, 152, 159
\markdownCleanup	397
\markdownConvert	397
\markdownEnd	53, 53–56, 148, 150–152, 159
\markdownError	148, 148
\markdownEscape	53, 56, 406
\markdownIfOption	58
\markdownIfSnippetExists	77
\markdownInfo	148, 148
\markdownInput	53, 55, 56, 150, 153, 155, 160, 403, 407
\markdownInputFilename	396
\markdownInputFileStream	398
\markdownInputPlainTeX	407
\markdownLoadPlainTeXTheme	156, 163, 380
\markdownLuaExecute	400, 403
\markdownLuaOptions	394, 397
\markdownMakeOther	148, 446
\markdownOptionFinalizeCache	59
\markdownOptionFrozenCache	59
\markdownOptionHybrid	62
\markdownOptionInputTempFileName	59
\markdownOptionNoDefaults	61
\markdownOptionOutputDir	60, 60, 63
\markdownOptionPlain	60

<code>\markdownOptionStripPercentSigns</code>	61
<code>\markdownOutputFileStream</code>	398
<code>\markdownPrepare</code>	397
<code>\markdownPrepareInputFilename</code>	396
<code>\markdownPrepareLuaOptions</code>	394
<code>\markdownReadAndConvert</code>	148, 398, 407, 408, 447
<code>\markdownReadAndConvertProcessLine</code>	399, 400
<code>\markdownReadAndConvertStripPercentSigns</code>	399
<code>\markdownReadAndConvertTab</code>	398
<code>\markdownRendererAttributeName</code>	82
<code>\markdownRendererAttributeIdentifier</code>	82
<code>\markdownRendererAttributeKeyValue</code>	82
<code>\markdownRendererBlockQuoteBegin</code>	83
<code>\markdownRendererBlockQuoteEnd</code>	84
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	84
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	84
<code>\markdownRendererCite</code>	87, 88
<code>\markdownRendererCodeSpan</code>	89
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	90
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	90
<code>\markdownRendererContentBlock</code>	90, 91
<code>\markdownRendererContentBlockCode</code>	91
<code>\markdownRendererContentBlockOnlineImage</code>	91
<code>\markdownRendererDisplayMath</code>	122
<code>\markdownRendererDlBegin</code>	92
<code>\markdownRendererDlBeginTight</code>	92
<code>\markdownRendererDlDefinitionBegin</code>	93
<code>\markdownRendererDlDefinitionEnd</code>	94
<code>\markdownRendererDlEnd</code>	94
<code>\markdownRendererDlEndTight</code>	95
<code>\markdownRendererDlItem</code>	93
<code>\markdownRendererDlItemEnd</code>	93
<code>\markdownRendererDocumentBegin</code>	107
<code>\markdownRendererDocumentEnd</code>	107
<code>\markdownRendererEllipsis</code>	41, 95
<code>\markdownRendererEmphasis</code>	95, 135
<code>\markdownRendererError</code>	124
<code>\markdownRendererFancyOlBegin</code>	109, 110
<code>\markdownRendererFancyOlBeginTight</code>	110
<code>\markdownRendererFancyOlEnd</code>	114
<code>\markdownRendererFancyOlEndTight</code>	114
<code>\markdownRendererFancyOlItem</code>	112

<code>\markdownRendererFancyOliItemEnd</code>	112
<code>\markdownRendererFancyOliItemWithNumber</code>	112
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	96
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	96
<code>\markdownRendererFencedDivAttributeContextBegin</code>	97
<code>\markdownRendererFencedDivAttributeContextEnd</code>	97
<code>\markdownRendererHalfTickedBox</code>	123
<code>\markdownRendererHardLineBreak</code>	105
<code>\markdownRendererHeaderAttributeContextBegin</code>	98
<code>\markdownRendererHeaderAttributeContextEnd</code>	98
<code>\markdownRendererHeadingFive</code>	100
<code>\markdownRendererHeadingFour</code>	100
<code>\markdownRendererHeadingOne</code>	98
<code>\markdownRendererHeadingSix</code>	100
<code>\markdownRendererHeadingThree</code>	99
<code>\markdownRendererHeadingTwo</code>	99
<code>\markdownRendererImage</code>	102
<code>\markdownRendererImageAttributeContextBegin</code>	102
<code>\markdownRendererImageAttributeContextEnd</code>	102
<code>\markdownRendererInlineHtmlComment</code>	101
<code>\markdownRendererInlineHtmlTag</code>	101
<code>\markdownRendererInlineMath</code>	122
<code>\markdownRendererInputBlockHtmlElement</code>	101
<code>\markdownRendererInputFencedCode</code>	89
<code>\markdownRendererInputRawBlock</code>	115
<code>\markdownRendererInputRawInline</code>	114
<code>\markdownRendererInputVerbatim</code>	88
<code>\markdownRendererInterblockSeparator</code>	103
<code>\markdownRendererJekyllDataBegin</code>	125
<code>\markdownRendererJekyllDataBoolean</code>	127
<code>\markdownRendererJekyllDataEmpty</code>	131
<code>\markdownRendererJekyllDataEnd</code>	125
<code>\markdownRendererJekyllDataMappingBegin</code>	126
<code>\markdownRendererJekyllDataMappingEnd</code>	126
<code>\markdownRendererJekyllDataNumber</code>	128
<code>\markdownRendererJekyllDataProgrammaticString</code>	128, 128, 129
<code>\markdownRendererJekyllDataSequenceBegin</code>	127
<code>\markdownRendererJekyllDataSequenceEnd</code>	127
<code>\markdownRendererJekyllDataString</code>	129, 133
<code>\markdownRendererJekyllDataStringPrototype</code>	143
<code>\markdownRendererJekyllDataTypographicString</code>	128, 128, 129, 364
<code>\markdownRendererLineBlockBegin</code>	104

<code>\markdownRendererLineBlockEnd</code>	104
<code>\markdownRendererLink</code>	105, 135
<code>\markdownRendererLinkAttributeContextBegin</code>	106
<code>\markdownRendererLinkAttributeContextEnd</code>	106
<code>\markdownRendererMark</code>	107
<code>\markdownRendererNbsp</code>	108
<code>\markdownRendererNote</code>	108
<code>\markdownRendererOlBegin</code>	109
<code>\markdownRendererOlBeginTight</code>	109
<code>\markdownRendererOlEnd</code>	113
<code>\markdownRendererOlEndTight</code>	113
<code>\markdownRendererOlItem</code>	42, 110
<code>\markdownRendererOlItemEnd</code>	111
<code>\markdownRendererOlItemWithNumber</code>	42, 111
<code>\markdownRendererParagraphSeparator</code>	104
<code>\markdownRendererReplacementCharacter</code>	116
<code>\markdownRendererSectionBegin</code>	115
<code>\markdownRendererSectionEnd</code>	115
<code>\markdownRendererSoftLineBreak</code>	105
<code>\markdownRendererStrikeThrough</code>	119
<code>\markdownRendererStrongEmphasis</code>	96
<code>\markdownRendererSubscript</code>	120
<code>\markdownRendererSuperscript</code>	120
<code>\markdownRendererTable</code>	121
<code>\markdownRendererTableAttributeContextBegin</code>	121
<code>\markdownRendererTableAttributeContextEnd</code>	121
<code>\markdownRendererTextCite</code>	88
<code>\markdownRendererThematicBreak</code>	123
<code>\markdownRendererTickedBox</code>	123
<code>\markdownRendererUlBegin</code>	85
<code>\markdownRendererUlBeginTight</code>	85
<code>\markdownRendererUlEnd</code>	87
<code>\markdownRendererUlEndTight</code>	87
<code>\markdownRendererUlItem</code>	86
<code>\markdownRendererUlItemEnd</code>	86
<code>\markdownRendererUntickedBox</code>	123
<code>\markdownRendererWarning</code>	124
<code>\markdownSetup</code>	57, 58, 62, 154, 155, 161, 408, 415
<code>\markdownSetupSnippet</code>	76, 76
<code>\markdownThemeVersion</code>	69, 69
<code>\markdownWarning</code>	148, 148
<code>\markinline</code>	53, 55, 150, 152, 401, 406

<code>\markinlinePlainTeX</code>	406
<code>new</code>	7, 18, 368, 369
<code>notes</code>	38, 108
<code>parsers</code>	235, 273
<code>parsers.punctuation</code>	236
<code>pipeTables</code>	7, 39, 45, 121
<code>preserveTabs</code>	39, 43, 274
<code>rawAttribute</code>	35, 39, 40, 114, 115
<code>reader</code>	8, 29, 163, 235, 272, 319
<code>reader->add_special_character</code>	8, 9, 29, 313
<code>reader->auto_link_email</code>	302
<code>reader->auto_link_url</code>	302
<code>reader->create_parser</code>	274
<code>reader->finalize_grammar</code>	309, 374
<code>reader->initialize_named_group</code>	313
<code>reader->insert_pattern</code>	8, 9, 29, 309, 315
<code>reader->lookup_note_reference</code>	288
<code>reader->lookup_reference</code>	287
<code>reader->normalize_tag</code>	273
<code>reader->options</code>	273
<code>reader->parser_functions</code>	274
<code>reader->parser_functions.name</code>	274
<code>reader->parsers</code>	273, 273
<code>reader->register_link</code>	287
<code>reader->update_rule</code>	309, 312, 315
<code>reader->writer</code>	273
<code>reader.new</code>	272, 273, 374
<code>relativeReferences</code>	40
<code>\setupmarkdown</code>	161, 161
<code>\setupyaml</code>	161
<code>shiftHeadings</code>	7, 41
<code>singletonCache</code>	18
<code>slice</code>	7, 41, 216, 228, 229
<code>smartEllipses</code>	41, 95, 156
<code>\startmarkdown</code>	159, 159, 447
<code>startNumber</code>	42, 110–112
<code>\startyaml</code>	159, 159, 447
<code>\stopmarkdown</code>	159, 159, 447
<code>\stopyaml</code>	159, 159, 447

strikeThrough	42, 119, 441
stripIndent	43, 274
stripPercentSigns	398, 399
subscripts	43, 120
superscripts	44, 120
syntax	310, 315
tableAttributes	44, 121, 437
tableCaptions	7, 44, 45, 121
taskLists	45, 123, 428
texComments	46, 275
texMathDollars	35, 46, 122
texMathDoubleBackslash	35, 47, 122
texMathSingleBackslash	35, 47, 122
tightLists	47, 85, 87, 92, 95, 109, 110, 113, 114, 415
underscores	48
unicodeNormalization	18, 19
unicodeNormalizationForm	18, 19
util.cache	164, 164
util.cache_verbatim	164
util.encode_json_string	165
util.err	164
util.escaper	167
util.expand_tabs_in_line	165
util.flatten	166
util.intersperse	167
util.map	167
util.pathname	168
util.rope_last	166
util.rope_to_string	166
util.salt	168
util.table_copy	165
util.walk	165, 166
util.warning	169
walkable_syntax	8, 20, 26, 309, 312–315
writer	163, 163, 215, 319
writer->active_attributes	227, 227–229
writer->attribute_type_levels	227
writer->attributes	225
writer->block_html_element	223
writer->blockquote	224

writer->bulletitem	222
writer->bulletlist	221
writer->citations	320
writer->code	220
writer->contentblock	325
writer->defer_call	234, 234
writer->definitionlist	328
writer->display_math	360
writer->div_begin	341
writer->div_end	341
writer->document	224
writer->ellipsis	218
writer->emphasis	223
writer->error	220
writer->escape	219
writer->escaped_chars	218, 219
writer->escaped_minimal_strings	218, 219
writer->escaped_strings	218
writer->escaped_uri_chars	218, 219
writer->fancyitem	331
writer->fancylist	330
writer->fencedCode	336
writer->flatten_inlines	216, 216
writer->get_state	234
writer->hard_line_break	218
writer->heading	232
writer->identifier	219
writer->image	221
writer->infostring	219
writer->inline_html_comment	223
writer->inline_html_tag	223
writer->inline_math	360
writer->interblocksep	217
writer->is_writing	216, 216
writer->jekyllData	364
writer->lineblock	348
writer->link	220
writer->mark	349
writer->math	219
writer->nbsp	217
writer->note	351
writer->options	216

writer->ordereditem	222
writer->orderedlist	222
writer->paragraph	217
writer->paragraphsep	217
writer->plain	217
writer->pop_attributes	227, 228, 229
writer->push_attributes	227, 228, 229
writer->rawBlock	337
writer->rawInline	358
writer->set_state	234
writer->slice_begin	216
writer->slice_end	216
writer->soft_line_break	217
writer->space	217
writer->span	319
writer->strike_through	358
writer->string	219
writer->strong	224
writer->subscript	359
writer->superscript	360
writer->table	354
writer->thematic_break	218
writer->checkbox	223
writer->undosep	217, 318
writer->uri	219
writer->verbatim	224
writer->warning	169, 220
writer.new	215, 215, 216, 374
\yaml	152
yaml	150, 151, 152, 407
\yamlBegin	53, 54, 148, 151, 159
\yamlEnd	53, 54, 148, 151, 152, 159
\yamlInput	53, 56, 150, 153, 161, 407
\yamlSetup	58