

A Markdown Interpreter for T_EX

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

Version 3.8.0-0-ga4bab835
2024-10-31

Contents

1	Introduction	1	3	Implementation	155
1.1	Requirements	2	3.1	Lua Implementation . . .	156
1.2	Feedback	7	3.2	Plain T _E X Implementation	368
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation . .	394
2	Interfaces	7	3.4	ConT _E Xt Implementation	432
2.1	Lua Interface	8			
2.2	Plain T _E X Interface	53		References	442
2.3	L ^A T _E X Interface	139			
2.4	ConT _E Xt Interface	150		Index	443

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	148
5	The banner of the Markdown package	149

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 LuaTeX engine (though not necessarily in the LuaMetaTeX engine).

LPeg ≥ 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 ≥ 0.10 is included in LuaTeX $\geq 0.72.0$ (TeX Live ≥ 2013).

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

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

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

MD5 A library that provides MD5 crypto functions. It is used by the Lunamark library to compute the digest of the input for caching purposes. MD5 is included in all releases of LuaTeX (TeX Live ≥ 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. We carry a copy of the library in file **markdown-tinyyaml.lua** distributed together with the Markdown package.

```
27 # hard lua-tinyyaml # TODO: Uncomment after TeX Live 2022 deprecation.
```

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
```

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` L^AT_EX theme (see Section 2.3.4).

```
40 soft latex
41 soft epstopdf-pkg # required by `latex`
```

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

```
42 soft fancyvrb
```

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

```
43 soft csvsimple
44 soft pgf # required by `csvsimple`, which loads `pgfkeys.sty`
45 soft tools # required by `csvsimple`, which loads `shellesc.sty`
```

gobble A package that provides the `\@gobblethree` T_EX command that is used in the default renderer prototype for citations. The package is included in T_EXLive \geq 2016.

```
46 soft gobble
```

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

```
47 soft amsmath
48 soft amsfons
```

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the [witiko/graphicx/http](#) L^AT_EX theme, see Section 2.3.4.

49 `soft catchfile`

grffile A package that extends the name processing of the graphics package to support a larger range of file names in $2006 \leq \text{T_EX Live} \leq 2019$. Since T_EX Live ≥ 2020 , the functionality of the package has been integrated in the L^AT_EX 2_ε kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#) L^AT_EX themes, see Section 2.3.4.

50 `soft grffile`

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.9, and also in the default renderer prototype for identifier attributes.

51 `soft etoolbox`

soulutf8 and xcolor Packages that are used in the default renderer prototypes for strike-throughs and marked text in pdfT_EX.

52 `soft soul`

53 `soft xcolor`

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

54 `soft lua-ul`

55 `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.

56 `soft ltxcmds`

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

57 `soft verse`

1.1.4 ConT_EXt Prerequisites

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

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

1.2 Feedback

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

1.3 Acknowledgements

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

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

Funding by the Faculty of Informatics at the Masaryk University in Brno [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 T_EX implementation of the package draws inspiration from several sources including the source code of L^AT_EX 2_ε, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from T_EX, the filecontents package by Scott Pakin and others.

2 Interfaces

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

Since neither T_EX nor Lua provide interfaces as a language construct, the separation to interfaces and implementations is a *gentlemen's agreement*. It serves as a means of 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 T_EX *token renderers* is exposed by the Lua layer. The plain T_EX layer exposes the conversion capabilities of Lua as T_EX macros. The L^AT_EX and ConT_EXt layers provide syntactic sugar on top of plain T_EX macros. The user can interface with any and all layers.

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

⁵See <https://tex.stackexchange.com>.

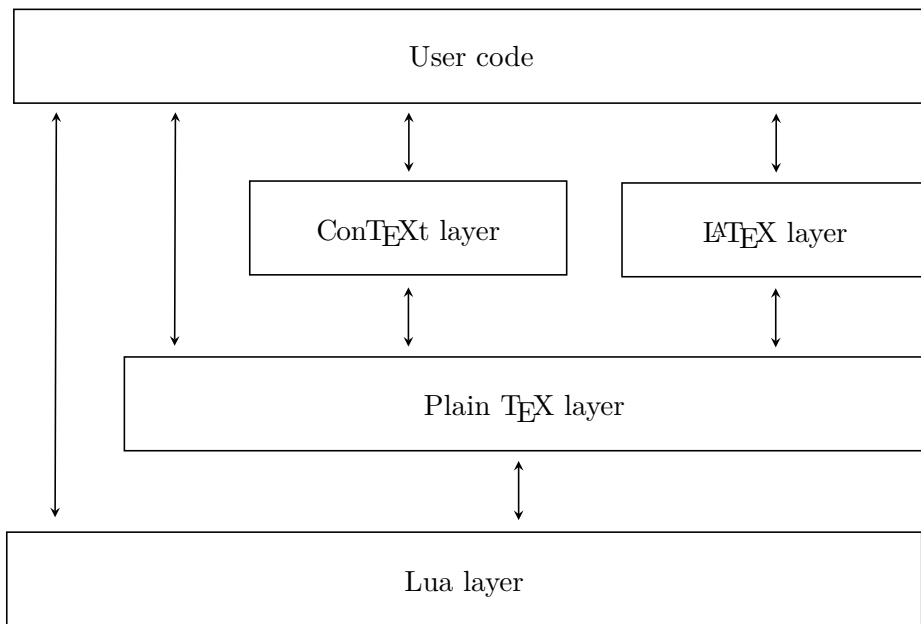


Figure 1: A block diagram of the Markdown package

2.1 Lua Interface

The Lua interface provides the conversion from UTF-8 encoded markdown to plain T_EX. This interface is used by the plain T_EX 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.

```
58 local M = {metadata = metadata}
```

2.1.1 Conversion from Markdown to Plain T_EX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain T_EX according to the table `options` that contains options recognized by the Lua interface (see Section 2.1.3). The `options` parameter is optional; when unspecified, the behaviour will be the same as if `options` were an empty table.

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

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


2.1.2 User-Defined Syntax Extensions

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

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

```
59 local walkable_syntax = {
60   Block = {
61     "Blockquote",
62     "Verbatim",
63     "ThematicBreak",
64     "BulletList",
65     "OrderedList",
66     "DisplayHtml",
67     "Heading",
68   },
69   BlockOrParagraph = {
70     "Block",
71     "Paragraph",
72     "Plain",
73   },
74   Inline = {
75     "Str",
76     "Space",
77     "Endline",
78     "EndlineBreak",
79     "LinkAndEmph",
80     "Code",
81     "AutoLinkUrl",
82     "AutoLinkEmail",
83     "AutoLinkRelativeReference",
84     "InlineHtml",
85     "HtmlEntity",
86     "EscapedChar",
87     "Smart",
88     "Symbol",
89   },
90 }
```

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.

```
91 local defaultOptions = {}
```

To enable the enumeration of Lua options, we will maintain the [\g_@@_lua_options_seq](#) sequence.

```
92 \ExplSyntaxOn
93 \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.

```
94 \prop_new:N \g_@@_lua_option_types_prop
95 \prop_new:N \g_@@_default_lua_options_prop
96 \seq_new:N \g_@@_option_layers_seq
97 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
98 \seq_gput_right:NV
99   \g_@@_option_layers_seq
100   \c_@@_option_layer_lua_tl
101 \cs_new:Nn
102   \@@_add_lua_option:nnn
103   {
104     \@@_add_option:Vnnn
105     \c_@@_option_layer_lua_tl
106     { #1 }
107     { #2 }
108     { #3 }
109   }
110 \cs_new:Nn
111   \@@_add_option:nnnn
112   {
113     \seq_gput_right:cn
```

```

114     { g_@@_ #1 _options_seq }
115     { #2 }
116   \prop_gput:cnn
117     { g_@@_ #1 _option_types_prop }
118     { #2 }
119     { #3 }
120   \prop_gput:cnn
121     { g_@@_default_ #1 _options_prop }
122     { #2 }
123     { #4 }
124   \@@_typecheck_option:n
125     { #2 }
126 }
127 \cs_generate_variant:Nn
128   \@@_add_option:nnnn
129   { Vnnn }
130 \tl_const:Nn \c_@@_option_value_true_tl { true }
131 \tl_const:Nn \c_@@_option_value_false_tl { false }
132 \cs_new:Nn \@@_typecheck_option:n
133 {
134   \@@_get_option_type:nN
135     { #1 }
136     \l_tmpa_tl
137   \str_case_e:Vn
138     \l_tmpa_tl
139     {
140       { \c_@@_option_type_boolean_tl }
141       {
142         \@@_get_option_value:nN
143           { #1 }
144           \l_tmpa_tl
145         \bool_if:nF
146           {
147             \str_if_eq_p:VV
148               \l_tmpa_tl
149               \c_@@_option_value_true_tl ||
150             \str_if_eq_p:VV
151               \l_tmpa_tl
152               \c_@@_option_value_false_tl
153           }
154         {
155           \msg_error:nnnV
156             { markdown }
157             { failed-typecheck-for-boolean-option }
158             { #1 }
159             \l_tmpa_tl
160         }
161       }
162     }

```

```

161         }
162     }
163 }
164 \msg_new:nnn
165 { markdown }
166 { failed-typecheck-for-boolean-option }
167 {
168     Option~#1~has~value~#2,~
169     but~a~boolean~(true~or~false)~was~expected.
170 }
171 \cs_generate_variant:Nn
172   \str_case_e:nn
173   { Vn }
174 \cs_generate_variant:Nn
175   \msg_error:nnnn
176   { nnnV }
177 \seq_new:N
178   \g_@@_option_types_seq
179 \tl_const:Nn
180   \c_@@_option_type_clist_tl
181   { clist }
182 \seq_gput_right:NV
183   \g_@@_option_types_seq
184   \c_@@_option_type_clist_tl
185 \tl_const:Nn
186   \c_@@_option_type_counter_tl
187   { counter }
188 \seq_gput_right:NV
189   \g_@@_option_types_seq
190   \c_@@_option_type_counter_tl
191 \tl_const:Nn
192   \c_@@_option_type_boolean_tl
193   { boolean }
194 \seq_gput_right:NV
195   \g_@@_option_types_seq
196   \c_@@_option_type_boolean_tl
197 \tl_const:Nn
198   \c_@@_option_type_number_tl
199   { number }
200 \seq_gput_right:NV
201   \g_@@_option_types_seq
202   \c_@@_option_type_number_tl
203 \tl_const:Nn
204   \c_@@_option_type_path_tl
205   { path }
206 \seq_gput_right:NV
207   \g_@@_option_types_seq

```

```

208 \c_@@_option_type_path_tl
209 \tl_const:Nn
210 \c_@@_option_type_slice_tl
211 { slice }
212 \seq_gput_right:NV
213 \g_@@_option_types_seq
214 \c_@@_option_type_slice_tl
215 \tl_const:Nn
216 \c_@@_option_type_string_tl
217 { string }
218 \seq_gput_right:NV
219 \g_@@_option_types_seq
220 \c_@@_option_type_string_tl
221 \cs_new:Nn
222 \@@_get_option_type:nN
223 {
224   \bool_set_false:N
225     \l_tmpa_bool
226   \seq_map_inline:Nn
227     \g_@@_option_layers_seq
228     {
229       \prop_get:cnNT
230       { g_@@_ ##1 _option_types_prop }
231       { #1 }
232       \l_tmpa_tl
233       {
234         \bool_set_true:N
235           \l_tmpa_bool
236         \seq_map_break:
237       }
238     }
239   \bool_if:nF
240     \l_tmpa_bool
241   {
242     \msg_error:nnn
243       { markdown }
244       { undefined-option }
245       { #1 }
246   }
247   \seq_if_in:NVF
248     \g_@@_option_types_seq
249     \l_tmpa_tl
250   {
251     \msg_error:nnnV
252       { markdown }
253       { unknown-option-type }
254       { #1 }

```

```

255         \l_tmpa_tl
256     }
257     \tl_set_eq:NN
258     #2
259     \l_tmpa_tl
260 }
261 \msg_new:nnn
262 { markdown }
263 { unknown-option-type }
264 {
265     Option~#1~has~unknown~type~#2.
266 }
267 \msg_new:nnn
268 { markdown }
269 { undefined-option }
270 {
271     Option~#1~is~undefined.
272 }
273 \cs_new:Nn
274 \@@_get_default_option_value:nN
275 {
276     \bool_set_false:N
277     \l_tmpa_bool
278     \seq_map_inline:Nn
279     \g_@@_option_layers_seq
280     {
281         \prop_get:cnNT
282         { g_@@_default_ ##1 _options_prop }
283         { #1 }
284         #2
285         {
286             \bool_set_true:N
287             \l_tmpa_bool
288             \seq_map_break:
289         }
290     }
291     \bool_if:nF
292     \l_tmpa_bool
293     {
294         \msg_error:nnn
295         { markdown }
296         { undefined-option }
297         { #1 }
298     }
299 }
300 \cs_new:Nn
301 \@@_get_option_value:nN

```

```

302 {
303   \@@_option_tl_to_csname:nN
304   { #1 }
305   \l_tmpa_tl
306   \cs_if_free:cTF
307   { \l_tmpa_tl }
308   {
309     \@@_get_default_option_value:nN
310     { #1 }
311     #2
312   }
313   {
314     \@@_get_option_type:nN
315     { #1 }
316     \l_tmpa_tl
317     \str_if_eq:NNTF
318     \c_@@_option_type_counter_tl
319     \l_tmpa_tl
320     {
321       \@@_option_tl_to_csname:nN
322       { #1 }
323       \l_tmpa_tl
324       \tl_set:Nx
325       #2
326       { \the \cs:w \l_tmpa_tl \cs_end: }
327     }
328     {
329       \@@_option_tl_to_csname:nN
330       { #1 }
331       \l_tmpa_tl
332       \tl_set:Nv
333       #2
334       { \l_tmpa_tl }
335     }
336   }
337 }
338 \cs_new:Nn \@@_option_tl_to_csname:nN
339 {
340   \tl_set:Nn
341   \l_tmpa_tl
342   { \str_uppercase:n { #1 } }
343   \tl_set:Nx
344   #2
345   {
346     markdownOption
347     \tl_head:f { \l_tmpa_tl }
348     \tl_tail:n { #1 }

```

```

349     }
350 }

```

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.

```

351 \cs_new:Nn \@@_with_various_cases:nn
352 {
353   \seq_clear:N
354   \l_tmpa_seq
355   \seq_map_inline:Nn
356     \g_@@_cases_seq
357     {
358       \tl_set:Nn
359         \l_tmpa_tl
360         { #1 }
361       \use:c { ##1 }
362       \l_tmpa_tl
363       \seq_put_right:NV
364         \l_tmpa_seq
365         \l_tmpa_tl
366     }
367   \seq_map_inline:Nn
368     \l_tmpa_seq
369     { #2 }
370 }

```

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

```

371 \cs_new:Nn \@@_with_various_cases_break:
372 {
373   \seq_map_break:
374 }

```

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

```

375 \seq_new:N \g_@@_cases_seq
376 \cs_new:Nn \@@_camel_case:N
377 {
378   \regex_replace_all:nnN
379     { _ ([a-z]) }
380     { \c { str_uppercase:n } \cB\{ \1 \cE\} }
381     #1
382   \tl_set:Nx
383     #1
384     { #1 }
385 }
386 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }

```



```

387 \cs_new:Nn \@@_snake_case:N
388 {
389   \regex_replace_all:nnN
390   { ([a-z])([A-Z]) }
391   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
392   #1
393   \tl_set:Nx
394   #1
395   { #1 }
396 }
397 \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.

```

398 \@@_add_lua_option:nnn
399 { eagerCache }
400 { boolean }
401 { true }
402 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.

```
403 \@@_add_lua_option:nnn
404   { experimental }
405   { boolean }
406   { false }
407 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.

```
408 \@@_add_lua_option:nnn
409   { singletonCache }
410   { boolean }
411   { true }
412 defaultOptions.singletonCache = true
413 local singletonCache = {
414   convert = nil,
415   options = nil,
416 }
```

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

`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.

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

421 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.

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

426 defaultOptions.unicodeNormalizationForm = "nfc"
```

2.1.5 File and Directory Names

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

`cacheDir`= $\langle path \rangle$

default: .

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

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

```
427 \@@_add_lua_option:nnn
428   { cacheDir }
429   { path }
430   { \markdownOptionOutputDir / _markdown_\jobname }
431 defaultOptions.cacheDir = "."
```

`contentBlocksLanguageMap`= $\langle filename \rangle$

default: markdown-languages.json

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

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

`debugExtensionsFileName`= $\langle filename \rangle$

default: debug-extensions.json

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

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

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

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

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

```
442 \@@_add_lua_option:nnn
443   { frozenCacheFileName }
444   { path }
445   { \markdownOptionCacheDir / frozenCache.tex }
446 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`.

```
447 \@@_add_lua_option:nnn
448   { autoIdentifiers }
449   { boolean }
450   { false }
451 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.

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

```

452 \@@_add_lua_option:nnn
453   { blankBeforeBlockquote }
454   { boolean }
455   { false }

456 defaultOptions.blankBeforeBlockquote = false

```

`blankBeforeCodeFence=true, false` default: false

true Require a blank line between a paragraph and the following fenced code block.

false Do not require a blank line between a paragraph and the following fenced code block.

```

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

461 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.

```

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

466 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.

```

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

471 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.

```
472 \@@_add_lua_option:nnn
473   { blankBeforeList }
474   { boolean }
475   { false }

476 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.

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

481 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.

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

486 defaultOptions.breakableBlockquotes = true
```

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

`citationNbsps=true, false`

default: false

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

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

491 defaultOptions.citationNbsps = true
```

`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.

```
492 \@@_add_lua_option:nnn
493 { citations }
494 { boolean }
495 { false }

496 defaultOptions.citations = false
```

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

`codeSpans=true, false`

default: true

true Enable the code span syntax:

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

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

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

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

`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.

```
502 \@@_add_lua_option:nnn  
503   { contentBlocks }  
504   { boolean }  
505   { false }  
  
506 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
```

```
507 \@@_add_lua_option:nnn
508   { contentLevel }
509   { string }
510   { block }
511 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.

```
512 \@@_add_lua_option:nnn
513   { debugExtensions }
514   { boolean }
515   { false }
516 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.

```

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

521 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.

```

522 \@@_add_lua_option:nnn
523   { ensureJekyllData }
524   { boolean }
525   { false }

526 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[jeekyllData, expectJekyllData]{markdown}
\begin{document}
\begin{markdown}
- this
- is
- YAML
...
- followed
- by
- Markdown
\end{markdown}
\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}

```

```

527 \@@_add_lua_option:nnn
528   { expectJekyllData }

```

```

529 { boolean }
530 { false }

531 defaultOptions.expectJekyllData = false

```

`extensions`= $\langle filenames \rangle$

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:

```

532 metadata.user_extension_api_version = 2
533 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.

```
534 \cs_generate_variant:Nn
535   \@@_add_lua_option:nnn
536   { nnV }
537 \@@_add_lua_option:nnV
538   { extensions }
539   { clist }
540   \c_empty_clist
541 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.

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

`fencedCode=true, false`

default: `true`

`true` Enable the commonmark fenced code block extension:

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

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

```

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

```

false Disable the commonmark fenced code block extension.

```

547 \@@_add_lua_option:nnn
548   { fencedCode }
549   { boolean }
550   { true }

551 defaultOptions.fencedCode = true

```

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.

```

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

556 defaultOptions.fencedCodeAttributes = false

```

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

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.

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

561 defaultOptions.fencedDivs = false
```

finalizeCache=true, false

default: false

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

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

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

566 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 T_EX macro **\markdownFrozenCache***<number>* that will typeset markdown document number *<number>*.

¹³See https://pandoc.org/MANUAL.html#extension-fenced_divs.


```

567 \@@_add_lua_option:nnn
568   { frozenCacheCounter }
569   { counter }
570   { 0 }

571 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.

See also the option `autoIdentifiers`.

```

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

576 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.

```

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

581 defaultOptions.hashEnumerators = false

```

¹⁴See https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers.

`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.

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

586 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.

```
587 \@@_add_lua_option:nnn
588   { html }
589   { boolean }
590   { true }

591 defaultOptions.html = true
```

`hybrid=true, false`

default: `false`

`true` Disable the escaping of special plain T_EX characters, which makes it possible to intersperse your markdown markup with T_EX code. The intended usage is in documents prepared manually by a human author. In such documents, it can often be desirable to mix T_EX 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}
\]

```

```

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

```

`inlineCodeAttributes=true, false`

default: false

true Enable the Pandoc inline code span attribute extension¹⁵:

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

false Enable the Pandoc inline code span attribute extension.

```

597 \@@_add_lua_option:nnn
598   { inlineCodeAttributes }
599   { boolean }
600   { false }
601 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.

```

602 \@@_add_lua_option:nnn
603   { inlineNotes }
604   { boolean }
605   { false }
606 defaultOptions.inlineNotes = false

```

¹⁵See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.

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

`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.
---
```

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

```
607 \@@_add_lua_option:nnn
608 { jeekyllData }
609 { boolean }
610 { false }
611 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.

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

¹⁷See <https://pandoc.org/MANUAL.html#extension-yaml-metadata-block>.

¹⁸See <https://pandoc.org/MANUAL.html#extension-link-attributes>.

`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.

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

`mark=true, false`

default: false

`true` Enable the Pandoc mark syntax extension²⁰:

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

`false` Disable the Pandoc mark syntax extension.

```
622 \@@_add_lua_option:nnn
623   { mark }
624   { boolean }
625   { false }
626 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.
```

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

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

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

```
[^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.

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

pipeTables=true, false default: false

true Enable the PHP Markdown pipe table syntax extension:

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

false Disable the PHP Markdown pipe table syntax extension.

```
632 \@@_add_lua_option:nnn
633   { pipeTables }
634   { boolean }
635   { false }
636 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.

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

641 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}
\[distance[i] =
 \begin{dcases}
 a & b \\
 c & d
 \end{dcases}
\]
```

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.

```
642 \@@_add_lua_option:nnn
643 { rawAttribute }
644 { boolean }
645 { false }

646 defaultOptions.rawAttribute = false
```

---

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



`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.

```
647 \@@_add_lua_option:nnn
```

```
648 { relativeReferences }
```

```
649 { boolean }
```

```
650 { false }
```

```
651 defaultOptions.relativeReferences = false
```

`shiftHeadings=<shift amount>`

default: 0

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

```
652 \@@_add_lua_option:nnn
```

```
653 { shiftHeadings }
```

```
654 { number }
```

```
655 { 0 }
```

```
656 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.

---

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

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

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

```

657 \@@_add_lua_option:nnn
658 { slice }
659 { slice }
660 { ^~$ }
661 defaultOptions.slice = "^ $"

```

`smartEllipses=true, false`

default: false

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

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

```

662 \@@_add_lua_option:nnn
663 { smartEllipses }
664 { boolean }
665 { false }
666 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 `\markdownRenderOlItemWithNumber`  $\TeX$  macro.

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

```

667 \@@_add_lua_option:nnn
668 { startNumber }
669 { boolean }
670 { true }
671 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.

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

676 defaultOptions.strikeThrough = false
```

`stripIndent=true, false`

default: false

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

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

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

```
677 \@@_add_lua_option:nnn
678 { stripIndent }
679 { boolean }
680 { false }

681 defaultOptions.stripIndent = false
```

---

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

`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.

```
682 \@@_add_lua_option:nnn
683 { subscripts }
684 { boolean }
685 { false }

686 defaultOptions.subscripts = false
```

`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.

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

691 defaultOptions.superscripts = false
```

`tableAttributes=true, false` default: false

**true**

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

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

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

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

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

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

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

696 defaultOptions.tableAttributes = false
```

**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.

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

701 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.

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

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

```

702 \@@_add_lua_option:nnn
703   { taskLists }
704   { boolean }
705   { false }
706 defaultOptions.taskLists = false

```

`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.

```

707 \@@_add_lua_option:nnn
708   { texComments }
709   { boolean }
710   { false }
711 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.

```

712 \@@_add_lua_option:nnn
713   { texMathDollars }
714   { boolean }
715   { false }
716 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:

<pre>inline math: \\\(E=mc^2\\) display math: \\[E=mc^2\\]</pre>

false Disable the Pandoc double backslash math syntax extension.

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

721 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false` default: false

true Enable the Pandoc single backslash math syntax extension³¹:

<pre>inline math: \\\(E=mc^2\\) display math: \\[E=mc^2\\]</pre>

false Disable the Pandoc single backslash math syntax extension.

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

726 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.

```

727 \@@_add_lua_option:nnn
728   { tightLists }
729   { boolean }
730   { true }
731 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.

```

732 \@@_add_lua_option:nnn
733   { underscores }
734   { boolean }
735   { true }
736 \ExplSyntaxOff
737 defaultOptions.underscores = true

```


2.1.7 Command-Line Interface

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

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

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

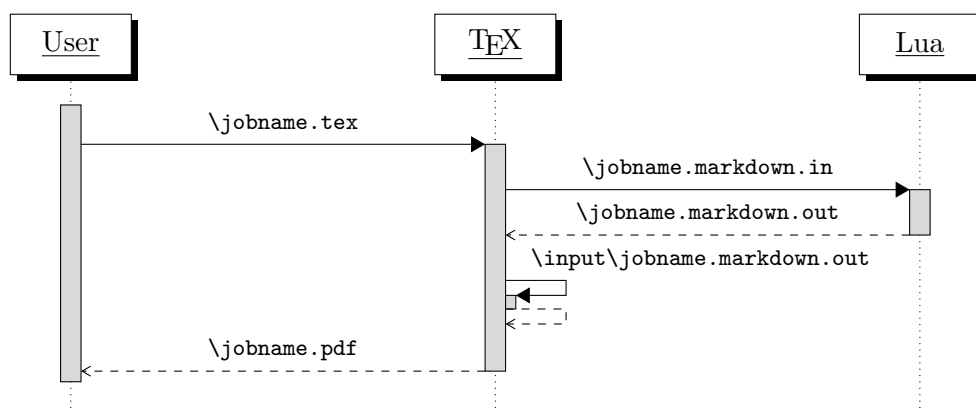


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T_EX interface

```

738
739 local HELP_STRING = [[
740 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
741 where OPTIONS are documented in the Lua interface section of the
742 technical Markdown package documentation.
743
744 When OUTPUT_FILE is unspecified, the result of the conversion will be
745 written to the standard output. When INPUT_FILE is also unspecified, the
746 result of the conversion will be read from the standard input.
747
748 Report bugs to: witiko@mail.muni.cz
749 Markdown package home page: <https://github.com/witiko/markdown>]]
750

```

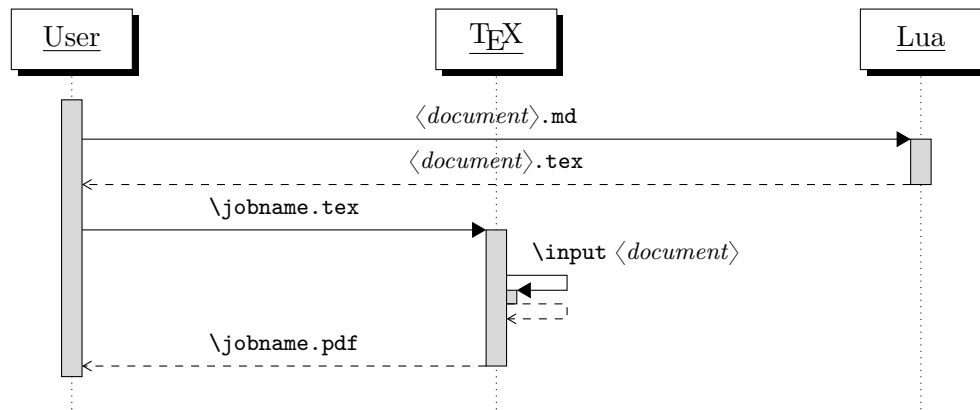


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

751 local VERSION_STRING = [[
752 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
753
754 Copyright (C) ]] .. table.concat(metadata.copyright,
755                                   "\nCopyright (C) ") .. [[
756
757 License: ]] .. metadata.license
758
759 local function warn(s)
760   io.stderr:write("Warning: " .. s .. "\n")
761 end
762
763 local function error(s)
764   io.stderr:write("Error: " .. s .. "\n")
765   os.exit(1)
766 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.

```

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

```

776     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
777 end)
778 return cased_option_name
779 end
780
781 local cases = {camel_case, snake_case}
782 local various_case_options = {}
783 for option_name, _ in pairs(defaultOptions) do
784     for _, case in ipairs(cases) do
785         various_case_options[case(option_name)] = option_name
786     end
787 end
788
789 local process_options = true
790 local options = {}
791 local input_filename
792 local output_filename
793 for i = 1, #arg do
794     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.

```

795     if arg[i] == "--" then
796         process_options = false
797         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.

```

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

```

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

```

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

```

```

812         table.insert(options[key], item)
813     end
814 else
815     if default_type ~= "string" then
816         if default_type == "nil" then
817             warn('Option "' .. key .. '" not recognized.')
818         else
819             warn('Option "' .. key .. '" type not recognized, ' ..
820                 'please file a report to the package maintainer.')
821         end
822         warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
823             key .. '" as a string.')
824     end
825     options[key] = value
826 end
827 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.

```

828     elseif arg[i] == "--help" or arg[i] == "-h" then
829         print(HELP_STRING)
830         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.

```

831     elseif arg[i] == "--version" or arg[i] == "-v" then
832         print(VERSION_STRING)
833         os.exit()
834     end
835 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.

```

836     if input_filename == nil then
837         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.

```

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

```

844 `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.

```
845 \def\markdownLastModified{((LASTMODIFIED))}%  
846 \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 `\input`ting 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.

```
847 \let\markdownBegin\relax  
848 \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.

```
849 \let\yamlBegin\relax
850 \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 `\yamlEnd` macros:

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

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

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

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

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

```
851 \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.

```
852 \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.

```
853 \def\yamlInput#1{%
854   \begingroup
855   \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
856   \markdownInput{#1}%
857   \endgroup
858 }%
```

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.

```
859 \let\markdownEscape\relax
```


2.2.2 Options

The plain T_EX options are represented by T_EX 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 T_EX interface.

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

```
860 \ExplSyntaxOn
861 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
862 \cs_generate_variant:Nn
863   \tl_const:Nn
864   { NV }
865 \tl_if_exist:NF
866   \c_@@_top_layer_tl
867   {
868     \tl_const:NV
869       \c_@@_top_layer_tl
870       \c_@@_option_layer_plain_tex_tl
871   }
```

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

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

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

```
873 \prop_new:N \g_@@_plain_tex_option_types_prop
874 \prop_new:N \g_@@_default_plain_tex_options_prop
875 \seq_gput_right:NV
876   \g_@@_option_layers_seq
877   \c_@@_option_layer_plain_tex_tl
878 \cs_new:Nn
879   \@@_add_plain_tex_option:nnn
880   {
881     \@@_add_option:Vnnn
882       \c_@@_option_layer_plain_tex_tl
883       { #1 }
884       { #2 }
885       { #3 }
886   }
```

The plain T_EX options may be also be specified via the `\markdownSetup` macro. Here, the plain T_EX 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.

```

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

```

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

```

897 \cs_gset_eq:NN
898   \yamlSetup
899   \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.

```

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

```

```

925   }
926   \msg_new:nnn
927   { markdown }
928   { expected-boolean-option }
929   {
930     Option~#1~has~type~#2,~
931     but~a~boolean~was~expected.
932   }
933   \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`.

```

934   \@@_add_plain_tex_option:nnn
935   { frozenCache }
936   { boolean }
937   { 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.

```

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

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

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 L^AT_EX and ConT_EXt. Furthermore, the default definitions may change at any time, which may pose a problem for maintainers of Markdown themes and templates who may require a stable output.

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

Here is how you would enable the macro in a L^AT_EX document:

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

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

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

```
950 \@@_add_plain_tex_option:nnn
951   { noDefaults }
952   { boolean }
953   { 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`.

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

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

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

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.

```

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

```

Do not override options defined before loading the package.

```

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

```

1012 \ExplSyntaxOn
1013 \cs_new:Nn
1014   \@@_define_option_command_output_dir:
1015   {
1016     \cs_if_free:NT
1017     \markdownOptionOutputDir
1018     {
1019       \bool_if:nTF
1020       {
1021         \cs_if_exist_p:N
1022         \luabridge_tl_set:Nn &&
1023         (
1024           \int_compare_p:nNn
1025           { \g_luabridge_method_int }
1026           =
1027           { \c_luabridge_method_directlua_int } ||

```

```

1028         \sys_if_shell_unrestricted_p:
1029     )
1030 }
1031 {

```

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.

```

1032     \group_begin:
1033     \cctab_select:N
1034     \c_str_cctab
1035     \luabridge_tl_set:Nn
1036     \l_tmpa_tl
1037     { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
1038     \tl_gset:NV
1039     \markdownOptionOutputDir
1040     \l_tmpa_tl
1041     \group_end:
1042 }
1043 {
1044     \tl_gset:Nn
1045     \markdownOptionOutputDir
1046     { . }
1047 }
1048 }
1049 }
1050 \cs_new:Nn
1051 \@@_set_option_value:nn
1052 {
1053     \@@_define_option:n
1054     { #1 }
1055     \@@_get_option_type:nN
1056     { #1 }
1057     \l_tmpa_tl
1058     \str_if_eq:NNTF
1059     \c_@@_option_type_counter_tl
1060     \l_tmpa_tl
1061     {
1062         \@@_option_tl_to_csname:nN
1063         { #1 }
1064         \l_tmpa_tl
1065         \int_gset:cn
1066         { \l_tmpa_tl }
1067         { #2 }
1068     }
1069     {
1070         \@@_option_tl_to_csname:nN

```



```

1071         { #1 }
1072         \l_tmpa_tl
1073         \cs_set:cpn
1074         { \l_tmpa_tl }
1075         { #2 }
1076     }
1077 }
1078 \cs_generate_variant:Nn
1079   \@@_set_option_value:nn
1080   { nV }
1081 \cs_new:Nn
1082   \@@_define_option:n
1083   {
1084     \@@_option_tl_to_csname:nN
1085     { #1 }
1086     \l_tmpa_tl
1087     \cs_if_free:cT
1088     { \l_tmpa_tl }
1089     {
1090       \@@_get_option_type:nN
1091       { #1 }
1092       \l_tmpb_tl
1093       \str_if_eq:NNT
1094       \c_@@_option_type_counter_tl
1095       \l_tmpb_tl
1096       {
1097         \@@_option_tl_to_csname:nN
1098         { #1 }
1099         \l_tmpa_tl
1100         \int_new:c
1101         { \l_tmpa_tl }
1102       }
1103     }
1104   }
1105 \cs_new:Nn
1106   \@@_define_option_keyval:nnn
1107   {
1108     \prop_get:cnN
1109     { g_@@_ #1 _option_types_prop }
1110     { #2 }
1111     \l_tmpa_tl
1112     \str_if_eq:VVTF
1113     \l_tmpa_tl
1114     \c_@@_option_type_boolean_tl
1115     {
1116       \keys_define:nn
1117       { markdown/options }

```

```
1118         {
```

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

```
1119             #3 .code:n = {
1120                 \tl_set:Nx
1121                 \l_tmpa_tl
1122                 {
1123                     \str_case:nnF
1124                     { ##1 }
1125                     {
1126                         { yes } { true }
1127                         { no } { false }
1128                     }
1129                     { ##1 }
1130                 }
1131                 \@@_set_option_value:nV
1132                 { #2 }
1133                 \l_tmpa_tl
1134             },
1135             #3 .default:n = { true },
1136         }
1137     }
1138     {
1139         \keys_define:nn
1140         { markdown/options }
1141         {
1142             #3 .code:n = {
1143                 \@@_set_option_value:nn
1144                 { #2 }
1145                 { ##1 }
1146             },
1147         }
1148     }
```

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.

```
1149     \str_if_eq:VVT
1150     \l_tmpa_tl
1151     \c_@@_option_type_clist_tl
1152     {
1153         \tl_set:Nn
1154         \l_tmpa_tl
1155         { #3 }
```

```

1156     \tl_reverse:N
1157     \l_tmpa_tl
1158     \str_if_eq:enF
1159     {
1160         \tl_head:V
1161         \l_tmpa_tl
1162     }
1163     { s }
1164     {
1165         \msg_error:nnn
1166         { markdown }
1167         { malformed-name-for-clist-option }
1168         { #3 }
1169     }
1170     \tl_set:Nx
1171     \l_tmpa_tl
1172     {
1173         \tl_tail:V
1174         \l_tmpa_tl
1175     }
1176     \tl_reverse:N
1177     \l_tmpa_tl
1178     \tl_put_right:Nn
1179     \l_tmpa_tl
1180     {
1181         .code:n = {
1182             \@_get_option_value:nN
1183             { #2 }
1184             \l_tmpa_tl
1185             \clist_set:NV
1186             \l_tmpa_clist
1187             { \l_tmpa_tl, { ##1 } }
1188             \@_set_option_value:nV
1189             { #2 }
1190             \l_tmpa_clist
1191         }
1192     }
1193     \keys_define:nV
1194     { markdown/options }
1195     \l_tmpa_tl
1196 }
1197 }
1198 \cs_generate_variant:Nn
1199 \clist_set:Nn
1200 { NV }
1201 \cs_generate_variant:Nn
1202 \keys_define:nn

```

```

1203 { nV }
1204 \cs_generate_variant:Nn
1205 \@@_set_option_value:nn
1206 { nV }
1207 \prg_generate_conditional_variant:Nnn
1208 \str_if_eq:nn
1209 { en }
1210 { p, F }
1211 \msg_new:nnn
1212 { markdown }
1213 { malformed-name-for-clist-option }
1214 {
1215     Clist-option-name~#1~does~not~end~with~-s.
1216 }

```

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.

```

1217 \str_if_eq:VVT
1218 \c_@@_top_layer_tl
1219 \c_@@_option_layer_plain_tex_tl
1220 {
1221     \@@_define_option_commands_and_keyvals:
1222 }
1223 \ExplSyntaxOff

```

2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>`, optionally followed by `@<theme version>`, load a TeX document (further referred to as *a theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (_). The theme name must be *qualified* and contain no underscores or at signs (@). Themes are inspired by the Beamer L^AT_EX 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].

```

1224 \ExplSyntaxOn
1225 \keys_define:nn
1226 { markdown/options }
1227 {
1228   theme .code:n = {
1229     \@@_set_theme:n
1230     { #1 }
1231   },
1232   import .code:n = {
1233     \tl_set:Nn
1234     \l_tmpa_tl
1235     { #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.

```

1236   \tl_replace_all:NnV
1237   \l_tmpa_tl
1238   { / }
1239   \c_backslash_str
1240   \keys_set:nV
1241   { markdown/options/import }
1242   \l_tmpa_tl
1243 },
1244 }

```

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.

```

1245 \seq_new:N
1246 \g_@@_theme_names_seq
1247 \seq_new:N
1248 \g_@@_theme_versions_seq
1249 \tl_new:N

```

```

1250 \g_@@_current_theme_tl
1251 \tl_gset:Nn
1252 \g_@@_current_theme_tl
1253 { }
1254 \seq_gput_right:NV
1255 \g_@@_theme_names_seq
1256 \g_@@_current_theme_tl
1257 \cs_new:Npn
1258 \markdownThemeVersion
1259 { }
1260 \seq_gput_right:NV
1261 \g_@@_theme_versions_seq
1262 \g_@@_current_theme_tl
1263 \cs_new:Nn
1264 \@@_set_theme:n
1265 {

```

First, we validate the theme name.

```

1266 \str_if_in:nnF
1267 { #1 }
1268 { / }
1269 {
1270 \msg_error:nnn
1271 { markdown }
1272 { unqualified-theme-name }
1273 { #1 }
1274 }
1275 \str_if_in:nnT
1276 { #1 }
1277 { _ }
1278 {
1279 \msg_error:nnn
1280 { markdown }
1281 { underscores-in-theme-name }
1282 { #1 }
1283 }

```

Next, we extract the theme version.

```

1284 \str_if_in:nnTF
1285 { #1 }
1286 { @ }
1287 {
1288 \regex_extract_once:nnN
1289 { (.*?) @ (.*?) }
1290 { #1 }
1291 \l_tmpa_seq
1292 \seq_gpop_left:NN
1293 \l_tmpa_seq

```

```

1294         \l_tmpa_tl
1295     \seq_gpop_left:NN
1296         \l_tmpa_seq
1297         \l_tmpa_tl
1298     \tl_gset:NV
1299         \g_@@_current_theme_tl
1300         \l_tmpa_tl
1301     \seq_gpop_left:NN
1302         \l_tmpa_seq
1303         \l_tmpa_tl
1304     \cs_gset:Npe
1305         \markdownThemeVersion
1306         {
1307             \tl_use:N
1308                 \l_tmpa_tl
1309         }
1310     }
1311     {
1312         \tl_gset:Nn
1313             \g_@@_current_theme_tl
1314             { #1 }
1315         \cs_gset:Npn
1316             \markdownThemeVersion
1317             { latest }
1318     }

```

Next, we munge the theme name.

```

1319     \str_set:NV
1320         \l_tmpa_str
1321         \g_@@_current_theme_tl
1322     \str_replace_all:Nnn
1323         \l_tmpa_str
1324         { / }
1325         { _ }

```

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

```

1326     \tl_set:NV
1327         \l_tmpa_tl
1328         \g_@@_current_theme_tl
1329     \tl_put_right:Nn
1330         \g_@@_current_theme_tl
1331         { / }
1332     \seq_gput_right:NV
1333         \g_@@_theme_names_seq
1334         \g_@@_current_theme_tl
1335     \seq_gput_right:NV
1336         \g_@@_theme_versions_seq

```

```

1337     \markdownThemeVersion
1338     \@@_load_theme:VeV
1339     \l_tmpa_tl
1340     { \markdownThemeVersion }
1341     \l_tmpa_str

```

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

```

1342     \seq_gpop_right:NN
1343     \g_@@_theme_names_seq
1344     \l_tmpa_tl
1345     \seq_get_right:NN
1346     \g_@@_theme_names_seq
1347     \l_tmpa_tl
1348     \tl_gset:Nv
1349     \g_@@_current_theme_tl
1350     \l_tmpa_tl
1351     \seq_gpop_right:NN
1352     \g_@@_theme_versions_seq
1353     \l_tmpa_tl
1354     \seq_get_right:NN
1355     \g_@@_theme_versions_seq
1356     \l_tmpa_tl
1357     \cs_gset:Npe
1358     \markdownThemeVersion
1359     {
1360         \tl_use:N
1361         \l_tmpa_tl
1362     }
1363 }
1364 \msg_new:nnnn
1365 { markdown }
1366 { unqualified-theme-name }
1367 { Won't~load~theme~with~unqualified~name~#1 }
1368 { Theme~names~must~contain~at~least~one~forward~slash }
1369 \msg_new:nnnn
1370 { markdown }
1371 { underscores-in-theme-name }
1372 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1373 { Theme~names~must~not~contain~underscores~in~their~names }
1374 \cs_generate_variant:Nn
1375 \tl_replace_all:Nnn
1376 { NnV }
1377 \cs_generate_variant:Nn
1378 \cs_gset:Npn
1379 { Npe }
1380 \ExplSyntaxOff

```


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

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

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

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

witiko/markdown/defaults A plain T_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 \ExplSyntaxOn
1382 \prop_new:N
1383   \g_@@_snippets_prop
1384 \cs_new:Nn
1385   \@@_setup_snippet:nn
1386   {
1387     \tl_if_empty:nT
1388       { #1 }
1389     {
1390       \msg_error:nnn
1391         { markdown }
1392         { empty-snippet-name }
1393         { #1 }
1394     }
1395     \tl_set:NV
1396       \l_tmpa_tl
1397       \g_@@_current_theme_tl
1398     \tl_put_right:Nn
```

```

1399     \l_tmpa_tl
1400     { #1 }
1401     \@@_if_snippet_exists:nT
1402     { #1 }
1403     {
1404         \msg_warning:nnV
1405         { markdown }
1406         { redefined-snippet }
1407         \l_tmpa_tl
1408     }
1409     \keys_precompile:nnN
1410     { markdown/options }
1411     { #2 }
1412     \l_tmpb_tl
1413     \prop_gput:NVV
1414     \g_@@_snippets_prop
1415     \l_tmpa_tl
1416     \l_tmpb_tl
1417 }
1418 \cs_gset_eq:NN
1419 \markdownSetupSnippet
1420 \@@_setup_snippet:nn
1421 \msg_new:nnnn
1422 { markdown }
1423 { empty-snippet-name }
1424 { Empty-snippet-name~#1 }
1425 { Pick-a-non-empty-name-for~your~snippet }
1426 \msg_new:nnn
1427 { markdown }
1428 { redefined-snippet }
1429 { Redefined~snippet~#1 }

```

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

```

1430 \tl_new:N
1431 \l_@@_current_snippet_tl
1432 \prg_new_conditional:Nnn
1433 \@@_if_snippet_exists:n
1434 { TF, T, F }
1435 {
1436     \tl_set:NV
1437     \l_@@_current_snippet_tl
1438     \g_@@_current_theme_tl
1439     \tl_put_right:Nn
1440     \l_@@_current_snippet_tl
1441     { #1 }
1442     \prop_if_in:NVTF
1443     \g_@@_snippets_prop

```

```

1444     \l_@@_current_snippet_tl
1445     { \prg_return_true: }
1446     { \prg_return_false: }
1447   }
1448   \cs_gset_eq:NN
1449     \markdownIfSnippetExists
1450     \@@_if_snippet_exists:nTF

```

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

```

1451   \keys_define:nn
1452     { markdown/options }
1453     {
1454       snippet .code:n = {
1455         \tl_set:NV
1456           \l_tmpa_tl
1457           \g_@@_current_theme_tl
1458         \tl_put_right:Nn
1459           \l_tmpa_tl
1460           { #1 }
1461         \@@_if_snippet_exists:nTF
1462           { #1 }
1463           {
1464             \prop_get:NVN
1465               \g_@@_snippets_prop
1466               \l_tmpa_tl
1467               \l_tmpb_tl
1468             \tl_use:N
1469               \l_tmpb_tl
1470           }
1471           {
1472             \msg_error:nnV
1473               { markdown }
1474               { undefined-snippet }
1475               \l_tmpa_tl
1476           }
1477       }
1478     }
1479   \msg_new:nnn
1480     { markdown }
1481     { undefined-snippet }
1482     { Can't~invoke~undefined~snippet~#1 }
1483   \ExplSyntaxOff

```

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

```

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

```

```

}
\begin{markdown}

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}

```

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

```

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown}

```

Alternatively, we can use the extended variant of the `import` L^AT_EX option that allows us to import the `romanNumerals` snippet to the current namespace for easier access:

```

\markdownSetup{
  import = {
    jdoe/lists = romanNumerals,
  },
}

```

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

```

    jdoe/yetanotherlongpackagename,
  },
}

```

```

1484 \ExplSyntaxOn
1485 \tl_new:N
1486   \l_@@_import_current_theme_tl
1487 \keys_define:nn
1488   { markdown/options/import }
1489   {

```

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

```

1490     unknown .default:n = {},
1491     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.

```

1492     \tl_set_eq:NN
1493       \l_@@_import_current_theme_tl
1494       \l_keys_key_str
1495     \tl_replace_all:NVN
1496       \l_@@_import_current_theme_tl
1497       \c_backslash_str
1498       { / }

```

Here, we import the snippets.

```

1499     \clist_map_inline:nn
1500       { #1 }
1501       {
1502         \regex_extract_once:nnNTF
1503           { ^(.*)\s+as\s+(.*)$ }
1504           { ##1 }
1505         \l_tmpa_seq
1506         {
1507           \seq_pop:NN
1508             \l_tmpa_seq
1509             \l_tmpa_tl
1510           \seq_pop:NN
1511             \l_tmpa_seq
1512             \l_tmpa_tl
1513           \seq_pop:NN
1514             \l_tmpa_seq
1515             \l_tmpb_tl

```

```

1516         }
1517         {
1518             \tl_set:Nn
1519             \l_tmpa_tl
1520             { ##1 }
1521             \tl_set:Nn
1522             \l_tmpb_tl
1523             { ##1 }
1524         }
1525         \tl_put_left:Nn
1526         \l_tmpa_tl
1527         { / }
1528         \tl_put_left:NV
1529         \l_tmpa_tl
1530         \l_@@_import_current_theme_tl
1531         \@@_setup_snippet:Vx
1532         \l_tmpb_tl
1533         { snippet = { \l_tmpa_tl } }
1534     }

```

Here, we load the theme.

```

1535         \@@_set_theme:V
1536         \l_@@_import_current_theme_tl
1537     },
1538 }
1539 \cs_generate_variant:Nn
1540 \tl_replace_all:Nnn
1541 { NVn }
1542 \cs_generate_variant:Nn
1543 \@@_set_theme:n
1544 { V }
1545 \cs_generate_variant:Nn
1546 \@@_setup_snippet:nn
1547 { 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.

```

1548 \ExplSyntaxOn
1549 \seq_new:N \g_@@_renderers_seq

```

To enable the reflection of token renderers and their parameters, we will maintain the `\g_@@_renderer_arities_prop` property list.

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

2.2.5.1 Attribute Renderers

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

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

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

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

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

```
1552 \def\markdownRendererAttributeIdentifier{%
1553   \markdownRendererAttributeIdentifierPrototype}%
1554 \ExplSyntaxOn
1555 \seq_gput_right:Nn
1556   \g_@@_renderers_seq
1557   { attributeIdentifier }
1558 \prop_gput:Nnn
1559   \g_@@_renderer_arities_prop
1560   { attributeIdentifier }
1561   { 1 }
1562 \ExplSyntaxOff
1563 \def\markdownRendererAttributeClassName{%
1564   \markdownRendererAttributeClassNamePrototype}%
1565 \ExplSyntaxOn
1566 \seq_gput_right:Nn
1567   \g_@@_renderers_seq
1568   { attributeClassName }
1569 \prop_gput:Nnn
1570   \g_@@_renderer_arities_prop
```



```

1571 { attributeClassName }
1572 { 1 }
1573 \ExplSyntaxOff
1574 \def\markdownRendererAttributeKeyValue{%
1575   \markdownRendererAttributeKeyValuePrototype}%
1576 \ExplSyntaxOn
1577 \seq_gput_right:Nn
1578   \g_@@_renderers_seq
1579   { attributeKeyValue }
1580 \prop_gput:Nnn
1581   \g_@@_renderer_arities_prop
1582   { attributeKeyValue }
1583   { 2 }
1584 \ExplSyntaxOff

```

2.2.5.2 Block Quote Renderers

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

```

1585 \def\markdownRendererBlockQuoteBegin{%
1586   \markdownRendererBlockQuoteBeginPrototype}%
1587 \ExplSyntaxOn
1588 \seq_gput_right:Nn
1589   \g_@@_renderers_seq
1590   { blockQuoteBegin }
1591 \prop_gput:Nnn
1592   \g_@@_renderer_arities_prop
1593   { blockQuoteBegin }
1594   { 0 }
1595 \ExplSyntaxOff

```

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

```

1596 \def\markdownRendererBlockQuoteEnd{%
1597   \markdownRendererBlockQuoteEndPrototype}%
1598 \ExplSyntaxOn
1599 \seq_gput_right:Nn
1600   \g_@@_renderers_seq
1601   { blockQuoteEnd }
1602 \prop_gput:Nnn
1603   \g_@@_renderer_arities_prop
1604   { blockQuoteEnd }
1605   { 0 }
1606 \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.

```

1607 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1608   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1609 \ExplSyntaxOn
1610 \seq_gput_right:Nn
1611   \g_@@_renderers_seq
1612   { bracketedSpanAttributeContextBegin }
1613 \prop_gput:Nnn
1614   \g_@@_renderer_arities_prop
1615   { bracketedSpanAttributeContextBegin }
1616   { 0 }
1617 \ExplSyntaxOff
1618 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1619   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1620 \ExplSyntaxOn
1621 \seq_gput_right:Nn
1622   \g_@@_renderers_seq
1623   { bracketedSpanAttributeContextEnd }
1624 \prop_gput:Nnn
1625   \g_@@_renderer_arities_prop
1626   { bracketedSpanAttributeContextEnd }
1627   { 0 }
1628 \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.

```

1629 \def\markdownRendererUlBegin{%
1630   \markdownRendererUlBeginPrototype}%
1631 \ExplSyntaxOn
1632 \seq_gput_right:Nn
1633   \g_@@_renderers_seq
1634   { ulBegin }
1635 \prop_gput:Nnn
1636   \g_@@_renderer_arities_prop
1637   { ulBegin }
1638   { 0 }
1639 \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.

```

1640 \def\markdownRendererUlBeginTight{%
1641   \markdownRendererUlBeginTightPrototype}%
1642 \ExplSyntaxOn
1643 \seq_gput_right:Nn
1644   \g_@@_renderers_seq
1645   { ulBeginTight }
1646 \prop_gput:Nnn
1647   \g_@@_renderer_arities_prop
1648   { ulBeginTight }
1649   { 0 }
1650 \ExplSyntaxOff

```

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

```

1651 \def\markdownRendererUlItem{%
1652   \markdownRendererUlItemPrototype}%
1653 \ExplSyntaxOn
1654 \seq_gput_right:Nn
1655   \g_@@_renderers_seq
1656   { ulItem }
1657 \prop_gput:Nnn
1658   \g_@@_renderer_arities_prop
1659   { ulItem }
1660   { 0 }
1661 \ExplSyntaxOff

```

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

```

1662 \def\markdownRendererUlItemEnd{%
1663   \markdownRendererUlItemEndPrototype}%
1664 \ExplSyntaxOn
1665 \seq_gput_right:Nn
1666   \g_@@_renderers_seq
1667   { ulItemEnd }
1668 \prop_gput:Nnn
1669   \g_@@_renderer_arities_prop
1670   { ulItemEnd }
1671   { 0 }
1672 \ExplSyntaxOff

```

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

```

1673 \def\markdownRendererUEnd{%
1674   \markdownRendererUEndPrototype}%
1675 \ExplSyntaxOn
1676 \seq_gput_right:Nn
1677   \g_@@_renderers_seq
1678   { ulEnd }
1679 \prop_gput:Nnn
1680   \g_@@_renderer_arities_prop
1681   { ulEnd }
1682   { 0 }
1683 \ExplSyntaxOff

```

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

```

1684 \def\markdownRendererUEndTight{%
1685   \markdownRendererUEndTightPrototype}%
1686 \ExplSyntaxOn
1687 \seq_gput_right:Nn
1688   \g_@@_renderers_seq
1689   { ulEndTight }
1690 \prop_gput:Nnn
1691   \g_@@_renderer_arities_prop
1692   { ulEndTight }
1693   { 0 }
1694 \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.

```

1695 \def\markdownRendererCite{%
1696   \markdownRendererCitePrototype}%
1697 \ExplSyntaxOn
1698 \seq_gput_right:Nn
1699   \g_@@_renderers_seq
1700   { cite }

```

```

1701 \prop_gput:Nnn
1702   \g_@@_renderer_arities_prop
1703   { cite }
1704   { 1 }
1705 \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.

```

1706 \def\markdownRendererTextCite{%
1707   \markdownRendererTextCitePrototype}%
1708 \ExplSyntaxOn
1709 \seq_gput_right:Nn
1710   \g_@@_renderers_seq
1711   { textCite }
1712 \prop_gput:Nnn
1713   \g_@@_renderer_arities_prop
1714   { textCite }
1715   { 1 }
1716 \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.

```

1717 \def\markdownRendererInputVerbatim{%
1718   \markdownRendererInputVerbatimPrototype}%
1719 \ExplSyntaxOn
1720 \seq_gput_right:Nn
1721   \g_@@_renderers_seq
1722   { inputVerbatim }
1723 \prop_gput:Nnn
1724   \g_@@_renderer_arities_prop
1725   { inputVerbatim }
1726   { 1 }
1727 \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.

```

1728 \def\markdownRendererInputFencedCode{%
1729   \markdownRendererInputFencedCodePrototype}%

```

```

1730 \ExplSyntaxOn
1731 \seq_gput_right:Nn
1732   \g_@@_renderers_seq
1733   { inputFencedCode }
1734 \prop_gput:Nnn
1735   \g_@@_renderer_arities_prop
1736   { inputFencedCode }
1737   { 3 }
1738 \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.

```

1739 \def\markdownRendererCodeSpan{%
1740   \markdownRendererCodeSpanPrototype}%
1741 \ExplSyntaxOn
1742 \seq_gput_right:Nn
1743   \g_@@_renderers_seq
1744   { codeSpan }
1745 \prop_gput:Nnn
1746   \g_@@_renderer_arities_prop
1747   { codeSpan }
1748   { 1 }
1749 \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.

```

1750 \def\markdownRendererCodeSpanAttributeContextBegin{%
1751   \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1752 \ExplSyntaxOn
1753 \seq_gput_right:Nn
1754   \g_@@_renderers_seq
1755   { codeSpanAttributeContextBegin }
1756 \prop_gput:Nnn
1757   \g_@@_renderer_arities_prop
1758   { codeSpanAttributeContextBegin }
1759   { 0 }
1760 \ExplSyntaxOff
1761 \def\markdownRendererCodeSpanAttributeContextEnd{%
1762   \markdownRendererCodeSpanAttributeContextEndPrototype}%

```

```

1763 \ExplSyntaxOn
1764 \seq_gput_right:Nn
1765   \g_@@_renderers_seq
1766   { codeSpanAttributeContextEnd }
1767 \prop_gput:Nnn
1768   \g_@@_renderer_arities_prop
1769   { codeSpanAttributeContextEnd }
1770   { 0 }
1771 \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.

```

1772 \def\markdownRendererContentBlock{%
1773   \markdownRendererContentBlockPrototype}%
1774 \ExplSyntaxOn
1775 \seq_gput_right:Nn
1776   \g_@@_renderers_seq
1777   { contentBlock }
1778 \prop_gput:Nnn
1779   \g_@@_renderer_arities_prop
1780   { contentBlock }
1781   { 4 }
1782 \ExplSyntaxOff

```

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

```

1783 \def\markdownRendererContentBlockOnlineImage{%
1784   \markdownRendererContentBlockOnlineImagePrototype}%
1785 \ExplSyntaxOn
1786 \seq_gput_right:Nn
1787   \g_@@_renderers_seq
1788   { contentBlockOnlineImage }
1789 \prop_gput:Nnn
1790   \g_@@_renderer_arities_prop
1791   { contentBlockOnlineImage }
1792   { 4 }
1793 \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 kpathsea³² 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.

```

1794 \def\markdownRendererContentBlockCode{%
1795   \markdownRendererContentBlockCodePrototype}%
1796 \ExplSyntaxOn
1797 \seq_gput_right:Nn
1798   \g_@@_renderers_seq
1799   { contentBlockCode }
1800 \prop_gput:Nnn
1801   \g_@@_renderer_arities_prop
1802   { contentBlockCode }
1803   { 5 }
1804 \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.

```

1805 \def\markdownRendererDlBegin{%
1806   \markdownRendererDlBeginPrototype}%
1807 \ExplSyntaxOn
1808 \seq_gput_right:Nn
1809   \g_@@_renderers_seq
1810   { dlBegin }
1811 \prop_gput:Nnn
1812   \g_@@_renderer_arities_prop
1813   { dlBegin }
1814   { 0 }
1815 \ExplSyntaxOff

```

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

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.

```

1816 \def\markdownRendererDlBeginTight{%
1817   \markdownRendererDlBeginTightPrototype}%
1818 \ExplSyntaxOn
1819 \seq_gput_right:Nn
1820   \g_@@_renderers_seq
1821   { dlBeginTight }
1822 \prop_gput:Nnn
1823   \g_@@_renderer_arities_prop
1824   { dlBeginTight }
1825   { 0 }
1826 \ExplSyntaxOff

```

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

```

1827 \def\markdownRendererDlItem{%
1828   \markdownRendererDlItemPrototype}%
1829 \ExplSyntaxOn
1830 \seq_gput_right:Nn
1831   \g_@@_renderers_seq
1832   { dlItem }
1833 \prop_gput:Nnn
1834   \g_@@_renderer_arities_prop
1835   { dlItem }
1836   { 1 }
1837 \ExplSyntaxOff

```

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

```

1838 \def\markdownRendererDlItemEnd{%
1839   \markdownRendererDlItemEndPrototype}%
1840 \ExplSyntaxOn
1841 \seq_gput_right:Nn
1842   \g_@@_renderers_seq
1843   { dlItemEnd }
1844 \prop_gput:Nnn
1845   \g_@@_renderer_arities_prop
1846   { dlItemEnd }
1847   { 0 }
1848 \ExplSyntaxOff

```

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

```

1849 \def\markdownRendererDlDefinitionBegin{%
1850   \markdownRendererDlDefinitionBeginPrototype}%
1851 \ExplSyntaxOn
1852 \seq_gput_right:Nn
1853   \g_@@_renderers_seq
1854   { dlDefinitionBegin }
1855 \prop_gput:Nnn
1856   \g_@@_renderer_arities_prop
1857   { dlDefinitionBegin }
1858   { 0 }
1859 \ExplSyntaxOff

```

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

```

1860 \def\markdownRendererDlDefinitionEnd{%
1861   \markdownRendererDlDefinitionEndPrototype}%
1862 \ExplSyntaxOn
1863 \seq_gput_right:Nn
1864   \g_@@_renderers_seq
1865   { dlDefinitionEnd }
1866 \prop_gput:Nnn
1867   \g_@@_renderer_arities_prop
1868   { dlDefinitionEnd }
1869   { 0 }
1870 \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.

```

1871 \def\markdownRendererDlEnd{%
1872   \markdownRendererDlEndPrototype}%
1873 \ExplSyntaxOn
1874 \seq_gput_right:Nn
1875   \g_@@_renderers_seq
1876   { dlEnd }
1877 \prop_gput:Nnn
1878   \g_@@_renderer_arities_prop
1879   { dlEnd }
1880   { 0 }
1881 \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.

```

1882 \def\markdownRendererDlEndTight{%

```

```

1883 \markdownRendererDlEndTightPrototype}%
1884 \ExplSyntaxOn
1885 \seq_gput_right:Nn
1886 \g_@@_renderers_seq
1887 { dlEndTight }
1888 \prop_gput:Nnn
1889 \g_@@_renderer_arities_prop
1890 { dlEndTight }
1891 { 0 }
1892 \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.

```

1893 \def\markdownRendererEllipsis{%
1894 \markdownRendererEllipsisPrototype}%
1895 \ExplSyntaxOn
1896 \seq_gput_right:Nn
1897 \g_@@_renderers_seq
1898 { ellipsis }
1899 \prop_gput:Nnn
1900 \g_@@_renderer_arities_prop
1901 { ellipsis }
1902 { 0 }
1903 \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.

```

1904 \def\markdownRendererEmphasis{%
1905 \markdownRendererEmphasisPrototype}%
1906 \ExplSyntaxOn
1907 \seq_gput_right:Nn
1908 \g_@@_renderers_seq
1909 { emphasis }
1910 \prop_gput:Nnn
1911 \g_@@_renderer_arities_prop
1912 { emphasis }
1913 { 1 }
1914 \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.

```

1915 \def\markdownRendererStrongEmphasis{%
1916   \markdownRendererStrongEmphasisPrototype}%
1917 \ExplSyntaxOn
1918 \seq_gput_right:Nn
1919   \g_@@_renderers_seq
1920   { strongEmphasis }
1921 \prop_gput:Nnn
1922   \g_@@_renderer_arities_prop
1923   { strongEmphasis }
1924   { 1 }
1925 \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.

```

1926 \def\markdownRendererFencedCodeAttributeContextBegin{%
1927   \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1928 \ExplSyntaxOn
1929 \seq_gput_right:Nn
1930   \g_@@_renderers_seq
1931   { fencedCodeAttributeContextBegin }
1932 \prop_gput:Nnn
1933   \g_@@_renderer_arities_prop
1934   { fencedCodeAttributeContextBegin }
1935   { 0 }
1936 \ExplSyntaxOff
1937 \def\markdownRendererFencedCodeAttributeContextEnd{%
1938   \markdownRendererFencedCodeAttributeContextEndPrototype}%
1939 \ExplSyntaxOn
1940 \seq_gput_right:Nn
1941   \g_@@_renderers_seq
1942   { fencedCodeAttributeContextEnd }
1943 \prop_gput:Nnn
1944   \g_@@_renderer_arities_prop
1945   { fencedCodeAttributeContextEnd }
1946   { 0 }
1947 \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.

```

1948 \def\markdownRendererFencedDivAttributeContextBegin{%
1949   \markdownRendererFencedDivAttributeContextBeginPrototype}%
1950 \ExplSyntaxOn
1951 \seq_gput_right:Nn
1952   \g_@@_renderers_seq
1953   { fencedDivAttributeContextBegin }
1954 \prop_gput:Nnn
1955   \g_@@_renderer_arities_prop
1956   { fencedDivAttributeContextBegin }
1957   { 0 }
1958 \ExplSyntaxOff
1959 \def\markdownRendererFencedDivAttributeContextEnd{%
1960   \markdownRendererFencedDivAttributeContextEndPrototype}%
1961 \ExplSyntaxOn
1962 \seq_gput_right:Nn
1963   \g_@@_renderers_seq
1964   { fencedDivAttributeContextEnd }
1965 \prop_gput:Nnn
1966   \g_@@_renderer_arities_prop
1967   { fencedDivAttributeContextEnd }
1968   { 0 }
1969 \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.

```

1970 \def\markdownRendererHeaderAttributeContextBegin{%
1971   \markdownRendererHeaderAttributeContextBeginPrototype}%
1972 \ExplSyntaxOn
1973 \seq_gput_right:Nn
1974   \g_@@_renderers_seq
1975   { headerAttributeContextBegin }
1976 \prop_gput:Nnn
1977   \g_@@_renderer_arities_prop
1978   { headerAttributeContextBegin }
1979   { 0 }
1980 \ExplSyntaxOff
1981 \def\markdownRendererHeaderAttributeContextEnd{%

```

```

1982 \markdownRendererHeaderAttributeContextEndPrototype}%
1983 \ExplSyntaxOn
1984 \seq_gput_right:Nn
1985 \g_@@_renderers_seq
1986 { headerAttributeContextEnd }
1987 \prop_gput:Nnn
1988 \g_@@_renderer_arities_prop
1989 { headerAttributeContextEnd }
1990 { 0 }
1991 \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.

```

1992 \def\markdownRendererHeadingOne{%
1993 \markdownRendererHeadingOnePrototype}%
1994 \ExplSyntaxOn
1995 \seq_gput_right:Nn
1996 \g_@@_renderers_seq
1997 { headingOne }
1998 \prop_gput:Nnn
1999 \g_@@_renderer_arities_prop
2000 { headingOne }
2001 { 1 }
2002 \ExplSyntaxOff

```

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

```

2003 \def\markdownRendererHeadingTwo{%
2004 \markdownRendererHeadingTwoPrototype}%
2005 \ExplSyntaxOn
2006 \seq_gput_right:Nn
2007 \g_@@_renderers_seq
2008 { headingTwo }
2009 \prop_gput:Nnn
2010 \g_@@_renderer_arities_prop
2011 { headingTwo }
2012 { 1 }
2013 \ExplSyntaxOff

```

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

```

2014 \def\markdownRendererHeadingThree{%
2015 \markdownRendererHeadingThreePrototype}%
2016 \ExplSyntaxOn
2017 \seq_gput_right:Nn

```

```

2018 \g_@@_renderers_seq
2019 { headingThree }
2020 \prop_gput:Nnn
2021 \g_@@_renderer_arities_prop
2022 { headingThree }
2023 { 1 }
2024 \ExplSyntaxOff

```

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

```

2025 \def\markdownRendererHeadingFour{%
2026 \markdownRendererHeadingFourPrototype}%
2027 \ExplSyntaxOn
2028 \seq_gput_right:Nn
2029 \g_@@_renderers_seq
2030 { headingFour }
2031 \prop_gput:Nnn
2032 \g_@@_renderer_arities_prop
2033 { headingFour }
2034 { 1 }
2035 \ExplSyntaxOff

```

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

```

2036 \def\markdownRendererHeadingFive{%
2037 \markdownRendererHeadingFivePrototype}%
2038 \ExplSyntaxOn
2039 \seq_gput_right:Nn
2040 \g_@@_renderers_seq
2041 { headingFive }
2042 \prop_gput:Nnn
2043 \g_@@_renderer_arities_prop
2044 { headingFive }
2045 { 1 }
2046 \ExplSyntaxOff

```

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

```

2047 \def\markdownRendererHeadingSix{%
2048 \markdownRendererHeadingSixPrototype}%
2049 \ExplSyntaxOn
2050 \seq_gput_right:Nn
2051 \g_@@_renderers_seq
2052 { headingSix }
2053 \prop_gput:Nnn
2054 \g_@@_renderer_arities_prop
2055 { headingSix }

```

```

2056 { 1 }
2057 \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.

```

2058 \def\markdownRendererInlineHtmlComment{%
2059   \markdownRendererInlineHtmlCommentPrototype}%
2060 \ExplSyntaxOn
2061 \seq_gput_right:Nn
2062   \g_@@_renderers_seq
2063   { inlineHtmlComment }
2064 \prop_gput:Nnn
2065   \g_@@_renderer_arities_prop
2066   { inlineHtmlComment }
2067   { 1 }
2068 \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.

```

2069 \def\markdownRendererInlineHtmlTag{%
2070   \markdownRendererInlineHtmlTagPrototype}%
2071 \ExplSyntaxOn
2072 \seq_gput_right:Nn
2073   \g_@@_renderers_seq
2074   { inlineHtmlTag }
2075 \prop_gput:Nnn
2076   \g_@@_renderer_arities_prop
2077   { inlineHtmlTag }
2078   { 1 }
2079 \ExplSyntaxOff
2080 \def\markdownRendererInputBlockHtmlElement{%
2081   \markdownRendererInputBlockHtmlElementPrototype}%
2082 \ExplSyntaxOn
2083 \seq_gput_right:Nn

```



```

2084 \g_@@_renderers_seq
2085 { inputBlockHtmlElement }
2086 \prop_gput:Nnn
2087 \g_@@_renderer_arities_prop
2088 { inputBlockHtmlElement }
2089 { 1 }
2090 \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.

```

2091 \def\markdownRendererImage{%
2092 \markdownRendererImagePrototype}%
2093 \ExplSyntaxOn
2094 \seq_gput_right:Nn
2095 \g_@@_renderers_seq
2096 { image }
2097 \prop_gput:Nnn
2098 \g_@@_renderer_arities_prop
2099 { image }
2100 { 4 }
2101 \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.

```

2102 \def\markdownRendererImageAttributeContextBegin{%
2103 \markdownRendererImageAttributeContextBeginPrototype}%
2104 \ExplSyntaxOn
2105 \seq_gput_right:Nn
2106 \g_@@_renderers_seq
2107 { imageAttributeContextBegin }
2108 \prop_gput:Nnn
2109 \g_@@_renderer_arities_prop
2110 { imageAttributeContextBegin }
2111 { 0 }
2112 \ExplSyntaxOff
2113 \def\markdownRendererImageAttributeContextEnd{%
2114 \markdownRendererImageAttributeContextEndPrototype}%
2115 \ExplSyntaxOn

```

```

2116 \seq_gput_right:Nn
2117   \g_@@_renderers_seq
2118   { imageAttributeContextEnd }
2119 \prop_gput:Nnn
2120   \g_@@_renderer_arities_prop
2121   { imageAttributeContextEnd }
2122   { 0 }
2123 \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.

```

2124 \def\markdownRendererInterblockSeparator{%
2125   \markdownRendererInterblockSeparatorPrototype}%
2126 \ExplSyntaxOn
2127 \seq_gput_right:Nn
2128   \g_@@_renderers_seq
2129   { interblockSeparator }
2130 \prop_gput:Nnn
2131   \g_@@_renderer_arities_prop
2132   { interblockSeparator }
2133   { 0 }
2134 \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.

```

2135 \def\markdownRendererParagraphSeparator{%
2136   \markdownRendererParagraphSeparatorPrototype}%
2137 \ExplSyntaxOn
2138 \seq_gput_right:Nn
2139   \g_@@_renderers_seq
2140   { paragraphSeparator }
2141 \prop_gput:Nnn
2142   \g_@@_renderer_arities_prop
2143   { paragraphSeparator }
2144   { 0 }
2145 \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.

```

2146 \def\markdownRendererLineBlockBegin{%
2147   \markdownRendererLineBlockBeginPrototype}%
2148 \ExplSyntaxOn
2149 \seq_gput_right:Nn
2150   \g_@@_renderers_seq
2151   { lineBlockBegin }
2152 \prop_gput:Nnn
2153   \g_@@_renderer_arities_prop
2154   { lineBlockBegin }
2155   { 0 }
2156 \ExplSyntaxOff
2157 \def\markdownRendererLineBlockEnd{%
2158   \markdownRendererLineBlockEndPrototype}%
2159 \ExplSyntaxOn
2160 \seq_gput_right:Nn
2161   \g_@@_renderers_seq
2162   { lineBlockEnd }
2163 \prop_gput:Nnn
2164   \g_@@_renderer_arities_prop
2165   { lineBlockEnd }
2166   { 0 }
2167 \ExplSyntaxOff

```

2.2.5.23 Line Break Renderers

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

```

2168 \def\markdownRendererSoftLineBreak{%
2169   \markdownRendererSoftLineBreakPrototype}%
2170 \ExplSyntaxOn
2171 \seq_gput_right:Nn
2172   \g_@@_renderers_seq
2173   { softLineBreak }
2174 \prop_gput:Nnn
2175   \g_@@_renderer_arities_prop
2176   { softLineBreak }
2177   { 0 }
2178 \ExplSyntaxOff

```

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

```

2179 \def\markdownRendererHardLineBreak{%
2180   \markdownRendererHardLineBreakPrototype}%

```

```

2181 \ExplSyntaxOn
2182 \seq_gput_right:Nn
2183   \g_@@_renderers_seq
2184   { hardLineBreak }
2185 \prop_gput:Nnn
2186   \g_@@_renderer_arities_prop
2187   { hardLineBreak }
2188   { 0 }
2189 \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.

```

2190 \def\markdownRendererLink{%
2191   \markdownRendererLinkPrototype}%
2192 \ExplSyntaxOn
2193 \seq_gput_right:Nn
2194   \g_@@_renderers_seq
2195   { link }
2196 \prop_gput:Nnn
2197   \g_@@_renderer_arities_prop
2198   { link }
2199   { 4 }
2200 \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.

```

2201 \def\markdownRendererLinkAttributeContextBegin{%
2202   \markdownRendererLinkAttributeContextBeginPrototype}%
2203 \ExplSyntaxOn
2204 \seq_gput_right:Nn
2205   \g_@@_renderers_seq
2206   { linkAttributeContextBegin }
2207 \prop_gput:Nnn
2208   \g_@@_renderer_arities_prop
2209   { linkAttributeContextBegin }
2210   { 0 }
2211 \ExplSyntaxOff
2212 \def\markdownRendererLinkAttributeContextEnd{%

```

```

2213 \markdownRendererLinkAttributeContextEndPrototype}%
2214 \ExplSyntaxOn
2215 \seq_gput_right:Nn
2216 \g_@@_renderers_seq
2217 { linkAttributeContextEnd }
2218 \prop_gput:Nnn
2219 \g_@@_renderer_arities_prop
2220 { linkAttributeContextEnd }
2221 { 0 }
2222 \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.

```

2223 \def\markdownRendererMark{%
2224 \markdownRendererMarkPrototype}%
2225 \ExplSyntaxOn
2226 \seq_gput_right:Nn
2227 \g_@@_renderers_seq
2228 { mark }
2229 \prop_gput:Nnn
2230 \g_@@_renderer_arities_prop
2231 { mark }
2232 { 1 }
2233 \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 \TeX document may contain any number of markdown documents. Additionally, markdown documents may appear not only in a sequence, but several markdown documents may also be *nested*. Redefinitions of the macros should take this into account.

```

2234 \def\markdownRendererDocumentBegin{%
2235 \markdownRendererDocumentBeginPrototype}%
2236 \ExplSyntaxOn
2237 \seq_gput_right:Nn
2238 \g_@@_renderers_seq
2239 { documentBegin }
2240 \prop_gput:Nnn
2241 \g_@@_renderer_arities_prop
2242 { documentBegin }

```

```

2243 { 0 }
2244 \ExplSyntaxOff
2245 \def\markdownRendererDocumentEnd{%
2246   \markdownRendererDocumentEndPrototype}%
2247 \ExplSyntaxOn
2248 \seq_gput_right:Nn
2249   \g_@@_renderers_seq
2250   { documentEnd }
2251 \prop_gput:Nnn
2252   \g_@@_renderer_arities_prop
2253   { documentEnd }
2254   { 0 }
2255 \ExplSyntaxOff

```

2.2.5.28 Non-Breaking Space Renderer

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

```

2256 \def\markdownRendererNbsp{%
2257   \markdownRendererNbspPrototype}%
2258 \ExplSyntaxOn
2259 \seq_gput_right:Nn
2260   \g_@@_renderers_seq
2261   { nbsp }
2262 \prop_gput:Nnn
2263   \g_@@_renderer_arities_prop
2264   { nbsp }
2265   { 0 }
2266 \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.

```

2267 \def\markdownRendererNote{%
2268   \markdownRendererNotePrototype}%
2269 \ExplSyntaxOn
2270 \seq_gput_right:Nn
2271   \g_@@_renderers_seq
2272   { note }
2273 \prop_gput:Nnn
2274   \g_@@_renderer_arities_prop
2275   { note }
2276   { 1 }
2277 \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.

```
2278 \def\markdownRendererOlBegin{%
2279   \markdownRendererOlBeginPrototype}%
2280 \ExplSyntaxOn
2281 \seq_gput_right:Nn
2282   \g_@@_renderers_seq
2283   { olBegin }
2284 \prop_gput:Nnn
2285   \g_@@_renderer_arities_prop
2286   { olBegin }
2287   { 0 }
2288 \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.

```
2289 \def\markdownRendererOlBeginTight{%
2290   \markdownRendererOlBeginTightPrototype}%
2291 \ExplSyntaxOn
2292 \seq_gput_right:Nn
2293   \g_@@_renderers_seq
2294   { olBeginTight }
2295 \prop_gput:Nnn
2296   \g_@@_renderer_arities_prop
2297   { olBeginTight }
2298   { 0 }
2299 \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`).

```
2300 \def\markdownRendererFancyOlBegin{%
2301   \markdownRendererFancyOlBeginPrototype}%
2302 \ExplSyntaxOn
2303 \seq_gput_right:Nn
2304   \g_@@_renderers_seq
2305   { fancyOlBegin }
```

```

2306 \prop_gput:Nnn
2307   \g_@@_renderer_arities_prop
2308   { fancyOlBegin }
2309   { 2 }
2310 \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.

```

2311 \def\markdownRendererFancyOlBeginTight{%
2312   \markdownRendererFancyOlBeginTightPrototype}%
2313 \ExplSyntaxOn
2314 \seq_gput_right:Nn
2315   \g_@@_renderers_seq
2316   { fancyOlBeginTight }
2317 \prop_gput:Nnn
2318   \g_@@_renderer_arities_prop
2319   { fancyOlBeginTight }
2320   { 2 }
2321 \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.

```

2322 \def\markdownRendererOlItem{%
2323   \markdownRendererOlItemPrototype}%
2324 \ExplSyntaxOn
2325 \seq_gput_right:Nn
2326   \g_@@_renderers_seq
2327   { olItem }
2328 \prop_gput:Nnn
2329   \g_@@_renderer_arities_prop
2330   { olItem }
2331   { 0 }
2332 \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.

```

2333 \def\markdownRendererOlItemEnd{%
2334   \markdownRendererOlItemEndPrototype}%
2335 \ExplSyntaxOn
2336 \seq_gput_right:Nn

```



```

2337 \g_@@_renderers_seq
2338 { olItemEnd }
2339 \prop_gput:Nnn
2340 \g_@@_renderer_arities_prop
2341 { olItemEnd }
2342 { 0 }
2343 \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.

```

2344 \def\markdownRendererOlItemWithNumber{%
2345 \markdownRendererOlItemWithNumberPrototype}%
2346 \ExplSyntaxOn
2347 \seq_gput_right:Nn
2348 \g_@@_renderers_seq
2349 { olItemWithNumber }
2350 \prop_gput:Nnn
2351 \g_@@_renderer_arities_prop
2352 { olItemWithNumber }
2353 { 1 }
2354 \ExplSyntaxOff

```

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

```

2355 \def\markdownRendererFancyOlItem{%
2356 \markdownRendererFancyOlItemPrototype}%
2357 \ExplSyntaxOn
2358 \seq_gput_right:Nn
2359 \g_@@_renderers_seq
2360 { fancyOlItem }
2361 \prop_gput:Nnn
2362 \g_@@_renderer_arities_prop
2363 { fancyOlItem }
2364 { 0 }
2365 \ExplSyntaxOff

```

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

```

2366 \def\markdownRendererFancyOlItemEnd{%
2367 \markdownRendererFancyOlItemEndPrototype}%
2368 \ExplSyntaxOn
2369 \seq_gput_right:Nn

```

```

2370 \g_@@_renderers_seq
2371 { fancyOliItemEnd }
2372 \prop_gput:Nnn
2373 \g_@@_renderer_arities_prop
2374 { fancyOliItemEnd }
2375 { 0 }
2376 \ExplSyntaxOff

```

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

```

2377 \def\markdownRendererFancyOliItemWithNumber{%
2378 \markdownRendererFancyOliItemWithNumberPrototype}%
2379 \ExplSyntaxOn
2380 \seq_gput_right:Nn
2381 \g_@@_renderers_seq
2382 { fancyOliItemWithNumber }
2383 \prop_gput:Nnn
2384 \g_@@_renderer_arities_prop
2385 { fancyOliItemWithNumber }
2386 { 1 }
2387 \ExplSyntaxOff

```

The `\markdownRendererOliEnd` 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.

```

2388 \def\markdownRendererOliEnd{%
2389 \markdownRendererOliEndPrototype}%
2390 \ExplSyntaxOn
2391 \seq_gput_right:Nn
2392 \g_@@_renderers_seq
2393 { oliEnd }
2394 \prop_gput:Nnn
2395 \g_@@_renderer_arities_prop
2396 { oliEnd }
2397 { 0 }
2398 \ExplSyntaxOff

```

The `\markdownRendererOliEndTight` 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.

```

2399 \def\markdownRendererOliEndTight{%
2400 \markdownRendererOliEndTightPrototype}%

```

```

2401 \ExplSyntaxOn
2402 \seq_gput_right:Nn
2403   \g_@@_renderers_seq
2404   { olEndTight }
2405 \prop_gput:Nnn
2406   \g_@@_renderer_arities_prop
2407   { olEndTight }
2408   { 0 }
2409 \ExplSyntaxOff

```

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

```

2410 \def\markdownRendererFancyOlEnd{%
2411   \markdownRendererFancyOlEndPrototype}%
2412 \ExplSyntaxOn
2413 \seq_gput_right:Nn
2414   \g_@@_renderers_seq
2415   { fancyOlEnd }
2416 \prop_gput:Nnn
2417   \g_@@_renderer_arities_prop
2418   { fancyOlEnd }
2419   { 0 }
2420 \ExplSyntaxOff

```

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

```

2421 \def\markdownRendererFancyOlEndTight{%
2422   \markdownRendererFancyOlEndTightPrototype}%
2423 \ExplSyntaxOn
2424 \seq_gput_right:Nn
2425   \g_@@_renderers_seq
2426   { fancyOlEndTight }
2427 \prop_gput:Nnn
2428   \g_@@_renderer_arities_prop
2429   { fancyOlEndTight }
2430   { 0 }
2431 \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.

```

2432 \def\markdownRendererInputRawInline{%
2433   \markdownRendererInputRawInlinePrototype}%
2434 \ExplSyntaxOn
2435 \seq_gput_right:Nn
2436   \g_@@_renderers_seq
2437   { inputRawInline }
2438 \prop_gput:Nnn
2439   \g_@@_renderer_arities_prop
2440   { inputRawInline }
2441   { 2 }
2442 \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.

```

2443 \def\markdownRendererInputRawBlock{%
2444   \markdownRendererInputRawBlockPrototype}%
2445 \ExplSyntaxOn
2446 \seq_gput_right:Nn
2447   \g_@@_renderers_seq
2448   { inputRawBlock }
2449 \prop_gput:Nnn
2450   \g_@@_renderer_arities_prop
2451   { inputRawBlock }
2452   { 2 }
2453 \ExplSyntaxOff

```

2.2.5.32 Section Renderers

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

```

2454 \def\markdownRendererSectionBegin{%
2455   \markdownRendererSectionBeginPrototype}%
2456 \ExplSyntaxOn
2457 \seq_gput_right:Nn
2458   \g_@@_renderers_seq
2459   { sectionBegin }
2460 \prop_gput:Nnn
2461   \g_@@_renderer_arities_prop
2462   { sectionBegin }
2463   { 0 }
2464 \ExplSyntaxOff
2465 \def\markdownRendererSectionEnd{%

```

```

2466 \markdownRendererSectionEndPrototype}%
2467 \ExplSyntaxOn
2468 \seq_gput_right:Nn
2469 \g_@@_renderers_seq
2470 { sectionEnd }
2471 \prop_gput:Nnn
2472 \g_@@_renderer_arities_prop
2473 { sectionEnd }
2474 { 0 }
2475 \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.

```

2476 \def\markdownRendererReplacementCharacter{%
2477 \markdownRendererReplacementCharacterPrototype}%
2478 \ExplSyntaxOn
2479 \seq_gput_right:Nn
2480 \g_@@_renderers_seq
2481 { replacementCharacter }
2482 \prop_gput:Nnn
2483 \g_@@_renderer_arities_prop
2484 { replacementCharacter }
2485 { 0 }
2486 \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`.

```

2487 \def\markdownRendererLeftBrace{%
2488 \markdownRendererLeftBracePrototype}%
2489 \ExplSyntaxOn
2490 \seq_gput_right:Nn
2491 \g_@@_renderers_seq
2492 { leftBrace }
2493 \prop_gput:Nnn
2494 \g_@@_renderer_arities_prop
2495 { leftBrace }
2496 { 0 }
2497 \ExplSyntaxOff
2498 \def\markdownRendererRightBrace{%
2499 \markdownRendererRightBracePrototype}%
2500 \ExplSyntaxOn
2501 \seq_gput_right:Nn

```

```

2502 \g_@@_renderers_seq
2503 { rightBrace }
2504 \prop_gput:Nnn
2505 \g_@@_renderer_arities_prop
2506 { rightBrace }
2507 { 0 }
2508 \ExplSyntaxOff
2509 \def\markdownRendererDollarSign{%
2510 \markdownRendererDollarSignPrototype}%
2511 \ExplSyntaxOn
2512 \seq_gput_right:Nn
2513 \g_@@_renderers_seq
2514 { dollarSign }
2515 \prop_gput:Nnn
2516 \g_@@_renderer_arities_prop
2517 { dollarSign }
2518 { 0 }
2519 \ExplSyntaxOff
2520 \def\markdownRendererPercentSign{%
2521 \markdownRendererPercentSignPrototype}%
2522 \ExplSyntaxOn
2523 \seq_gput_right:Nn
2524 \g_@@_renderers_seq
2525 { percentSign }
2526 \prop_gput:Nnn
2527 \g_@@_renderer_arities_prop
2528 { percentSign }
2529 { 0 }
2530 \ExplSyntaxOff
2531 \def\markdownRendererAmpersand{%
2532 \markdownRendererAmpersandPrototype}%
2533 \ExplSyntaxOn
2534 \seq_gput_right:Nn
2535 \g_@@_renderers_seq
2536 { ampersand }
2537 \prop_gput:Nnn
2538 \g_@@_renderer_arities_prop
2539 { ampersand }
2540 { 0 }
2541 \ExplSyntaxOff
2542 \def\markdownRendererUnderscore{%
2543 \markdownRendererUnderscorePrototype}%
2544 \ExplSyntaxOn
2545 \seq_gput_right:Nn
2546 \g_@@_renderers_seq
2547 { underscore }
2548 \prop_gput:Nnn

```

```

2549 \g_@@_renderer_arities_prop
2550 { underscore }
2551 { 0 }
2552 \ExplSyntaxOff
2553 \def\markdownRendererHash{%
2554 \markdownRendererHashPrototype}%
2555 \ExplSyntaxOn
2556 \seq_gput_right:Nn
2557 \g_@@_renderers_seq
2558 { hash }
2559 \prop_gput:Nnn
2560 \g_@@_renderer_arities_prop
2561 { hash }
2562 { 0 }
2563 \ExplSyntaxOff
2564 \def\markdownRendererCircumflex{%
2565 \markdownRendererCircumflexPrototype}%
2566 \ExplSyntaxOn
2567 \seq_gput_right:Nn
2568 \g_@@_renderers_seq
2569 { circumflex }
2570 \prop_gput:Nnn
2571 \g_@@_renderer_arities_prop
2572 { circumflex }
2573 { 0 }
2574 \ExplSyntaxOff
2575 \def\markdownRendererBackslash{%
2576 \markdownRendererBackslashPrototype}%
2577 \ExplSyntaxOn
2578 \seq_gput_right:Nn
2579 \g_@@_renderers_seq
2580 { backslash }
2581 \prop_gput:Nnn
2582 \g_@@_renderer_arities_prop
2583 { backslash }
2584 { 0 }
2585 \ExplSyntaxOff
2586 \def\markdownRendererTilde{%
2587 \markdownRendererTildePrototype}%
2588 \ExplSyntaxOn
2589 \seq_gput_right:Nn
2590 \g_@@_renderers_seq
2591 { tilde }
2592 \prop_gput:Nnn
2593 \g_@@_renderer_arities_prop
2594 { tilde }
2595 { 0 }

```

```

2596 \ExplSyntaxOff
2597 \def\markdownRendererPipe{%
2598   \markdownRendererPipePrototype}%
2599 \ExplSyntaxOn
2600 \seq_gput_right:Nn
2601   \g_@@_renderers_seq
2602   { pipe }
2603 \prop_gput:Nnn
2604   \g_@@_renderer_arities_prop
2605   { pipe }
2606   { 0 }
2607 \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.

```

2608 \def\markdownRendererStrikeThrough{%
2609   \markdownRendererStrikeThroughPrototype}%
2610 \ExplSyntaxOn
2611 \seq_gput_right:Nn
2612   \g_@@_renderers_seq
2613   { strikeThrough }
2614 \prop_gput:Nnn
2615   \g_@@_renderer_arities_prop
2616   { strikeThrough }
2617   { 1 }
2618 \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.

```

2619 \def\markdownRendererSubscript{%
2620   \markdownRendererSubscriptPrototype}%
2621 \ExplSyntaxOn
2622 \seq_gput_right:Nn
2623   \g_@@_renderers_seq
2624   { subscript }
2625 \prop_gput:Nnn
2626   \g_@@_renderer_arities_prop
2627   { subscript }
2628   { 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.

```
2629 \def\markdownRendererSuperscript{%
2630   \markdownRendererSuperscriptPrototype}%
2631 \ExplSyntaxOn
2632 \seq_gput_right:Nn
2633   \g_@@_renderers_seq
2634   { superscript }
2635 \prop_gput:Nnn
2636   \g_@@_renderer_arities_prop
2637   { superscript }
2638   { 1 }
2639 \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.

```
2640 \def\markdownRendererTableAttributeContextBegin{%
2641   \markdownRendererTableAttributeContextBeginPrototype}%
2642 \ExplSyntaxOn
2643 \seq_gput_right:Nn
2644   \g_@@_renderers_seq
2645   { tableAttributeContextBegin }
2646 \prop_gput:Nnn
2647   \g_@@_renderer_arities_prop
2648   { tableAttributeContextBegin }
2649   { 0 }
2650 \ExplSyntaxOff
2651 \def\markdownRendererTableAttributeContextEnd{%
2652   \markdownRendererTableAttributeContextEndPrototype}%
2653 \ExplSyntaxOn
2654 \seq_gput_right:Nn
2655   \g_@@_renderers_seq
2656   { tableAttributeContextEnd }
2657 \prop_gput:Nnn
2658   \g_@@_renderer_arities_prop
2659   { tableAttributeContextEnd }
2660   { 0 }
2661 \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.

```
2662 \def\markdownRendererTable{%
2663   \markdownRendererTablePrototype}%
2664 \ExplSyntaxOn
2665 \seq_gput_right:Nn
2666   \g_@@_renderers_seq
2667   { table }
2668 \prop_gput:Nnn
2669   \g_@@_renderer_arities_prop
2670   { table }
2671   { 3 }
2672 \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.

```
2673 \def\markdownRendererInlineMath{%
2674   \markdownRendererInlineMathPrototype}%
2675 \ExplSyntaxOn
2676 \seq_gput_right:Nn
2677   \g_@@_renderers_seq
2678   { inlineMath }
2679 \prop_gput:Nnn
2680   \g_@@_renderer_arities_prop
2681   { inlineMath }
2682   { 1 }
2683 \ExplSyntaxOff
2684 \def\markdownRendererDisplayMath{%
2685   \markdownRendererDisplayMathPrototype}%
```

```

2686 \ExplSyntaxOn
2687 \seq_gput_right:Nn
2688   \g_@@_renderers_seq
2689   { displayMath }
2690 \prop_gput:Nnn
2691   \g_@@_renderer_arities_prop
2692   { displayMath }
2693   { 1 }
2694 \ExplSyntaxOff

```

2.2.5.41 Thematic Break Renderer

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

```

2695 \def\markdownRendererThematicBreak{%
2696   \markdownRendererThematicBreakPrototype}%
2697 \ExplSyntaxOn
2698 \seq_gput_right:Nn
2699   \g_@@_renderers_seq
2700   { thematicBreak }
2701 \prop_gput:Nnn
2702   \g_@@_renderer_arities_prop
2703   { thematicBreak }
2704   { 0 }
2705 \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.

```

2706 \def\markdownRendererTickedBox{%
2707   \markdownRendererTickedBoxPrototype}%
2708 \ExplSyntaxOn
2709 \seq_gput_right:Nn
2710   \g_@@_renderers_seq
2711   { tickedBox }
2712 \prop_gput:Nnn
2713   \g_@@_renderer_arities_prop
2714   { tickedBox }
2715   { 0 }
2716 \ExplSyntaxOff
2717 \def\markdownRendererHalfTickedBox{%
2718   \markdownRendererHalfTickedBoxPrototype}%

```

```

2719 \ExplSyntaxOn
2720 \seq_gput_right:Nn
2721   \g_@@_renderers_seq
2722   { halfTickedBox }
2723 \prop_gput:Nnn
2724   \g_@@_renderer_arities_prop
2725   { halfTickedBox }
2726   { 0 }
2727 \ExplSyntaxOff
2728 \def\markdownRendererUntickedBox{%
2729   \markdownRendererUntickedBoxPrototype}%
2730 \ExplSyntaxOn
2731 \seq_gput_right:Nn
2732   \g_@@_renderers_seq
2733   { untickedBox }
2734 \prop_gput:Nnn
2735   \g_@@_renderer_arities_prop
2736   { untickedBox }
2737   { 0 }
2738 \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.

```

2739 \def\markdownRendererWarning{%
2740   \markdownRendererWarningPrototype}%
2741 \def\markdownRendererError{%
2742   \markdownRendererErrorPrototype}%
2743 \ExplSyntaxOn
2744 \seq_gput_right:Nn
2745   \g_@@_renderers_seq
2746   { warning }
2747 \prop_gput:Nnn
2748   \g_@@_renderer_arities_prop
2749   { warning }

```

```

2750 { 4 }
2751 \seq_gput_right:Nn
2752 \g_@@_renderers_seq
2753 { error }
2754 \prop_gput:Nnn
2755 \g_@@_renderer_arities_prop
2756 { error }
2757 { 4 }
2758 \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.

```

2759 \def\markdownRendererJekyllDataBegin{%
2760   \markdownRendererJekyllDataBeginPrototype}%
2761 \ExplSyntaxOn
2762 \seq_gput_right:Nn
2763 \g_@@_renderers_seq
2764 { jekyllDataBegin }
2765 \prop_gput:Nnn
2766 \g_@@_renderer_arities_prop
2767 { jekyllDataBegin }
2768 { 0 }
2769 \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.

```

2770 \def\markdownRendererJekyllDataEnd{%
2771   \markdownRendererJekyllDataEndPrototype}%
2772 \ExplSyntaxOn
2773 \seq_gput_right:Nn
2774 \g_@@_renderers_seq
2775 { jekyllDataEnd }
2776 \prop_gput:Nnn
2777 \g_@@_renderer_arities_prop
2778 { jekyllDataEnd }
2779 { 0 }
2780 \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.

```

2781 \def\markdownRendererJekyllDataMappingBegin{%
2782   \markdownRendererJekyllDataMappingBeginPrototype}%
2783 \ExplSyntaxOn
2784 \seq_gput_right:Nn
2785   \g_@@_renderers_seq
2786   { jekyllDataMappingBegin }
2787 \prop_gput:Nnn
2788   \g_@@_renderer_arities_prop
2789   { jekyllDataMappingBegin }
2790   { 2 }
2791 \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.

```

2792 \def\markdownRendererJekyllDataMappingEnd{%
2793   \markdownRendererJekyllDataMappingEndPrototype}%
2794 \ExplSyntaxOn
2795 \seq_gput_right:Nn
2796   \g_@@_renderers_seq
2797   { jekyllDataMappingEnd }
2798 \prop_gput:Nnn
2799   \g_@@_renderer_arities_prop
2800   { jekyllDataMappingEnd }
2801   { 0 }
2802 \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.

```

2803 \def\markdownRendererJekyllDataSequenceBegin{%
2804   \markdownRendererJekyllDataSequenceBeginPrototype}%
2805 \ExplSyntaxOn
2806 \seq_gput_right:Nn
2807   \g_@@_renderers_seq
2808   { jekyllDataSequenceBegin }
2809 \prop_gput:Nnn
2810   \g_@@_renderer_arities_prop
2811   { jekyllDataSequenceBegin }
2812   { 2 }
2813 \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.

```

2814 \def\markdownRendererJekyllDataSequenceEnd{%
2815   \markdownRendererJekyllDataSequenceEndPrototype}%
2816 \ExplSyntaxOn
2817 \seq_gput_right:Nn
2818   \g_@@_renderers_seq
2819   { jekyllDataSequenceEnd }
2820 \prop_gput:Nnn
2821   \g_@@_renderer_arities_prop
2822   { jekyllDataSequenceEnd }
2823   { 0 }
2824 \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.

```

2825 \def\markdownRendererJekyllDataBoolean{%
2826   \markdownRendererJekyllDataBooleanPrototype}%
2827 \ExplSyntaxOn
2828 \seq_gput_right:Nn
2829   \g_@@_renderers_seq
2830   { jekyllDataBoolean }
2831 \prop_gput:Nnn
2832   \g_@@_renderer_arities_prop
2833   { jekyllDataBoolean }
2834   { 2 }
2835 \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.

```

2836 \def\markdownRendererJekyllDataNumber{%
2837   \markdownRendererJekyllDataNumberPrototype}%
2838 \ExplSyntaxOn
2839 \seq_gput_right:Nn
2840   \g_@@_renderers_seq
2841   { jekyllDataNumber }
2842 \prop_gput:Nnn
2843   \g_@@_renderer_arities_prop

```

```

2844 { jekyllDataNumber }
2845 { 2 }
2846 \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 `\markdownRendererJekyllDataProgrammaticString` receives the scalar value after all markdown markup and special \TeX characters in the string have been replaced by \TeX macros, `\markdownRendererJekyllDataTypographicString` 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 \TeX .

```

2847 \def\markdownRendererJekyllDataTypographicString{%
2848   \markdownRendererJekyllDataTypographicStringPrototype}%
2849 \def\markdownRendererJekyllDataProgrammaticString{%
2850   \markdownRendererJekyllDataProgrammaticStringPrototype}%
2851 \ExplSyntaxOn
2852 \seq_gput_right:Nn
2853   \g_@@_renderers_seq
2854   { jekyllDataTypographicString }
2855 \prop_gput:Nnn
2856   \g_@@_renderer_arities_prop
2857   { jekyllDataTypographicString }
2858   { 2 }
2859 \seq_gput_right:Nn
2860   \g_@@_renderers_seq
2861   { jekyllDataProgrammaticString }
2862 \prop_gput:Nnn
2863   \g_@@_renderer_arities_prop
2864   { jekyllDataProgrammaticString }
2865   { 2 }
2866 \ExplSyntaxOff

```

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

```

2867 \ExplSyntaxOn
2868 \cs_gset:Npn
2869   \markdownRendererJekyllDataTypographicString

```



```

2870 {
2871   \cs_if_exist:NTF
2872     \markdownRendererJekyllDataString
2873     {
2874       \@@_if_option:nTF
2875         { experimental }
2876         {
2877           \markdownError
2878             {
2879               The~jekyllDataString~renderer~has~been~deprecated,~
2880               to~be~removed~in~Markdown~4.0.0
2881             }
2882         }
2883       {
2884         \markdownWarning
2885           {
2886             The~jekyllDataString~renderer~has~been~deprecated,~
2887             to~be~removed~in~Markdown~4.0.0
2888           }
2889         \markdownRendererJekyllDataString
2890       }
2891     }
2892   {
2893     \cs_if_exist:NTF
2894       \markdownRendererJekyllDataStringPrototype
2895       {
2896         \@@_if_option:nTF
2897           { experimental }
2898           {
2899             \markdownError
2900               {
2901                 The~jekyllDataString~renderer~prototype~
2902                 has~been~deprecated,~
2903                 to~be~removed~in~Markdown~4.0.0
2904               }
2905           }
2906         {
2907           \markdownWarning
2908             {
2909               The~jekyllDataString~renderer~prototype~
2910               has~been~deprecated,~
2911               to~be~removed~in~Markdown~4.0.0
2912             }
2913           \markdownRendererJekyllDataStringPrototype
2914         }
2915       }
2916     {

```

```

2917         \markdownRendererJekyllDataTypographicStringPrototype
2918     }
2919 }
2920 }
2921 \seq_gput_right:Nn
2922   \g_@@_renderers_seq
2923   { jekyllDataString }
2924 \prop_gput:Nnn
2925   \g_@@_renderer_arities_prop
2926   { jekyllDataString }
2927   { 2 }
2928 \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.

```

2929 \def\markdownRendererJekyllDataEmpty{%
2930   \markdownRendererJekyllDataEmptyPrototype}%
2931 \ExplSyntaxOn
2932 \seq_gput_right:Nn
2933   \g_@@_renderers_seq
2934   { jekyllDataEmpty }
2935 \prop_gput:Nnn
2936   \g_@@_renderer_arities_prop
2937   { jekyllDataEmpty }
2938   { 1 }
2939 \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` key, whose value must be a list of key-values, where the keys correspond to the markdown token renderer macros and the values are new definitions of these token renderers.

```

2940 \ExplSyntaxOn
2941 \cs_new:Nn \@@_define_renderers:
2942   {
2943     \seq_map_inline:Nn
2944       \g_@@_renderers_seq
2945       {
2946         \@@_define_renderer:n
2947         { ##1 }

```

```

2948     }
2949 }
2950 \cs_new:Nn \@@_define_renderer:n
2951 {
2952     \@@_renderer_tl_to_csname:nN
2953     { #1 }
2954     \l_tmpa_tl
2955     \prop_get:NnN
2956     \g_@@_renderer_arities_prop
2957     { #1 }
2958     \l_tmpb_tl
2959     \@@_define_renderer:ncV
2960     { #1 }
2961     { \l_tmpa_tl }
2962     \l_tmpb_tl
2963 }
2964 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2965 {
2966     \tl_set:Nn
2967     \l_tmpa_tl
2968     { \str_uppercase:n { #1 } }
2969     \tl_set:Nx
2970     #2
2971     {
2972         markdownRenderer
2973         \tl_head:f { \l_tmpa_tl }
2974         \tl_tail:n { #1 }
2975     }
2976 }
2977 \tl_new:N
2978 \l_@@_renderer_definition_tl
2979 \bool_new:N
2980 \g_@@_appending_renderer_bool
2981 \cs_new:Nn \@@_define_renderer:nNn
2982 {
2983     \keys_define:nn
2984     { markdown/options/renderers }
2985     {
2986         #1 .code:n = {
2987             \tl_set:Nn
2988             \l_@@_renderer_definition_tl
2989             { ##1 }
2990             \regex_replace_all:nnN
2991             { \cP\#0 }
2992             { #1 }
2993             \l_@@_renderer_definition_tl
2994             \bool_if:NT

```

```

2995         \g_@@_appending_renderer_bool
2996     {
2997         \@@_tl_set_from_cs:NNn
2998         \l_tmpa_tl
2999         #2
3000         { #3 }
3001         \tl_put_left:NV
3002         \l_@@_renderer_definition_tl
3003         \l_tmpa_tl
3004     }
3005     \cs_generate_from_arg_count:NNnV
3006     #2
3007     \cs_set:Npn
3008     { #3 }
3009     \l_@@_renderer_definition_tl
3010 },
3011 }

```

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.

```

3012     \str_if_eq:nnT
3013     { #1 }
3014     { jekyllDataString }
3015     {
3016         \cs_undefine:N
3017         #2
3018     }
3019 }

```

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.

```

3020 \cs_new_protected:Nn
3021 \@@_tl_set_from_cs:NNn
3022 {
3023     \tl_set:Nn
3024     \l_tmpa_tl
3025     { #2 }
3026     \int_step_inline:nn
3027     { #3 }
3028     {
3029         \exp_args:NNc
3030         \tl_put_right:Nn
3031         \l_tmpa_tl
3032         { @@_tl_set_from_cs_parameter_ ##1 }

```

```

3033     }
3034     \exp_args:NNV
3035     \tl_set:No
3036     \l_tmpb_tl
3037     \l_tmpa_tl
3038     \regex_replace_all:nnN
3039     { \cP. }
3040     { \0\0 }
3041     \l_tmpb_tl
3042     \int_step_inline:nn
3043     { #3 }
3044     {
3045         \regex_replace_all:nnN
3046         { \c { @@_tl_set_from_cs_parameter_ ##1 } }
3047         { \cP\# ##1 }
3048         \l_tmpb_tl
3049     }
3050     \tl_set:NV
3051     #1
3052     \l_tmpb_tl
3053 }
3054 \cs_generate_variant:Nn
3055 \@@_define_renderer:nNn
3056 { ncV }
3057 \cs_generate_variant:Nn
3058 \cs_generate_from_arg_count:NNnn
3059 { NNnV }
3060 \cs_generate_variant:Nn
3061 \tl_put_left:Nn
3062 { Nv }
3063 \keys_define:nn
3064 { markdown/options }
3065 {
3066     renderers .code:n = {
3067         \keys_set:nn
3068         { markdown/options/renderers }
3069         { #1 }
3070     },
3071 }

```

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.
  }
}

```

```

    }
}

```

```

3072 \tl_new:N
3073   \l_@@_renderer_glob_definition_tl
3074 \seq_new:N
3075   \l_@@_renderer_glob_results_seq
3076 \regex_const:Nn
3077   \c_@@_appending_key_regex
3078   { \s*+$ }
3079 \keys_define:nn
3080   { markdown/options/renderers }
3081   {
3082     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 += {...},
        },
      }
    },
  },
}

```

```

3083     \regex_match:NVTF
3084     \c_@@_appending_key_regex
3085     \l_keys_key_str
3086     {
3087         \bool_gset_true:N
3088         \g_@@_appending_renderer_bool
3089         \tl_set:NV
3090         \l_tmpa_tl
3091         \l_keys_key_str
3092         \regex_replace_once:NnN
3093         \c_@@_appending_key_regex
3094         { }
3095         \l_tmpa_tl
3096         \tl_set:Nx
3097         \l_tmpb_tl
3098         { { \l_tmpa_tl } = }
3099         \tl_put_right:Nn
3100         \l_tmpb_tl
3101         { { #1 } }
3102         \keys_set:nV
3103         { markdown/options/renderers }
3104         \l_tmpb_tl
3105         \bool_gset_false:N
3106         \g_@@_appending_renderer_bool
3107     }

```

In addition to exact token renderer names, we also support wildcards (*) and enumerations (l) 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 = {
    *lItem(|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.
  }
}
```

```
3108     {
3109         \@@_glob_seq:VnN
3110         \l_keys_key_str
3111         { g_@@_renderers_seq }
3112         \l_@@_renderer_glob_results_seq
3113     \seq_if_empty:NTF
3114         \l_@@_renderer_glob_results_seq
3115         {
3116             \msg_error:nnV
3117             { markdown }
3118             { undefined-renderer }
3119             \l_keys_key_str
3120         }
3121     {
3122         \tl_set:Nn
3123         \l_@@_renderer_glob_definition_tl
3124         { \exp_not:n { #1 } }
3125     \seq_map_inline:Nn
3126         \l_@@_renderer_glob_results_seq
3127         {
3128             \tl_set:Nn
3129             \l_tmpa_tl
3130             { { ##1 } = }
3131             \tl_put_right:Nx
3132             \l_tmpa_tl
3133             { { \l_@@_renderer_glob_definition_tl } }
3134             \keys_set:nV
3135             { markdown/options/renderers }
3136             \l_tmpa_tl
3137         }
3138     }
3139 },
3140 },
3141 }
3142 \msg_new:nnn
3143 { markdown }
3144 { undefined-renderer }
3145 {
```



```

3146     Renderer~#1~is~undefined.
3147 }
3148 \cs_generate_variant:Nn
3149   \@@_glob_seq:nnN
3150   { VnN }
3151 \cs_generate_variant:Nn
3152   \cs_generate_from_arg_count:NNnn
3153   { cNVV }
3154 \cs_generate_variant:Nn
3155   \msg_error:nnn
3156   { nnV }
3157 \prg_generate_conditional_variant:NNn
3158   \regex_match:Nn
3159   { NV }
3160   { TF }
3161 \prop_new:N
3162   \g_@@_glob_cache_prop
3163 \tl_new:N
3164   \l_@@_current_glob_tl
3165 \cs_new:Nn
3166   \@@_glob_seq:nnN
3167   {
3168     \tl_set:Nn
3169       \l_@@_current_glob_tl
3170       { ~ #1 $ }
3171     \prop_get:NeNTF
3172       \g_@@_glob_cache_prop
3173       { #2 / \l_@@_current_glob_tl }
3174     \l_tmpa_clist
3175     {
3176       \seq_set_from_clist:NN
3177         #3
3178         \l_tmpa_clist
3179     }
3180     {
3181       \seq_clear:N
3182         #3
3183       \regex_replace_all:nnN
3184         { \* }
3185         { .* }
3186         \l_@@_current_glob_tl
3187       \regex_set:NV
3188         \l_tmpa_regex
3189         \l_@@_current_glob_tl
3190       \seq_map_inline:cn
3191         { #2 }
3192       {

```

```

3193         \regex_match:NnT
3194         \l_tmpa_regex
3195         { ##1 }
3196         {
3197             \seq_put_right:Nn
3198             #3
3199             { ##1 }
3200         }
3201     }
3202     \clist_set_from_seq:NN
3203     \l_tmpa_clist
3204     #3
3205     \prop_gput:NeV
3206     \g_@@_glob_cache_prop
3207     { #2 / \l_@@_current_glob_tl }
3208     \l_tmpa_clist
3209 }
3210 }
3211 % TODO: Remove in TeX Live 2023.
3212 \prg_generate_conditional_variant:Nnn
3213     \prop_get:NnN
3214     { NeN }
3215     { TF }
3216 \cs_generate_variant:Nn
3217     \regex_set:Nn
3218     { NV }
3219 \cs_generate_variant:Nn
3220     \prop_gput:Nnn
3221     { 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.

```

3222 \str_if_eq:VVT
3223     \c_@@_top_layer_tl
3224     \c_@@_option_layer_plain_tex_tl
3225     {
3226         \@@_define_renderers:
3227     }
3228 \ExplSyntaxOff

```

2.2.6 Token Renderer Prototypes

2.2.6.1 YAML Metadata Renderer Prototypes

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

```
3229 \ExplSyntaxOn
3230 \keys_define:nn
3231   { markdown/jekyllData }
3232   { }
3233 \ExplSyntaxOff
```

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

```
3234 \ExplSyntaxOn
3235 \@@_with_various_cases:nn
3236   { jekyllDataRenderers }
3237   {
3238     \keys_define:nn
3239       { markdown/options }
3240       {
3241         #1 .code:n = {
3242           \tl_set:Nn
3243             \l_tmpa_tl
3244             { ##1 }
3245
```

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.

```
3245       \tl_replace_all:NnV
3246       \l_tmpa_tl
3247       { / }
3248       \c_backslash_str
3249       \keys_set:nV
3250       { markdown/options/jekyll-data-renderers }
3251       \l_tmpa_tl
3252     },
3253   }
3254 }
3255 \keys_define:nn
3256   { markdown/options/jekyll-data-renderers }
3257   {
3258     unknown .code:n = {
3259       \tl_set_eq:NN
3260         \l_tmpa_tl
3261         \l_keys_key_str
3262       \tl_replace_all:NVn
3263         \l_tmpa_tl
```

```

3264     \c_backslash_str
3265     { / }
3266     \tl_put_right:Nn
3267     \l_tmpa_tl
3268     {
3269         .code:n = { #1 }
3270     }
3271     \keys_define:nV
3272     { markdown/jekyllData }
3273     \l_tmpa_tl
3274 }
3275 }
3276 \cs_generate_variant:Nn
3277 \keys_define:nn
3278 { nV }
3279 \ExplSyntaxOff

```

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

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

```

3280 \ExplSyntaxOn
3281 \cs_new:Nn \@@_define_renderer_prototypes:
3282 {
3283     \seq_map_inline:Nn
3284     \g_@@_renderers_seq
3285     {
3286         \@@_define_renderer_prototype:n
3287         { ##1 }
3288     }
3289 }
3290 \cs_new:Nn \@@_define_renderer_prototype:n
3291 {
3292     \@@_renderer_prototype_tl_to_csname:nN
3293     { #1 }
3294     \l_tmpa_tl
3295     \prop_get:NnN
3296     \g_@@_renderer_arities_prop
3297     { #1 }
3298     \l_tmpb_tl
3299     \@@_define_renderer_prototype:ncV
3300     { #1 }
3301     { \l_tmpa_tl }

```

```

3302     \l_tmpb_tl
3303   }
3304 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3305   {
3306     \tl_set:Nn
3307       \l_tmpa_tl
3308       { \str_uppercase:n { #1 } }
3309     \tl_set:Nx
3310       #2
3311       {
3312         markdownRenderer
3313         \tl_head:f { \l_tmpa_tl }
3314         \tl_tail:n { #1 }
3315         Prototype
3316       }
3317   }
3318 \tl_new:N
3319   \l_@@_renderer_prototype_definition_tl
3320 \bool_new:N
3321   \g_@@_appending_renderer_prototype_bool
3322 \cs_new:Nn \@@_define_renderer_prototype:nNn
3323   {
3324     \keys_define:nn
3325       { markdown/options/renderer-prototypes }
3326       {
3327         #1 .code:n = {
3328           \tl_set:Nn
3329             \l_@@_renderer_prototype_definition_tl
3330             { ##1 }
3331           \regex_replace_all:nnN
3332             { \cP\#0 }
3333             { #1 }
3334           \l_@@_renderer_prototype_definition_tl
3335           \bool_if:NT
3336             \g_@@_appending_renderer_prototype_bool
3337             {
3338               \@@_tl_set_from_cs:NNn
3339               \l_tmpa_tl
3340               #2
3341               { #3 }
3342               \tl_put_left:NV
3343               \l_@@_renderer_prototype_definition_tl
3344               \l_tmpa_tl
3345             }
3346           \cs_generate_from_arg_count:NNnV
3347             #2
3348           \cs_set:Npn

```

```

3349         { #3 }
3350         \l_@@_renderer_prototype_definition_tl
3351     },
3352 }

```

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.

```

3353     \str_if_eq:nnF
3354     { #1 }
3355     { jekyllDataString }
3356     {
3357         \cs_if_free:NT
3358         #2
3359         {
3360             \cs_generate_from_arg_count:NNnn
3361             #2
3362             \cs_set:Npn
3363             { #3 }
3364             { }
3365         }
3366     }
3367 }
3368 \cs_generate_variant:Nn
3369 \@@_define_renderer_prototype:nNn
3370 { 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.
  }
}

```

```

3371 \keys_define:nn
3372 { markdown/options/renderer-prototypes }
3373 {
3374     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 += {...},
        },
      }
    },
  },
}

```

```

3375   \regex_match:NVTF
3376   \c_@@_appending_key_regex
3377   \l_keys_key_str
3378   {
3379     \bool_gset_true:N
3380     \g_@@_appending_renderer_prototype_bool
3381     \tl_set:NV
3382     \l_tmpa_tl
3383     \l_keys_key_str
3384     \regex_replace_once:NnN
3385     \c_@@_appending_key_regex
3386     { }
3387     \l_tmpa_tl
3388     \tl_set:Nx
3389     \l_tmpb_tl
3390     { { \l_tmpa_tl } = }
3391     \tl_put_right:Nn
3392     \l_tmpb_tl

```

```

3393         { { #1 } }
3394     \keys_set:nV
3395     { markdown/options/renderer-prototypes }
3396     \l_tmpb_tl
3397     \bool_gset_false:N
3398     \g_@@_appending_renderer_prototype_bool
3399 }

```

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.
  }
}

```

```

3400 {
3401   \@@_glob_seq:VnN
3402   \l_keys_key_str
3403   { g_@@_renderers_seq }
3404   \l_@@_renderer_glob_results_seq
3405   \seq_if_empty:NTF

```



```

3406         \l_@@_renderer_glob_results_seq
3407     {
3408         \msg_error:nnV
3409         { markdown }
3410         { undefined-renderer-prototype }
3411         \l_keys_key_str
3412     }
3413     {
3414         \tl_set:Nn
3415         \l_@@_renderer_glob_definition_tl
3416         { \exp_not:n { #1 } }
3417         \seq_map_inline:Nn
3418         \l_@@_renderer_glob_results_seq
3419         {
3420             \tl_set:Nn
3421             \l_tmpa_tl
3422             { { ##1 } = }
3423             \tl_put_right:Nx
3424             \l_tmpa_tl
3425             { { \l_@@_renderer_glob_definition_tl } }
3426             \keys_set:nV
3427             { markdown/options/renderer-prototypes }
3428             \l_tmpa_tl
3429         }
3430     }
3431 },
3432 },
3433 }
3434 \msg_new:nnn
3435 { markdown }
3436 { undefined-renderer-prototype }
3437 {
3438     Renderer~prototype~#1~is~undefined.
3439 }
3440 \@@_with_various_cases:nn
3441 { rendererPrototypes }
3442 {
3443     \keys_define:nn
3444     { markdown/options }
3445     {
3446         #1 .code:n = {
3447             \keys_set:nn
3448             { markdown/options/renderer-prototypes }
3449             { ##1 }
3450         },
3451     }
3452 }

```

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

```

3453 \str_if_eq:VVT
3454   \c_@@_top_layer_tl
3455   \c_@@_option_layer_plain_tex_tl
3456   {
3457     \@@_define_renderer_prototypes:
3458   }
3459 \ExplSyntaxOff

```

2.2.7 Logging Facilities

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

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

2.2.8 Miscellanea

The `\markdownMakeOther` macro is used by the package, when a T_EX engine that does not support direct Lua access is starting to buffer a text. The plain T_EX implementation changes the category code of plain T_EX 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*.

```

3460 \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 T_EX 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.

```

3461 \let\markdownReadAndConvert\relax
3462 \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*.

```

3463   \catcode`\|=0\catcode`\=12%

```

```

3464 |gdef|markdownBegin{%
3465   |markdownReadAndConvert{\markdownEnd}%
3466                               {\markdownEnd}}%
3467 |gdef|yamlBegin{%
3468   |begingroup
3469   |yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
3470   |markdownReadAndConvert{\yamlEnd}%
3471                               {\yamlEnd}}%
3472 |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.

```

3473 \ExplSyntaxOn
3474 \keys_define:nn
3475   { markdown/options }
3476   {
3477     code .code:n = { #1 },
3478   }
3479 \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.

```

3480 \ExplSyntaxOn
3481 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3482 \cs_generate_variant:Nn
3483   \tl_const:Nn
3484   { NV }
3485 \tl_if_exist:NF
3486   \c_@@_top_layer_tl
3487   {
3488     \tl_const:NV
3489       \c_@@_top_layer_tl
3490       \c_@@_option_layer_latex_tl
3491   }
3492 \ExplSyntaxOff

```

3493 `\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.

3494 `\newenvironment{markdown}\relax\relax`

3495 `\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:

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

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.

```
3496 \newenvironment{yaml}\relax\relax
```

Furthermore, the environment accepts L^AT_EX interface options (see Section 2.3.3) as the only optional argument.

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

The following example L^AT_EX code showcases the usage of the `yaml` environment:

```

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

```

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

```

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

```

You can't directly extend the `yaml` L^AT_EX environment by using it in other environments. However, you can work around this limitation by using the `\yaml` and `\yamlEnd` macros directly in the definition, similarly to the `\markdown` and `\markdownEnd` macros described previously. Unlike with the `\markdown` and `\markdownEnd` macros, The `\yamlEnd` macro `__must__` be used in or after the replacement after-text.

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

2.3.1.2 Typesetting Markdown and YAML from external documents

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of

the input markdown document to plain T_EX. Unlike the `\markdownInput` macro provided by the plain T_EX interface, this macro also accepts L^AT_EX interface options (see Section 2.3.3) as its optional argument. These options will only influence this markdown document.

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

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

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

The following example L^AT_EX 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 “[markdown](#)foo 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 “[markdown](#)foo `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.

```
3497 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3498 \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.

```

3499 \ExplSyntaxOn
3500 \str_if_eq:VVT
3501   \c_@@_top_layer_tl
3502   \c_@@_option_layer_latex_tl
3503   {
3504     \@@_define_option_commands_and_keyvals:
3505     \@@_define_renderers:
3506     \@@_define_renderer_prototypes:
3507   }
3508 \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}
```

```
3509 \newif\ifmarkdownLaTeXLoaded
3510 \markdownLaTeXLoadedfalse
```

Due to limitations of L^AT_EX, themes may not be loaded after the beginning of a L^AT_EX document.

Built-in L^AT_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";

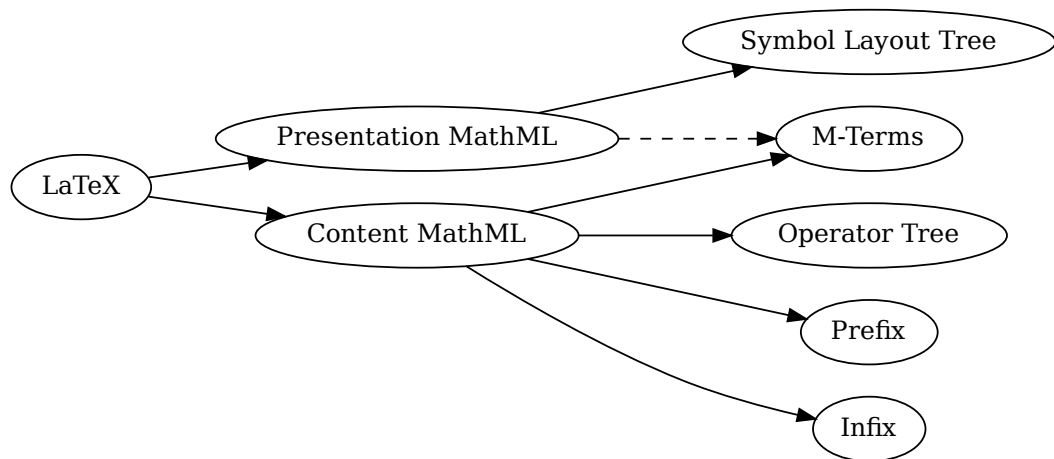
 latex -> pmml;
 latex -> cmml;
 pmml -> slt;
 cmml -> opt;
 cmml -> prefix;
 cmml -> infix;
 pmml -> mterms [style=dashed];
 cmml -> mterms;
```

```

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  $\text{\TeX}$  option is enabled.

3511 \ProvidesPackage{markdownthemewitiko\_dot}[2021/03/09]%

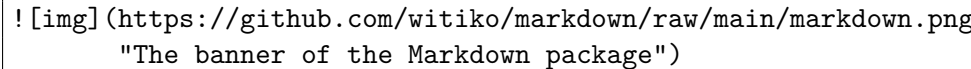
**witiko/graphicx/http** A theme that adds support for downloading images whose URL has the http or https protocol.

```

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

```

```

\begin{document}
\begin{markdown}

\end{markdown}
\end{document}

```

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

```

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

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

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

```



## Chapter 1

# Introduction

## 1.1 Section

### 1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

**Figure 5: The banner of the Markdown package**

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

```
3512 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%
```

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

```

3513 \AtEndOfPackage{
3514 \markdownLaTeXLoadedtrue
3515 }

```

At the end of the  $\text{\LaTeX}$  module, we load the `witiko/markdown/defaults`  $\text{\LaTeX}$  theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

3516 \ExplSyntaxOn
3517 \str_if_eq:VVT
3518 \c_@@_top_layer_tl
3519 \c_@@_option_layer_latex_tl
3520 {
3521 \ExplSyntaxOff
3522 \AtEndOfPackage
3523 {
3524 \@@_if_option:nF
3525 { noDefaults }
3526 {
3527 \@@_if_option:nTF
3528 { experimental }
3529 {
3530 \@@_setup:n
3531 { theme = witiko/markdown/defaults@experimental }
3532 }
3533 {
3534 \@@_setup:n
3535 { theme = witiko/markdown/defaults }
3536 }
3537 }
3538 }
3539 \ExplSyntaxOn
3540 }
3541 \ExplSyntaxOff
3542 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/10/29]%

```

Please, see Section 3.3.2 for implementation details of the built-in  $\text{\LaTeX}$  themes.

## 2.4 ConT $\text{\E}$ Xt Interface

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

```

3543 \ExplSyntaxOn
3544 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3545 \cs_generate_variant:Nn
3546 \tl_const:Nn
3547 { NV }
3548 \tl_if_exist:NF
3549 \c_@@_top_layer_tl
3550 {

```

```

3551 \tl_const:NV
3552 \c_@@_top_layer_tl
3553 \c_@@_option_layer_context_tl
3554 }
3555 \ExplSyntaxOff

```

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

```

3556 \writestatus{loading}{ConTEXt User Module / markdown}%
3557 \startmodule[markdown]
3558 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3559 \do\#\do\^\do_do\%do\~}%
3560 \input markdown/markdown

```

The ConT<sub>E</sub>Xt interface is implemented by the `t-markdown.tex` ConT<sub>E</sub>Xt module file that can be loaded as follows:

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

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

```

3561 \let\startmarkdown\relax
3562 \let\stopmarkdown\relax

```

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

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

The following example ConT<sub>E</sub>Xt code showcases the usage of the `\startmarkdown` and `\stopmarkdown` macros:

```

\usemodule[t][markdown]
\starttext
\startmarkdown

```

```

Hello **world** ...
\stopmarkdown
\stoptext

```

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

```

3563 \let\startyaml\relax
3564 \let\stopyaml\relax

```

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

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

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

```

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

```

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

```

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

```

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

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

```

3565 \let\inputmarkdown\relax

```



Furthermore, the `\inputmarkdown` macro also accepts ConT<sub>E</sub>Xt interface options (see Section 2.4.2) as its optional argument. These options will only influence this markdown document.

The following example ConT<sub>E</sub>Xt code showcases the usage of the `\inputmarkdown` macro:

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

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

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

The `\inputyaml` macro aliases the macro `\yamlInput` exposed by the plain T<sub>E</sub>X interface.

```
3566 \let\inputyaml\relax
```

Furthermore, the `\inputyaml` macro also accepts ConT<sub>E</sub>Xt interface options (see Section 2.4.2) as its optional argument. These options will only influence this YAML document.

The following example ConT<sub>E</sub>Xt code showcases the usage of the `\inputyaml` macro:

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

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

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

## 2.4.2 Options

The ConT<sub>E</sub>Xt 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.

ConT<sub>E</sub>Xt options map directly to the options recognized by the plain T<sub>E</sub>X interface (see Section 2.2.2).

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

```
3567 \ExplSyntaxOn
3568 \cs_new:Npn
3569 \setupmarkdown
3570 [#1]
3571 {
3572 \@@_setup:n
3573 { #1 }
3574 }
```

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

```
3575 \cs_gset_eq:NN
3576 \setupyaml
3577 \setupmarkdown
```

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

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

```
3578 \cs_new:Nn \@@_caseless:N
3579 {
3580 \regex_replace_all:nnN
3581 { ([a-z])([A-Z]) }
3582 { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
3583 #1
3584 \tl_set:Nx
3585 #1
3586 { #1 }
3587 }
3588 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
```

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

```

3589 \str_if_eq:VVT
3590 \c_@@_top_layer_tl
3591 \c_@@_option_layer_context_tl
3592 {
3593 \@@_define_option_commands_and_keyvals:
3594 \@@_define_renderers:
3595 \@@_define_renderer_prototypes:
3596 }
3597 \ExplSyntaxOff

```

### 2.4.3 Themes

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

Having the Markdown package automatically load either the generic `.tex theme file` or the ConT<sub>E</sub>Xt-specific `t-*.tex` theme file allows developers to have a single *theme file*, when the theme is small or the difference between T<sub>E</sub>X formats is unimportant, and scale up to separate theme files native to different T<sub>E</sub>X formats for large multi-format themes, where different code is needed for different T<sub>E</sub>X formats. To enable code reuse, developers can load the `.tex` theme file from the `t-*.tex` theme file using the `\markdownLoadPlainTeXTheme` macro.

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

```

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

```

Built-in ConT<sub>E</sub>Xt themes provided with the Markdown package include:

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

```

3598 \startmodule[markdownthemewitiko_markdown_defaults]
3599 \unprotect

```

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

### 3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain T<sub>E</sub>X, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module and the remaining markdown reader and plain T<sub>E</sub>X writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3600 local upper, format, length =
3601 string.upper, string.format, string.len
3602 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3603 lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3604 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<sub>E</sub>X 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.

```
3605 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.

```
3606 function util.err(msg, exit_code)
3607 io.stderr:write("markdown.lua: " .. msg .. "\n")
3608 os.exit(exit_code or 1)
3609 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.

```
3610 function util.cache(dir, string, salt, transform, suffix)
3611 local digest = md5.sumhexa(string .. (salt or ""))
3612 local name = util.pathname(dir, digest .. suffix)
3613 local file = io.open(name, "r")
3614 if file == nil then -- If no cache entry exists, create a new one.
3615 file = assert(io.open(name, "w"),
```

```

3616 [[Could not open file "]] .. name .. [[" for writing]])
3617 local result = string
3618 if transform ~= nil then
3619 result = transform(result)
3620 end
3621 assert(file:write(result))
3622 assert(file:close())
3623 end
3624 return name
3625 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.

```

3626 function util.cache_verbatim(dir, string)
3627 local name = util.cache(dir, string, nil, nil, ".verbatim")
3628 return name
3629 end

```

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

```

3630 function util.table_copy(t)
3631 local u = { }
3632 for k, v in pairs(t) do u[k] = v end
3633 return setmetatable(u, getmetatable(t))
3634 end

```

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

```

3635 function util.encode_json_string(s)
3636 s = s:gsub([[\\]], [[\\]])
3637 s = s:gsub([["]], [[\"]])
3638 return [["]] .. s .. [["]]
3639 end

```

The `util.expand_tabs_in_line` expands tabs in string `s`. If `tabstop` is specified, it is used as the tab stop width. Otherwise, the tab stop width of 4 characters is used. The method is a copy of the tab expansion algorithm from Ierusalimschy [14, Chapter 21].

```

3640 function util.expand_tabs_in_line(s, tabstop)
3641 local tab = tabstop or 4
3642 local corr = 0
3643 return (s:gsub("(\\t)", function(p)
3644 local sp = tab - (p - 1 + corr) % tab
3645 corr = corr - 1 + sp
3646 return string.rep(" ", sp)
3647 end))
3648 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.

```
3649 function util.walk(t, f)
3650 local typ = type(t)
3651 if typ == "string" then
3652 f(t)
3653 elseif typ == "table" then
3654 local i = 1
3655 local n
3656 n = t[i]
3657 while n do
3658 util.walk(n, f)
3659 i = i + 1
3660 n = t[i]
3661 end
3662 elseif typ == "function" then
3663 local ok, val = pcall(t)
3664 if ok then
3665 util.walk(val, f)
3666 end
3667 else
3668 f(tostring(t))
3669 end
3670 end
```

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

```
3671 function util.flatten(ary)
3672 local new = {}
3673 for _,v in ipairs(ary) do
3674 if type(v) == "table" then
3675 for _,w in ipairs(util.flatten(v)) do
3676 new[#new + 1] = w
3677 end
3678 else
3679 new[#new + 1] = v
3680 end
3681 end
3682 return new
3683 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.

```
3684 function util.rope_to_string(rope)
3685 local buffer = {}
3686 util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3687 return table.concat(buffer)
```

```
3688 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.

```
3689 function util.rope_last(rope)
3690 if #rope == 0 then
3691 return nil
3692 else
3693 local l = rope[#rope]
3694 if type(l) == "table" then
3695 return util.rope_last(l)
3696 else
3697 return l
3698 end
3699 end
3700 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  $1 \leq i \leq \#ary$ .

```
3701 function util.intersperse(ary, x)
3702 local new = {}
3703 local l = #ary
3704 for i,v in ipairs(ary) do
3705 local n = #new
3706 new[n + 1] = v
3707 if i ~= l then
3708 new[n + 2] = x
3709 end
3710 end
3711 return new
3712 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  $1 \leq i \leq \#ary$ .

```
3713 function util.map(ary, f)
3714 local new = {}
3715 for i,v in ipairs(ary) do
3716 new[i] = f(v)
3717 end
3718 return new
3719 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.

```
3720 function util.escaper(char_escapes, string_escapes)
```

Build a string of escapable characters.

```
3721 local char_escapes_list = ""
3722 for i,_ in pairs(char_escapes) do
3723 char_escapes_list = char_escapes_list .. i
3724 end
```

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

```
3725 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.

```
3726 if string_escapes then
3727 for k,v in pairs(string_escapes) do
3728 escapable = P(k) / v + escapable
3729 end
3730 end
```

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

```
3731 local escape_string = Cs((escapable + any)^0)
```

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

```
3732 return function(s)
3733 return lpeg.match(escape_string, s)
3734 end
3735 end
```

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

```
3736 function util.pathname(dir, file)
3737 if #dir == 0 then
3738 return file
3739 else
3740 return dir .. "/" .. file
3741 end
3742 end
```



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

```
3743 function util.salt(options)
3744 local opt_string = {}
3745 for k, _ in pairs(defaultOptions) do
3746 local v = options[k]
3747 if type(v) == "table" then
3748 for _, i in ipairs(v) do
3749 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
3750 end
3751 end
3752 end
3753 end
```

The `cacheDir` option is disregarded.

```
3751 elseif k ~= "cacheDir" then
3752 opt_string[#opt_string+1] = k .. "=" .. tostring(v)
3753 end
3754 end
3755 table.sort(opt_string)
3756 local salt = table.concat(opt_string, ",")
3757 .. "," .. metadata.version
3758 return salt
3759 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.

```
3760 function util.warning(s)
3761 io.stderr:write("Warning: " .. s .. "\n")
3762 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.

```
3763 local entities = {}
3764
3765 local character_entities = {
3766 ["Tab"] = 9,
3767 ["NewLine"] = 10,
3768 ["excl"] = 33,
3769 ["QUOT"] = 34,
3770 ["quot"] = 34,
3771 ["num"] = 35,
3772 ["dollar"] = 36,
3773 ["percent"] = 37,
3774 ["AMP"] = 38,
3775 ["amp"] = 38,
```

```

3776 ["apos"] = 39,
3777 ["lpar"] = 40,
3778 ["rpar"] = 41,
3779 ["ast"] = 42,
3780 ["midast"] = 42,
3781 ["plus"] = 43,
3782 ["comma"] = 44,
3783 ["period"] = 46,
3784 ["sol"] = 47,
3785 ["colon"] = 58,
3786 ["semi"] = 59,
3787 ["LT"] = 60,
3788 ["lt"] = 60,
3789 ["nvlt"] = {60, 8402},
3790 ["bne"] = {61, 8421},
3791 ["equals"] = 61,
3792 ["GT"] = 62,
3793 ["gt"] = 62,
3794 ["nvgt"] = {62, 8402},
3795 ["quest"] = 63,
3796 ["commat"] = 64,
3797 ["lbrack"] = 91,
3798 ["lsqb"] = 91,
3799 ["bsol"] = 92,
3800 ["rbrack"] = 93,
3801 ["rsqb"] = 93,
3802 ["Hat"] = 94,
3803 ["UnderBar"] = 95,
3804 ["lowbar"] = 95,
3805 ["DiacriticalGrave"] = 96,
3806 ["grave"] = 96,
3807 ["fjlig"] = {102, 106},
3808 ["lbrace"] = 123,
3809 ["lcub"] = 123,
3810 ["VerticalLine"] = 124,
3811 ["verbar"] = 124,
3812 ["vert"] = 124,
3813 ["rbrace"] = 125,
3814 ["rcub"] = 125,
3815 ["NonBreakingSpace"] = 160,
3816 ["nbsp"] = 160,
3817 ["iexcl"] = 161,
3818 ["cent"] = 162,
3819 ["pound"] = 163,
3820 ["curren"] = 164,
3821 ["yen"] = 165,
3822 ["brvbar"] = 166,

```

```

3823 ["sect"] = 167,
3824 ["Dot"] = 168,
3825 ["DoubleDot"] = 168,
3826 ["die"] = 168,
3827 ["uml"] = 168,
3828 ["COPY"] = 169,
3829 ["copy"] = 169,
3830 ["ordf"] = 170,
3831 ["laquo"] = 171,
3832 ["not"] = 172,
3833 ["shy"] = 173,
3834 ["REG"] = 174,
3835 ["circledR"] = 174,
3836 ["reg"] = 174,
3837 ["macr"] = 175,
3838 ["strns"] = 175,
3839 ["deg"] = 176,
3840 ["PlusMinus"] = 177,
3841 ["plusmn"] = 177,
3842 ["pm"] = 177,
3843 ["sup2"] = 178,
3844 ["sup3"] = 179,
3845 ["DiacriticalAcute"] = 180,
3846 ["acute"] = 180,
3847 ["micro"] = 181,
3848 ["para"] = 182,
3849 ["CenterDot"] = 183,
3850 ["centerdot"] = 183,
3851 ["middot"] = 183,
3852 ["Cedilla"] = 184,
3853 ["cedil"] = 184,
3854 ["sup1"] = 185,
3855 ["ordm"] = 186,
3856 ["raquo"] = 187,
3857 ["frac14"] = 188,
3858 ["frac12"] = 189,
3859 ["half"] = 189,
3860 ["frac34"] = 190,
3861 ["iquest"] = 191,
3862 ["Agrave"] = 192,
3863 ["Aacute"] = 193,
3864 ["Acirc"] = 194,
3865 ["Atilde"] = 195,
3866 ["Auml"] = 196,
3867 ["Aring"] = 197,
3868 ["angst"] = 197,
3869 ["AElig"] = 198,

```

3870 ["Ccedil"] = 199,  
 3871 ["Egrave"] = 200,  
 3872 ["Eacute"] = 201,  
 3873 ["Ecirc"] = 202,  
 3874 ["Euml"] = 203,  
 3875 ["Igrave"] = 204,  
 3876 ["Iacute"] = 205,  
 3877 ["Icirc"] = 206,  
 3878 ["Iuml"] = 207,  
 3879 ["ETH"] = 208,  
 3880 ["Ntilde"] = 209,  
 3881 ["Ograve"] = 210,  
 3882 ["Oacute"] = 211,  
 3883 ["Ocirc"] = 212,  
 3884 ["Otilde"] = 213,  
 3885 ["Ouml"] = 214,  
 3886 ["times"] = 215,  
 3887 ["Oslash"] = 216,  
 3888 ["Ugrave"] = 217,  
 3889 ["Uacute"] = 218,  
 3890 ["Ucirc"] = 219,  
 3891 ["Uuml"] = 220,  
 3892 ["Yacute"] = 221,  
 3893 ["THORN"] = 222,  
 3894 ["szlig"] = 223,  
 3895 ["agrave"] = 224,  
 3896 ["aacute"] = 225,  
 3897 ["acirc"] = 226,  
 3898 ["atilde"] = 227,  
 3899 ["auml"] = 228,  
 3900 ["aring"] = 229,  
 3901 ["aelig"] = 230,  
 3902 ["ccedil"] = 231,  
 3903 ["egrave"] = 232,  
 3904 ["eacute"] = 233,  
 3905 ["ecirc"] = 234,  
 3906 ["euml"] = 235,  
 3907 ["igrave"] = 236,  
 3908 ["iacute"] = 237,  
 3909 ["icirc"] = 238,  
 3910 ["iuml"] = 239,  
 3911 ["eth"] = 240,  
 3912 ["ntilde"] = 241,  
 3913 ["ograve"] = 242,  
 3914 ["oacute"] = 243,  
 3915 ["ocirc"] = 244,  
 3916 ["otilde"] = 245,

```

3917 ["ouml"] = 246,
3918 ["div"] = 247,
3919 ["divide"] = 247,
3920 ["oslash"] = 248,
3921 ["ugrave"] = 249,
3922 ["uacute"] = 250,
3923 ["ucirc"] = 251,
3924 ["uuml"] = 252,
3925 ["yacute"] = 253,
3926 ["thorn"] = 254,
3927 ["yuml"] = 255,
3928 ["Amacr"] = 256,
3929 ["amacr"] = 257,
3930 ["Abreve"] = 258,
3931 ["abreve"] = 259,
3932 ["Aogon"] = 260,
3933 ["aogon"] = 261,
3934 ["Cacute"] = 262,
3935 ["cacute"] = 263,
3936 ["Ccirc"] = 264,
3937 ["ccirc"] = 265,
3938 ["Cdot"] = 266,
3939 ["cdot"] = 267,
3940 ["Ccaron"] = 268,
3941 ["ccaron"] = 269,
3942 ["Dcaron"] = 270,
3943 ["dcaron"] = 271,
3944 ["Dstrok"] = 272,
3945 ["dstrok"] = 273,
3946 ["Emacr"] = 274,
3947 ["emacr"] = 275,
3948 ["Edot"] = 278,
3949 ["edot"] = 279,
3950 ["Eogon"] = 280,
3951 ["eogon"] = 281,
3952 ["Ecaron"] = 282,
3953 ["ecaron"] = 283,
3954 ["Gcirc"] = 284,
3955 ["gcirc"] = 285,
3956 ["Gbreve"] = 286,
3957 ["gbreve"] = 287,
3958 ["Gdot"] = 288,
3959 ["gdot"] = 289,
3960 ["Gcedil"] = 290,
3961 ["Hcirc"] = 292,
3962 ["hcirc"] = 293,
3963 ["Hstrok"] = 294,

```

```

3964 ["hstrok"] = 295,
3965 ["Itilde"] = 296,
3966 ["itilde"] = 297,
3967 ["Imacr"] = 298,
3968 ["imacr"] = 299,
3969 ["Iogon"] = 302,
3970 ["iogon"] = 303,
3971 ["Idot"] = 304,
3972 ["imath"] = 305,
3973 ["inodot"] = 305,
3974 ["IJlig"] = 306,
3975 ["ijlig"] = 307,
3976 ["Jcirc"] = 308,
3977 ["jcirc"] = 309,
3978 ["Kcedil"] = 310,
3979 ["kcedil"] = 311,
3980 ["kgreen"] = 312,
3981 ["Lacute"] = 313,
3982 ["lacute"] = 314,
3983 ["Lcedil"] = 315,
3984 ["lcedil"] = 316,
3985 ["Lcaron"] = 317,
3986 ["lcaron"] = 318,
3987 ["Lmidot"] = 319,
3988 ["lmidot"] = 320,
3989 ["Lstrok"] = 321,
3990 ["lstrok"] = 322,
3991 ["Nacute"] = 323,
3992 ["nacute"] = 324,
3993 ["Ncedil"] = 325,
3994 ["ncedil"] = 326,
3995 ["Ncaron"] = 327,
3996 ["ncaron"] = 328,
3997 ["napos"] = 329,
3998 ["ENG"] = 330,
3999 ["eng"] = 331,
4000 ["Omacr"] = 332,
4001 ["omacr"] = 333,
4002 ["Odblac"] = 336,
4003 ["odblac"] = 337,
4004 ["OElig"] = 338,
4005 ["oelig"] = 339,
4006 ["Racute"] = 340,
4007 ["racute"] = 341,
4008 ["Rcedil"] = 342,
4009 ["rcedil"] = 343,
4010 ["Rcaron"] = 344,

```

```

4011 ["rcaron"] = 345,
4012 ["Sacute"] = 346,
4013 ["sacute"] = 347,
4014 ["Scirc"] = 348,
4015 ["scirc"] = 349,
4016 ["Scedil"] = 350,
4017 ["scedil"] = 351,
4018 ["Scaron"] = 352,
4019 ["scaron"] = 353,
4020 ["Tcedil"] = 354,
4021 ["tcedil"] = 355,
4022 ["Tcaron"] = 356,
4023 ["tcaron"] = 357,
4024 ["Tstrok"] = 358,
4025 ["tstrok"] = 359,
4026 ["Utilde"] = 360,
4027 ["utilde"] = 361,
4028 ["Umacr"] = 362,
4029 ["umacr"] = 363,
4030 ["Ubreve"] = 364,
4031 ["ubreve"] = 365,
4032 ["Uring"] = 366,
4033 ["uring"] = 367,
4034 ["Udblac"] = 368,
4035 ["udblac"] = 369,
4036 ["Uogon"] = 370,
4037 ["uogon"] = 371,
4038 ["Wcirc"] = 372,
4039 ["wcirc"] = 373,
4040 ["Ycirc"] = 374,
4041 ["ycirc"] = 375,
4042 ["Yuml"] = 376,
4043 ["Zacute"] = 377,
4044 ["zacute"] = 378,
4045 ["Zdot"] = 379,
4046 ["zdot"] = 380,
4047 ["Zcaron"] = 381,
4048 ["zcaron"] = 382,
4049 ["fnof"] = 402,
4050 ["imped"] = 437,
4051 ["gacute"] = 501,
4052 ["jmath"] = 567,
4053 ["circ"] = 710,
4054 ["Hacek"] = 711,
4055 ["caron"] = 711,
4056 ["Breve"] = 728,
4057 ["breve"] = 728,

```

```

4058 ["DiacriticalDot"] = 729,
4059 ["dot"] = 729,
4060 ["ring"] = 730,
4061 ["ogon"] = 731,
4062 ["DiacriticalTilde"] = 732,
4063 ["tilde"] = 732,
4064 ["DiacriticalDoubleAcute"] = 733,
4065 ["dblac"] = 733,
4066 ["DownBreve"] = 785,
4067 ["Alpha"] = 913,
4068 ["Beta"] = 914,
4069 ["Gamma"] = 915,
4070 ["Delta"] = 916,
4071 ["Epsilon"] = 917,
4072 ["Zeta"] = 918,
4073 ["Eta"] = 919,
4074 ["Theta"] = 920,
4075 ["Iota"] = 921,
4076 ["Kappa"] = 922,
4077 ["Lambda"] = 923,
4078 ["Mu"] = 924,
4079 ["Nu"] = 925,
4080 ["Xi"] = 926,
4081 ["Omicron"] = 927,
4082 ["Pi"] = 928,
4083 ["Rho"] = 929,
4084 ["Sigma"] = 931,
4085 ["Tau"] = 932,
4086 ["Upsilon"] = 933,
4087 ["Phi"] = 934,
4088 ["Chi"] = 935,
4089 ["Psi"] = 936,
4090 ["Omega"] = 937,
4091 ["ohm"] = 937,
4092 ["alpha"] = 945,
4093 ["beta"] = 946,
4094 ["gamma"] = 947,
4095 ["delta"] = 948,
4096 ["epsi"] = 949,
4097 ["epsilon"] = 949,
4098 ["zeta"] = 950,
4099 ["eta"] = 951,
4100 ["theta"] = 952,
4101 ["iota"] = 953,
4102 ["kappa"] = 954,
4103 ["lambda"] = 955,
4104 ["mu"] = 956,

```



```

4105 ["nu"] = 957,
4106 ["xi"] = 958,
4107 ["omicron"] = 959,
4108 ["pi"] = 960,
4109 ["rho"] = 961,
4110 ["sigmaf"] = 962,
4111 ["sigmav"] = 962,
4112 ["varsigma"] = 962,
4113 ["sigma"] = 963,
4114 ["tau"] = 964,
4115 ["upsi"] = 965,
4116 ["upsilon"] = 965,
4117 ["phi"] = 966,
4118 ["chi"] = 967,
4119 ["psi"] = 968,
4120 ["omega"] = 969,
4121 ["thetasym"] = 977,
4122 ["thetav"] = 977,
4123 ["vartheta"] = 977,
4124 ["Upsi"] = 978,
4125 ["upsih"] = 978,
4126 ["phiv"] = 981,
4127 ["straightphi"] = 981,
4128 ["varphi"] = 981,
4129 ["piv"] = 982,
4130 ["varpi"] = 982,
4131 ["Gammad"] = 988,
4132 ["digamma"] = 989,
4133 ["gammad"] = 989,
4134 ["kappav"] = 1008,
4135 ["varkappa"] = 1008,
4136 ["rhov"] = 1009,
4137 ["varrho"] = 1009,
4138 ["epsiv"] = 1013,
4139 ["straightepsilon"] = 1013,
4140 ["varepsilon"] = 1013,
4141 ["backepsilon"] = 1014,
4142 ["bepsi"] = 1014,
4143 ["IOcy"] = 1025,
4144 ["DJcy"] = 1026,
4145 ["GJcy"] = 1027,
4146 ["Jukcy"] = 1028,
4147 ["DScy"] = 1029,
4148 ["Iukcy"] = 1030,
4149 ["YIcy"] = 1031,
4150 ["Jsercy"] = 1032,
4151 ["LJcy"] = 1033,

```

```

4152 ["NJcy"] = 1034,
4153 ["TSHcy"] = 1035,
4154 ["KJcy"] = 1036,
4155 ["Ubrcy"] = 1038,
4156 ["DZcy"] = 1039,
4157 ["Acy"] = 1040,
4158 ["Bcy"] = 1041,
4159 ["Vcy"] = 1042,
4160 ["Gcy"] = 1043,
4161 ["Dcy"] = 1044,
4162 ["IEcy"] = 1045,
4163 ["ZHcy"] = 1046,
4164 ["Zcy"] = 1047,
4165 ["Icy"] = 1048,
4166 ["Jcy"] = 1049,
4167 ["Kcy"] = 1050,
4168 ["Lcy"] = 1051,
4169 ["Mcy"] = 1052,
4170 ["Ncy"] = 1053,
4171 ["Ocy"] = 1054,
4172 ["Pcy"] = 1055,
4173 ["Rcy"] = 1056,
4174 ["Scy"] = 1057,
4175 ["Tcy"] = 1058,
4176 ["Ucy"] = 1059,
4177 ["Fcy"] = 1060,
4178 ["KHcy"] = 1061,
4179 ["TScy"] = 1062,
4180 ["CHcy"] = 1063,
4181 ["SHcy"] = 1064,
4182 ["SHCHcy"] = 1065,
4183 ["HARDcy"] = 1066,
4184 ["Ycy"] = 1067,
4185 ["SOFTcy"] = 1068,
4186 ["Ecy"] = 1069,
4187 ["YUcy"] = 1070,
4188 ["YAcy"] = 1071,
4189 ["acy"] = 1072,
4190 ["bcy"] = 1073,
4191 ["vcy"] = 1074,
4192 ["gcy"] = 1075,
4193 ["dcy"] = 1076,
4194 ["iecy"] = 1077,
4195 ["zhcy"] = 1078,
4196 ["zcy"] = 1079,
4197 ["icy"] = 1080,
4198 ["jcy"] = 1081,

```

```

4199 ["kcy"] = 1082,
4200 ["lcy"] = 1083,
4201 ["mcy"] = 1084,
4202 ["ncy"] = 1085,
4203 ["ocy"] = 1086,
4204 ["pcy"] = 1087,
4205 ["rcy"] = 1088,
4206 ["scy"] = 1089,
4207 ["tcy"] = 1090,
4208 ["ucy"] = 1091,
4209 ["fcy"] = 1092,
4210 ["khcy"] = 1093,
4211 ["tscy"] = 1094,
4212 ["chcy"] = 1095,
4213 ["shcy"] = 1096,
4214 ["shchcy"] = 1097,
4215 ["hardcy"] = 1098,
4216 ["ycy"] = 1099,
4217 ["softcy"] = 1100,
4218 ["ecy"] = 1101,
4219 ["yucy"] = 1102,
4220 ["yacy"] = 1103,
4221 ["iocy"] = 1105,
4222 ["djcy"] = 1106,
4223 ["gjcy"] = 1107,
4224 ["jukcy"] = 1108,
4225 ["dscy"] = 1109,
4226 ["iukcy"] = 1110,
4227 ["yicy"] = 1111,
4228 ["jsercy"] = 1112,
4229 ["ljcy"] = 1113,
4230 ["njcy"] = 1114,
4231 ["tshcy"] = 1115,
4232 ["kjcy"] = 1116,
4233 ["ubrcy"] = 1118,
4234 ["dzcyc"] = 1119,
4235 ["ensp"] = 8194,
4236 ["emsp"] = 8195,
4237 ["emsp13"] = 8196,
4238 ["emsp14"] = 8197,
4239 ["numsp"] = 8199,
4240 ["puncsp"] = 8200,
4241 ["ThinSpace"] = 8201,
4242 ["thinsp"] = 8201,
4243 ["VeryThinSpace"] = 8202,
4244 ["hairsp"] = 8202,
4245 ["NegativeMediumSpace"] = 8203,

```

4246 ["NegativeThickSpace"] = 8203,  
 4247 ["NegativeThinSpace"] = 8203,  
 4248 ["NegativeVeryThinSpace"] = 8203,  
 4249 ["ZeroWidthSpace"] = 8203,  
 4250 ["zwnj"] = 8204,  
 4251 ["zwj"] = 8205,  
 4252 ["lrm"] = 8206,  
 4253 ["rlm"] = 8207,  
 4254 ["dash"] = 8208,  
 4255 ["hyphen"] = 8208,  
 4256 ["ndash"] = 8211,  
 4257 ["mdash"] = 8212,  
 4258 ["horbar"] = 8213,  
 4259 ["Verbar"] = 8214,  
 4260 ["Vert"] = 8214,  
 4261 ["OpenCurlyQuote"] = 8216,  
 4262 ["lsquo"] = 8216,  
 4263 ["CloseCurlyQuote"] = 8217,  
 4264 ["rsquo"] = 8217,  
 4265 ["rsquor"] = 8217,  
 4266 ["lsquor"] = 8218,  
 4267 ["sbquo"] = 8218,  
 4268 ["OpenCurlyDoubleQuote"] = 8220,  
 4269 ["ldquo"] = 8220,  
 4270 ["CloseCurlyDoubleQuote"] = 8221,  
 4271 ["rdquo"] = 8221,  
 4272 ["rdquor"] = 8221,  
 4273 ["bdquo"] = 8222,  
 4274 ["ldquor"] = 8222,  
 4275 ["dagger"] = 8224,  
 4276 ["Dagger"] = 8225,  
 4277 ["ddagger"] = 8225,  
 4278 ["bull"] = 8226,  
 4279 ["bullet"] = 8226,  
 4280 ["nldr"] = 8229,  
 4281 ["hellip"] = 8230,  
 4282 ["mldr"] = 8230,  
 4283 ["permil"] = 8240,  
 4284 ["pertenk"] = 8241,  
 4285 ["prime"] = 8242,  
 4286 ["Prime"] = 8243,  
 4287 ["tprime"] = 8244,  
 4288 ["backprime"] = 8245,  
 4289 ["bprime"] = 8245,  
 4290 ["lsaquo"] = 8249,  
 4291 ["rsaquo"] = 8250,  
 4292 ["OverBar"] = 8254,

```

4293 ["oline"] = 8254,
4294 ["caret"] = 8257,
4295 ["hybull"] = 8259,
4296 ["frasl"] = 8260,
4297 ["bsemi"] = 8271,
4298 ["qprime"] = 8279,
4299 ["MediumSpace"] = 8287,
4300 ["ThickSpace"] = {8287, 8202},
4301 ["NoBreak"] = 8288,
4302 ["ApplyFunction"] = 8289,
4303 ["af"] = 8289,
4304 ["InvisibleTimes"] = 8290,
4305 ["it"] = 8290,
4306 ["InvisibleComma"] = 8291,
4307 ["ic"] = 8291,
4308 ["euro"] = 8364,
4309 ["TripleDot"] = 8411,
4310 ["tdot"] = 8411,
4311 ["DotDot"] = 8412,
4312 ["Copf"] = 8450,
4313 ["complexes"] = 8450,
4314 ["incare"] = 8453,
4315 ["gscr"] = 8458,
4316 ["HilbertSpace"] = 8459,
4317 ["Hscr"] = 8459,
4318 ["hamilt"] = 8459,
4319 ["Hfr"] = 8460,
4320 ["Poincareplane"] = 8460,
4321 ["Hopf"] = 8461,
4322 ["quaternions"] = 8461,
4323 ["planckh"] = 8462,
4324 ["hbar"] = 8463,
4325 ["hslash"] = 8463,
4326 ["planck"] = 8463,
4327 ["plankv"] = 8463,
4328 ["Iscr"] = 8464,
4329 ["imagline"] = 8464,
4330 ["Ifr"] = 8465,
4331 ["Im"] = 8465,
4332 ["image"] = 8465,
4333 ["imagpart"] = 8465,
4334 ["Laplacetrif"] = 8466,
4335 ["Lscr"] = 8466,
4336 ["lagran"] = 8466,
4337 ["ell"] = 8467,
4338 ["Nopf"] = 8469,
4339 ["naturals"] = 8469,

```

```

4340 ["numero"] = 8470,
4341 ["copysr"] = 8471,
4342 ["weierp"] = 8472,
4343 ["wp"] = 8472,
4344 ["Popf"] = 8473,
4345 ["primes"] = 8473,
4346 ["Qopf"] = 8474,
4347 ["rationals"] = 8474,
4348 ["Rscr"] = 8475,
4349 ["realine"] = 8475,
4350 ["Re"] = 8476,
4351 ["Rfr"] = 8476,
4352 ["real"] = 8476,
4353 ["realpart"] = 8476,
4354 ["Ropf"] = 8477,
4355 ["reals"] = 8477,
4356 ["rx"] = 8478,
4357 ["TRADE"] = 8482,
4358 ["trade"] = 8482,
4359 ["Zopf"] = 8484,
4360 ["integers"] = 8484,
4361 ["mho"] = 8487,
4362 ["Zfr"] = 8488,
4363 ["zeetrif"] = 8488,
4364 ["iiota"] = 8489,
4365 ["Bernoullis"] = 8492,
4366 ["Bscr"] = 8492,
4367 ["bernou"] = 8492,
4368 ["Cayleys"] = 8493,
4369 ["Cfr"] = 8493,
4370 ["escr"] = 8495,
4371 ["Escr"] = 8496,
4372 ["expectation"] = 8496,
4373 ["Fouriertrf"] = 8497,
4374 ["Fscr"] = 8497,
4375 ["Mellintrf"] = 8499,
4376 ["Mscr"] = 8499,
4377 ["phmmat"] = 8499,
4378 ["order"] = 8500,
4379 ["orderof"] = 8500,
4380 ["oscr"] = 8500,
4381 ["alefsym"] = 8501,
4382 ["aleph"] = 8501,
4383 ["beth"] = 8502,
4384 ["gimel"] = 8503,
4385 ["daleth"] = 8504,
4386 ["CapitalDifferentialD"] = 8517,

```

```

4387 ["DD"] = 8517,
4388 ["DifferentialD"] = 8518,
4389 ["dd"] = 8518,
4390 ["ExponentialE"] = 8519,
4391 ["ee"] = 8519,
4392 ["exponentiale"] = 8519,
4393 ["ImaginaryI"] = 8520,
4394 ["ii"] = 8520,
4395 ["frac13"] = 8531,
4396 ["frac23"] = 8532,
4397 ["frac15"] = 8533,
4398 ["frac25"] = 8534,
4399 ["frac35"] = 8535,
4400 ["frac45"] = 8536,
4401 ["frac16"] = 8537,
4402 ["frac56"] = 8538,
4403 ["frac18"] = 8539,
4404 ["frac38"] = 8540,
4405 ["frac58"] = 8541,
4406 ["frac78"] = 8542,
4407 ["LeftArrow"] = 8592,
4408 ["ShortLeftArrow"] = 8592,
4409 ["larr"] = 8592,
4410 ["leftarrow"] = 8592,
4411 ["slarr"] = 8592,
4412 ["ShortUpArrow"] = 8593,
4413 ["UpArrow"] = 8593,
4414 ["uarr"] = 8593,
4415 ["uparrow"] = 8593,
4416 ["RightArrow"] = 8594,
4417 ["ShortRightArrow"] = 8594,
4418 ["rarr"] = 8594,
4419 ["rightarrow"] = 8594,
4420 ["srarr"] = 8594,
4421 ["DownArrow"] = 8595,
4422 ["ShortDownArrow"] = 8595,
4423 ["darr"] = 8595,
4424 ["downarrow"] = 8595,
4425 ["LeftRightArrow"] = 8596,
4426 ["harr"] = 8596,
4427 ["leftrightarrow"] = 8596,
4428 ["UpDownArrow"] = 8597,
4429 ["updownarrow"] = 8597,
4430 ["varr"] = 8597,
4431 ["UpperLeftArrow"] = 8598,
4432 ["nwarr"] = 8598,
4433 ["nwarrow"] = 8598,

```

```

4434 ["UpperRightArrow"] = 8599,
4435 ["nearr"] = 8599,
4436 ["nearrow"] = 8599,
4437 ["LowerRightArrow"] = 8600,
4438 ["searr"] = 8600,
4439 ["searrow"] = 8600,
4440 ["LowerLeftArrow"] = 8601,
4441 ["swarr"] = 8601,
4442 ["swarrow"] = 8601,
4443 ["nlarr"] = 8602,
4444 ["nleftarrow"] = 8602,
4445 ["nrarr"] = 8603,
4446 ["nrightarrow"] = 8603,
4447 ["nrarrw"] = {8605, 824},
4448 ["rarrw"] = 8605,
4449 ["rightsquigarrow"] = 8605,
4450 ["Larr"] = 8606,
4451 ["twoheadleftarrow"] = 8606,
4452 ["Uarr"] = 8607,
4453 ["Rarr"] = 8608,
4454 ["twoheadrightarrow"] = 8608,
4455 ["Darr"] = 8609,
4456 ["larrtl"] = 8610,
4457 ["leftarrowtail"] = 8610,
4458 ["rarrtl"] = 8611,
4459 ["rightarrowtail"] = 8611,
4460 ["LeftTeeArrow"] = 8612,
4461 ["mapstoleft"] = 8612,
4462 ["UpTeeArrow"] = 8613,
4463 ["mapstoup"] = 8613,
4464 ["RightTeeArrow"] = 8614,
4465 ["map"] = 8614,
4466 ["mapsto"] = 8614,
4467 ["DownTeeArrow"] = 8615,
4468 ["mapstodown"] = 8615,
4469 ["hookleftarrow"] = 8617,
4470 ["larrhk"] = 8617,
4471 ["hookrightarrow"] = 8618,
4472 ["rarrhk"] = 8618,
4473 ["larrlp"] = 8619,
4474 ["looparrowleft"] = 8619,
4475 ["looparrowright"] = 8620,
4476 ["rarrlp"] = 8620,
4477 ["harrw"] = 8621,
4478 ["leftrightsquigarrow"] = 8621,
4479 ["nharr"] = 8622,
4480 ["nleftrightarrow"] = 8622,

```



```

4481 ["Lsh"] = 8624,
4482 ["lsh"] = 8624,
4483 ["Rsh"] = 8625,
4484 ["rsh"] = 8625,
4485 ["ldsh"] = 8626,
4486 ["rdsh"] = 8627,
4487 ["crarr"] = 8629,
4488 ["cularr"] = 8630,
4489 ["curvearrowleft"] = 8630,
4490 ["curarr"] = 8631,
4491 ["curvearrowright"] = 8631,
4492 ["circlearrowleft"] = 8634,
4493 ["olarr"] = 8634,
4494 ["circlearrowright"] = 8635,
4495 ["orarr"] = 8635,
4496 ["LeftVector"] = 8636,
4497 ["leftharpoonup"] = 8636,
4498 ["lharu"] = 8636,
4499 ["DownLeftVector"] = 8637,
4500 ["leftharpoondown"] = 8637,
4501 ["lhard"] = 8637,
4502 ["RightUpVector"] = 8638,
4503 ["uharr"] = 8638,
4504 ["upharpoonright"] = 8638,
4505 ["LeftUpVector"] = 8639,
4506 ["uharl"] = 8639,
4507 ["upharpoonleft"] = 8639,
4508 ["RightVector"] = 8640,
4509 ["rharu"] = 8640,
4510 ["rightharpoonup"] = 8640,
4511 ["DownRightVector"] = 8641,
4512 ["rhard"] = 8641,
4513 ["rightharpoondown"] = 8641,
4514 ["RightDownVector"] = 8642,
4515 ["dharr"] = 8642,
4516 ["downharpoonright"] = 8642,
4517 ["LeftDownVector"] = 8643,
4518 ["dharl"] = 8643,
4519 ["downharpoonleft"] = 8643,
4520 ["RightArrowLeftArrow"] = 8644,
4521 ["rightleftarrows"] = 8644,
4522 ["rlarr"] = 8644,
4523 ["UpArrowDownArrow"] = 8645,
4524 ["udarr"] = 8645,
4525 ["LeftArrowRightArrow"] = 8646,
4526 ["leftrightarrows"] = 8646,
4527 ["lrarr"] = 8646,

```

```

4528 ["leftleftarrows"] = 8647,
4529 ["llarr"] = 8647,
4530 ["upuparrows"] = 8648,
4531 ["uuarr"] = 8648,
4532 ["rightrightarrows"] = 8649,
4533 ["rrarr"] = 8649,
4534 ["ddarr"] = 8650,
4535 ["downdownarrows"] = 8650,
4536 ["ReverseEquilibrium"] = 8651,
4537 ["leftrightharpoons"] = 8651,
4538 ["lrhar"] = 8651,
4539 ["Equilibrium"] = 8652,
4540 ["rightleftharpoons"] = 8652,
4541 ["rlhar"] = 8652,
4542 ["nLeftarrow"] = 8653,
4543 ["nlArr"] = 8653,
4544 ["nLeftrightarrow"] = 8654,
4545 ["nhArr"] = 8654,
4546 ["nRightarrow"] = 8655,
4547 ["nrArr"] = 8655,
4548 ["DoubleLeftArrow"] = 8656,
4549 ["Leftarrow"] = 8656,
4550 ["lArr"] = 8656,
4551 ["DoubleUpArrow"] = 8657,
4552 ["Uparrow"] = 8657,
4553 ["uArr"] = 8657,
4554 ["DoubleRightArrow"] = 8658,
4555 ["Implies"] = 8658,
4556 ["Rightarrow"] = 8658,
4557 ["rArr"] = 8658,
4558 ["DoubleDownArrow"] = 8659,
4559 ["Downarrow"] = 8659,
4560 ["dArr"] = 8659,
4561 ["DoubleLeftRightArrow"] = 8660,
4562 ["Leftrightarrow"] = 8660,
4563 ["hArr"] = 8660,
4564 ["iff"] = 8660,
4565 ["DoubleUpDownArrow"] = 8661,
4566 ["Updownarrow"] = 8661,
4567 ["vArr"] = 8661,
4568 ["nwArr"] = 8662,
4569 ["neArr"] = 8663,
4570 ["seArr"] = 8664,
4571 ["swArr"] = 8665,
4572 ["Lleftarrow"] = 8666,
4573 ["lAarr"] = 8666,
4574 ["Rrightarrow"] = 8667,

```

```

4575 ["rAarr"] = 8667,
4576 ["zigrarr"] = 8669,
4577 ["LeftArrowBar"] = 8676,
4578 ["larrb"] = 8676,
4579 ["RightArrowBar"] = 8677,
4580 ["rarrb"] = 8677,
4581 ["DownArrowUpArrow"] = 8693,
4582 ["duarr"] = 8693,
4583 ["loarr"] = 8701,
4584 ["roarr"] = 8702,
4585 ["hoarr"] = 8703,
4586 ["ForAll"] = 8704,
4587 ["forall"] = 8704,
4588 ["comp"] = 8705,
4589 ["complement"] = 8705,
4590 ["PartialD"] = 8706,
4591 ["npart"] = {8706, 824},
4592 ["part"] = 8706,
4593 ["Exists"] = 8707,
4594 ["exist"] = 8707,
4595 ["NotExists"] = 8708,
4596 ["nexist"] = 8708,
4597 ["nexists"] = 8708,
4598 ["empty"] = 8709,
4599 ["emptyset"] = 8709,
4600 ["emptyv"] = 8709,
4601 ["varnothing"] = 8709,
4602 ["Del"] = 8711,
4603 ["nabla"] = 8711,
4604 ["Element"] = 8712,
4605 ["in"] = 8712,
4606 ["isin"] = 8712,
4607 ["isinv"] = 8712,
4608 ["NotElement"] = 8713,
4609 ["notin"] = 8713,
4610 ["notinva"] = 8713,
4611 ["ReverseElement"] = 8715,
4612 ["SuchThat"] = 8715,
4613 ["ni"] = 8715,
4614 ["niv"] = 8715,
4615 ["NotReverseElement"] = 8716,
4616 ["notni"] = 8716,
4617 ["notniva"] = 8716,
4618 ["Product"] = 8719,
4619 ["prod"] = 8719,
4620 ["Coproduct"] = 8720,
4621 ["coprod"] = 8720,

```

```

4622 ["Sum"] = 8721,
4623 ["sum"] = 8721,
4624 ["minus"] = 8722,
4625 ["MinusPlus"] = 8723,
4626 ["mnplus"] = 8723,
4627 ["mp"] = 8723,
4628 ["dotplus"] = 8724,
4629 ["plusdo"] = 8724,
4630 ["Backslash"] = 8726,
4631 ["setminus"] = 8726,
4632 ["setmn"] = 8726,
4633 ["smallsetminus"] = 8726,
4634 ["ssetmn"] = 8726,
4635 ["lowast"] = 8727,
4636 ["SmallCircle"] = 8728,
4637 ["compfn"] = 8728,
4638 ["Sqrt"] = 8730,
4639 ["radic"] = 8730,
4640 ["Proportional"] = 8733,
4641 ["prop"] = 8733,
4642 ["propto"] = 8733,
4643 ["varpropto"] = 8733,
4644 ["vprop"] = 8733,
4645 ["infin"] = 8734,
4646 ["angrt"] = 8735,
4647 ["ang"] = 8736,
4648 ["angle"] = 8736,
4649 ["nang"] = {8736, 8402},
4650 ["angmsd"] = 8737,
4651 ["measuredangle"] = 8737,
4652 ["angsph"] = 8738,
4653 ["VerticalBar"] = 8739,
4654 ["mid"] = 8739,
4655 ["shortmid"] = 8739,
4656 ["smid"] = 8739,
4657 ["NotVerticalBar"] = 8740,
4658 ["nmid"] = 8740,
4659 ["nshortmid"] = 8740,
4660 ["nsmid"] = 8740,
4661 ["DoubleVerticalBar"] = 8741,
4662 ["par"] = 8741,
4663 ["parallel"] = 8741,
4664 ["shortparallel"] = 8741,
4665 ["spar"] = 8741,
4666 ["NotDoubleVerticalBar"] = 8742,
4667 ["npar"] = 8742,
4668 ["nparallel"] = 8742,

```

```

4669 ["nshortparallel"] = 8742,
4670 ["nspar"] = 8742,
4671 ["and"] = 8743,
4672 ["wedge"] = 8743,
4673 ["or"] = 8744,
4674 ["vee"] = 8744,
4675 ["cap"] = 8745,
4676 ["caps"] = {8745, 65024},
4677 ["cup"] = 8746,
4678 ["cups"] = {8746, 65024},
4679 ["Integral"] = 8747,
4680 ["int"] = 8747,
4681 ["Int"] = 8748,
4682 ["iiint"] = 8749,
4683 ["tint"] = 8749,
4684 ["ContourIntegral"] = 8750,
4685 ["conint"] = 8750,
4686 ["oint"] = 8750,
4687 ["Conint"] = 8751,
4688 ["DoubleContourIntegral"] = 8751,
4689 ["Cconint"] = 8752,
4690 ["cwint"] = 8753,
4691 ["ClockwiseContourIntegral"] = 8754,
4692 ["cwconint"] = 8754,
4693 ["CounterClockwiseContourIntegral"] = 8755,
4694 ["awconint"] = 8755,
4695 ["Therefore"] = 8756,
4696 ["there4"] = 8756,
4697 ["therefore"] = 8756,
4698 ["Because"] = 8757,
4699 ["because"] = 8757,
4700 ["because"] = 8757,
4701 ["ratio"] = 8758,
4702 ["Colon"] = 8759,
4703 ["Proportion"] = 8759,
4704 ["dotminus"] = 8760,
4705 ["minusd"] = 8760,
4706 ["mDDot"] = 8762,
4707 ["homtht"] = 8763,
4708 ["Tilde"] = 8764,
4709 ["nvsim"] = {8764, 8402},
4710 ["sim"] = 8764,
4711 ["thicksim"] = 8764,
4712 ["thksim"] = 8764,
4713 ["backsim"] = 8765,
4714 ["bsim"] = 8765,
4715 ["race"] = {8765, 817},

```

```

4716 ["ac"] = 8766,
4717 ["acE"] = {8766, 819},
4718 ["mstpos"] = 8766,
4719 ["acd"] = 8767,
4720 ["VerticalTilde"] = 8768,
4721 ["wr"] = 8768,
4722 ["wreath"] = 8768,
4723 ["NotTilde"] = 8769,
4724 ["nsim"] = 8769,
4725 ["EqualTilde"] = 8770,
4726 ["NotEqualTilde"] = {8770, 824},
4727 ["eqsim"] = 8770,
4728 ["esim"] = 8770,
4729 ["nesim"] = {8770, 824},
4730 ["TildeEqual"] = 8771,
4731 ["sime"] = 8771,
4732 ["simeq"] = 8771,
4733 ["NotTildeEqual"] = 8772,
4734 ["nsime"] = 8772,
4735 ["nsimeq"] = 8772,
4736 ["TildeFullEqual"] = 8773,
4737 ["cong"] = 8773,
4738 ["simne"] = 8774,
4739 ["NotTildeFullEqual"] = 8775,
4740 ["ncong"] = 8775,
4741 ["TildeTilde"] = 8776,
4742 ["ap"] = 8776,
4743 ["approx"] = 8776,
4744 ["asymp"] = 8776,
4745 ["thickapprox"] = 8776,
4746 ["thkap"] = 8776,
4747 ["NotTildeTilde"] = 8777,
4748 ["nap"] = 8777,
4749 ["napprox"] = 8777,
4750 ["ape"] = 8778,
4751 ["approxpeq"] = 8778,
4752 ["apid"] = 8779,
4753 ["napid"] = {8779, 824},
4754 ["backcong"] = 8780,
4755 ["bcong"] = 8780,
4756 ["CupCap"] = 8781,
4757 ["asympeq"] = 8781,
4758 ["nvap"] = {8781, 8402},
4759 ["Bumpeq"] = 8782,
4760 ["HumpDownHump"] = 8782,
4761 ["NotHumpDownHump"] = {8782, 824},
4762 ["bump"] = 8782,

```

```

4763 ["nbump"] = {8782, 824},
4764 ["HumpEqual"] = 8783,
4765 ["NotHumpEqual"] = {8783, 824},
4766 ["bumpe"] = 8783,
4767 ["bumpeq"] = 8783,
4768 ["nbumpe"] = {8783, 824},
4769 ["DotEqual"] = 8784,
4770 ["doteq"] = 8784,
4771 ["esdot"] = 8784,
4772 ["nedot"] = {8784, 824},
4773 ["doteqdot"] = 8785,
4774 ["eDot"] = 8785,
4775 ["efDot"] = 8786,
4776 ["fallingdotseq"] = 8786,
4777 ["erDot"] = 8787,
4778 ["risingdotseq"] = 8787,
4779 ["Assign"] = 8788,
4780 ["colone"] = 8788,
4781 ["coloneq"] = 8788,
4782 ["ecolon"] = 8789,
4783 ["eqcolon"] = 8789,
4784 ["ecir"] = 8790,
4785 ["eqcirc"] = 8790,
4786 ["circeq"] = 8791,
4787 ["cire"] = 8791,
4788 ["wedgeq"] = 8793,
4789 ["veeeq"] = 8794,
4790 ["triangleq"] = 8796,
4791 ["trie"] = 8796,
4792 ["equest"] = 8799,
4793 ["questeq"] = 8799,
4794 ["NotEqual"] = 8800,
4795 ["ne"] = 8800,
4796 ["Congruent"] = 8801,
4797 ["bnequiv"] = {8801, 8421},
4798 ["equiv"] = 8801,
4799 ["NotCongruent"] = 8802,
4800 ["nequiv"] = 8802,
4801 ["le"] = 8804,
4802 ["leq"] = 8804,
4803 ["nvle"] = {8804, 8402},
4804 ["GreaterEqual"] = 8805,
4805 ["ge"] = 8805,
4806 ["geq"] = 8805,
4807 ["nvge"] = {8805, 8402},
4808 ["LessFullEqual"] = 8806,
4809 ["lE"] = 8806,

```

```

4810 ["leqq"] = 8806,
4811 ["nlE"] = {8806, 824},
4812 ["nleqq"] = {8806, 824},
4813 ["GreaterFullEqual"] = 8807,
4814 ["NotGreaterFullEqual"] = {8807, 824},
4815 ["gE"] = 8807,
4816 ["geqq"] = 8807,
4817 ["ngE"] = {8807, 824},
4818 ["ngeqq"] = {8807, 824},
4819 ["lnE"] = 8808,
4820 ["lneqq"] = 8808,
4821 ["lvertneqq"] = {8808, 65024},
4822 ["lvnE"] = {8808, 65024},
4823 ["gnE"] = 8809,
4824 ["gneqq"] = 8809,
4825 ["gvertneqq"] = {8809, 65024},
4826 ["gvnE"] = {8809, 65024},
4827 ["Lt"] = 8810,
4828 ["NestedLessLess"] = 8810,
4829 ["NotLessLess"] = {8810, 824},
4830 ["ll"] = 8810,
4831 ["nLt"] = {8810, 8402},
4832 ["nLtv"] = {8810, 824},
4833 ["Gt"] = 8811,
4834 ["NestedGreaterGreater"] = 8811,
4835 ["NotGreaterGreater"] = {8811, 824},
4836 ["gg"] = 8811,
4837 ["nGt"] = {8811, 8402},
4838 ["nGtv"] = {8811, 824},
4839 ["between"] = 8812,
4840 ["twixt"] = 8812,
4841 ["NotCupCap"] = 8813,
4842 ["NotLess"] = 8814,
4843 ["nless"] = 8814,
4844 ["nlt"] = 8814,
4845 ["NotGreater"] = 8815,
4846 ["ngt"] = 8815,
4847 ["ngtr"] = 8815,
4848 ["NotLessEqual"] = 8816,
4849 ["nle"] = 8816,
4850 ["nleq"] = 8816,
4851 ["NotGreaterEqual"] = 8817,
4852 ["nge"] = 8817,
4853 ["ngeq"] = 8817,
4854 ["LessTilde"] = 8818,
4855 ["lesssim"] = 8818,
4856 ["lsim"] = 8818,

```



```

4857 ["GreaterTilde"] = 8819,
4858 ["gsim"] = 8819,
4859 ["gtrsim"] = 8819,
4860 ["NotLessTilde"] = 8820,
4861 ["nlsim"] = 8820,
4862 ["NotGreaterTilde"] = 8821,
4863 ["ngsim"] = 8821,
4864 ["LessGreater"] = 8822,
4865 ["lessgtr"] = 8822,
4866 ["lg"] = 8822,
4867 ["GreaterLess"] = 8823,
4868 ["gl"] = 8823,
4869 ["gtrless"] = 8823,
4870 ["NotLessGreater"] = 8824,
4871 ["ntlg"] = 8824,
4872 ["NotGreaterLess"] = 8825,
4873 ["ntgl"] = 8825,
4874 ["Precedes"] = 8826,
4875 ["pr"] = 8826,
4876 ["prec"] = 8826,
4877 ["Succeeds"] = 8827,
4878 ["sc"] = 8827,
4879 ["succ"] = 8827,
4880 ["PrecedesSlantEqual"] = 8828,
4881 ["prcue"] = 8828,
4882 ["preccurlyeq"] = 8828,
4883 ["SucceedsSlantEqual"] = 8829,
4884 ["sccue"] = 8829,
4885 ["succcurlyeq"] = 8829,
4886 ["PrecedesTilde"] = 8830,
4887 ["precsim"] = 8830,
4888 ["prsim"] = 8830,
4889 ["NotSucceedsTilde"] = {8831, 824},
4890 ["SucceedsTilde"] = 8831,
4891 ["scsim"] = 8831,
4892 ["succsim"] = 8831,
4893 ["NotPrecedes"] = 8832,
4894 ["npr"] = 8832,
4895 ["nprec"] = 8832,
4896 ["NotSucceeds"] = 8833,
4897 ["nsc"] = 8833,
4898 ["nsucc"] = 8833,
4899 ["NotSubset"] = {8834, 8402},
4900 ["nsubset"] = {8834, 8402},
4901 ["sub"] = 8834,
4902 ["subset"] = 8834,
4903 ["vnsup"] = {8834, 8402},

```

```

4904 ["NotSuperset"] = {8835, 8402},
4905 ["Superset"] = 8835,
4906 ["nsupset"] = {8835, 8402},
4907 ["sup"] = 8835,
4908 ["supset"] = 8835,
4909 ["vnsup"] = {8835, 8402},
4910 ["nsub"] = 8836,
4911 ["nsup"] = 8837,
4912 ["SubsetEqual"] = 8838,
4913 ["sube"] = 8838,
4914 ["subseteq"] = 8838,
4915 ["SupersetEqual"] = 8839,
4916 ["supe"] = 8839,
4917 ["supseteq"] = 8839,
4918 ["NotSubsetEqual"] = 8840,
4919 ["nsube"] = 8840,
4920 ["nsubseteq"] = 8840,
4921 ["NotSupersetEqual"] = 8841,
4922 ["nsupe"] = 8841,
4923 ["nsupseteq"] = 8841,
4924 ["subne"] = 8842,
4925 ["subsetneq"] = 8842,
4926 ["varsubsetneq"] = {8842, 65024},
4927 ["vsubne"] = {8842, 65024},
4928 ["supne"] = 8843,
4929 ["supsetneq"] = 8843,
4930 ["varsupsetneq"] = {8843, 65024},
4931 ["vsupne"] = {8843, 65024},
4932 ["cupdot"] = 8845,
4933 ["UnionPlus"] = 8846,
4934 ["uplus"] = 8846,
4935 ["NotSquareSubset"] = {8847, 824},
4936 ["SquareSubset"] = 8847,
4937 ["sqsub"] = 8847,
4938 ["sqsubset"] = 8847,
4939 ["NotSquareSuperset"] = {8848, 824},
4940 ["SquareSuperset"] = 8848,
4941 ["sqsup"] = 8848,
4942 ["sqsupset"] = 8848,
4943 ["SquareSubsetEqual"] = 8849,
4944 ["sqsube"] = 8849,
4945 ["sqsubseteq"] = 8849,
4946 ["SquareSupersetEqual"] = 8850,
4947 ["sqsupe"] = 8850,
4948 ["sqsupseteq"] = 8850,
4949 ["SquareIntersection"] = 8851,
4950 ["sqcap"] = 8851,

```

```

4951 ["sqcaps"] = {8851, 65024},
4952 ["SquareUnion"] = 8852,
4953 ["sqcup"] = 8852,
4954 ["sqcups"] = {8852, 65024},
4955 ["CirclePlus"] = 8853,
4956 ["oplus"] = 8853,
4957 ["CircleMinus"] = 8854,
4958 ["ominus"] = 8854,
4959 ["CircleTimes"] = 8855,
4960 ["otimes"] = 8855,
4961 ["osol"] = 8856,
4962 ["CircleDot"] = 8857,
4963 ["odot"] = 8857,
4964 ["circledcirc"] = 8858,
4965 ["ocir"] = 8858,
4966 ["circledast"] = 8859,
4967 ["oast"] = 8859,
4968 ["circleddash"] = 8861,
4969 ["odash"] = 8861,
4970 ["boxplus"] = 8862,
4971 ["plusb"] = 8862,
4972 ["boxminus"] = 8863,
4973 ["minusb"] = 8863,
4974 ["boxtimes"] = 8864,
4975 ["timesb"] = 8864,
4976 ["dotsquare"] = 8865,
4977 ["sdotb"] = 8865,
4978 ["RightTee"] = 8866,
4979 ["vdash"] = 8866,
4980 ["LeftTee"] = 8867,
4981 ["dashv"] = 8867,
4982 ["DownTee"] = 8868,
4983 ["top"] = 8868,
4984 ["UpTee"] = 8869,
4985 ["bot"] = 8869,
4986 ["bottom"] = 8869,
4987 ["perp"] = 8869,
4988 ["models"] = 8871,
4989 ["DoubleRightTee"] = 8872,
4990 ["vDash"] = 8872,
4991 ["Vdash"] = 8873,
4992 ["Vvdash"] = 8874,
4993 ["VDash"] = 8875,
4994 ["nvDash"] = 8876,
4995 ["nvDash"] = 8877,
4996 ["nVdash"] = 8878,
4997 ["nVDash"] = 8879,

```

```

4998 ["prurel"] = 8880,
4999 ["LeftTriangle"] = 8882,
5000 ["vartriangleleft"] = 8882,
5001 ["vltri"] = 8882,
5002 ["RightTriangle"] = 8883,
5003 ["vartriangleright"] = 8883,
5004 ["vrtri"] = 8883,
5005 ["LeftTriangleEqual"] = 8884,
5006 ["ltrie"] = 8884,
5007 ["nvltrie"] = {8884, 8402},
5008 ["trianglelefteq"] = 8884,
5009 ["RightTriangleEqual"] = 8885,
5010 ["nvrtrie"] = {8885, 8402},
5011 ["rtrie"] = 8885,
5012 ["trianglerighteq"] = 8885,
5013 ["origof"] = 8886,
5014 ["imof"] = 8887,
5015 ["multimap"] = 8888,
5016 ["mumap"] = 8888,
5017 ["hercon"] = 8889,
5018 ["intcal"] = 8890,
5019 ["intercal"] = 8890,
5020 ["veebar"] = 8891,
5021 ["barvee"] = 8893,
5022 ["angrtvb"] = 8894,
5023 ["ltri"] = 8895,
5024 ["Wedge"] = 8896,
5025 ["bigwedge"] = 8896,
5026 ["xwedge"] = 8896,
5027 ["Vee"] = 8897,
5028 ["bigvee"] = 8897,
5029 ["xvee"] = 8897,
5030 ["Intersection"] = 8898,
5031 ["bigcap"] = 8898,
5032 ["xcap"] = 8898,
5033 ["Union"] = 8899,
5034 ["bigcup"] = 8899,
5035 ["xcup"] = 8899,
5036 ["Diamond"] = 8900,
5037 ["diam"] = 8900,
5038 ["diamond"] = 8900,
5039 ["sdot"] = 8901,
5040 ["Star"] = 8902,
5041 ["sstarf"] = 8902,
5042 ["divideontimes"] = 8903,
5043 ["divonx"] = 8903,
5044 ["bowtie"] = 8904,

```

```

5045 ["ltimes"] = 8905,
5046 ["rtimes"] = 8906,
5047 ["leftthreetimes"] = 8907,
5048 ["lthree"] = 8907,
5049 ["rightthreetimes"] = 8908,
5050 ["rthree"] = 8908,
5051 ["backsimeq"] = 8909,
5052 ["bsime"] = 8909,
5053 ["curlyvee"] = 8910,
5054 ["cuvee"] = 8910,
5055 ["curlywedge"] = 8911,
5056 ["cuwed"] = 8911,
5057 ["Sub"] = 8912,
5058 ["Subset"] = 8912,
5059 ["Sup"] = 8913,
5060 ["Supset"] = 8913,
5061 ["Cap"] = 8914,
5062 ["Cup"] = 8915,
5063 ["fork"] = 8916,
5064 ["pitchfork"] = 8916,
5065 ["epar"] = 8917,
5066 ["lessdot"] = 8918,
5067 ["ltdot"] = 8918,
5068 ["gtdot"] = 8919,
5069 ["gtrdot"] = 8919,
5070 ["Ll"] = 8920,
5071 ["nLl"] = {8920, 824},
5072 ["Gg"] = 8921,
5073 ["ggg"] = 8921,
5074 ["nGg"] = {8921, 824},
5075 ["LessEqualGreater"] = 8922,
5076 ["leg"] = 8922,
5077 ["lesg"] = {8922, 65024},
5078 ["lesseqgtr"] = 8922,
5079 ["GreaterEqualLess"] = 8923,
5080 ["gel"] = 8923,
5081 ["gesl"] = {8923, 65024},
5082 ["gtreqless"] = 8923,
5083 ["cuepr"] = 8926,
5084 ["curlyeqprec"] = 8926,
5085 ["cuesc"] = 8927,
5086 ["curlyeqsucc"] = 8927,
5087 ["NotPrecedesSlantEqual"] = 8928,
5088 ["nprcue"] = 8928,
5089 ["NotSucceedsSlantEqual"] = 8929,
5090 ["nsccue"] = 8929,
5091 ["NotSquareSubsetEqual"] = 8930,

```

```

5092 ["nsqsube"] = 8930,
5093 ["NotSquareSupersetEqual"] = 8931,
5094 ["nsqsupe"] = 8931,
5095 ["lnsim"] = 8934,
5096 ["gnsim"] = 8935,
5097 ["precnsim"] = 8936,
5098 ["prnsim"] = 8936,
5099 ["scnsim"] = 8937,
5100 ["succnsim"] = 8937,
5101 ["NotLeftTriangle"] = 8938,
5102 ["nltri"] = 8938,
5103 ["ntriangleleft"] = 8938,
5104 ["NotRightTriangle"] = 8939,
5105 ["nrtri"] = 8939,
5106 ["ntriangleright"] = 8939,
5107 ["NotLeftTriangleEqual"] = 8940,
5108 ["nltrie"] = 8940,
5109 ["ntrianglelefteq"] = 8940,
5110 ["NotRightTriangleEqual"] = 8941,
5111 ["nrtrie"] = 8941,
5112 ["ntrianglerighteq"] = 8941,
5113 ["vellip"] = 8942,
5114 ["ctdot"] = 8943,
5115 ["utdot"] = 8944,
5116 ["dtdot"] = 8945,
5117 ["disin"] = 8946,
5118 ["isinsv"] = 8947,
5119 ["isins"] = 8948,
5120 ["isindot"] = 8949,
5121 ["notin dot"] = {8949, 824},
5122 ["notinvc"] = 8950,
5123 ["notin vb"] = 8951,
5124 ["isinE"] = 8953,
5125 ["notinE"] = {8953, 824},
5126 ["nisd"] = 8954,
5127 ["xnis"] = 8955,
5128 ["nis"] = 8956,
5129 ["notnivc"] = 8957,
5130 ["notnivb"] = 8958,
5131 ["barwed"] = 8965,
5132 ["barwedge"] = 8965,
5133 ["Barwed"] = 8966,
5134 ["doublebarwedge"] = 8966,
5135 ["LeftCeiling"] = 8968,
5136 ["lceil"] = 8968,
5137 ["RightCeiling"] = 8969,
5138 ["rceil"] = 8969,

```

```

5139 ["LeftFloor"] = 8970,
5140 ["lfloor"] = 8970,
5141 ["RightFloor"] = 8971,
5142 ["rfloor"] = 8971,
5143 ["drcrop"] = 8972,
5144 ["dlcrop"] = 8973,
5145 ["urcrop"] = 8974,
5146 ["ulcrop"] = 8975,
5147 ["bnot"] = 8976,
5148 ["proflin"] = 8978,
5149 ["profsurf"] = 8979,
5150 ["telrec"] = 8981,
5151 ["target"] = 8982,
5152 ["ulcorn"] = 8988,
5153 ["ulcorner"] = 8988,
5154 ["urcorn"] = 8989,
5155 ["urcorner"] = 8989,
5156 ["dlcorn"] = 8990,
5157 ["llcorner"] = 8990,
5158 ["drcorn"] = 8991,
5159 ["lrcorn"] = 8991,
5160 ["frown"] = 8994,
5161 ["sfrown"] = 8994,
5162 ["smile"] = 8995,
5163 ["ssmile"] = 8995,
5164 ["cylcty"] = 9005,
5165 ["profalar"] = 9006,
5166 ["topbot"] = 9014,
5167 ["ovbar"] = 9021,
5168 ["solbar"] = 9023,
5169 ["angzarr"] = 9084,
5170 ["lmoust"] = 9136,
5171 ["lmoustache"] = 9136,
5172 ["rmoust"] = 9137,
5173 ["rmoustache"] = 9137,
5174 ["OverBracket"] = 9140,
5175 ["tbrk"] = 9140,
5176 ["UnderBracket"] = 9141,
5177 ["bbrk"] = 9141,
5178 ["bbrktbrk"] = 9142,
5179 ["OverParenthesis"] = 9180,
5180 ["UnderParenthesis"] = 9181,
5181 ["OverBrace"] = 9182,
5182 ["UnderBrace"] = 9183,
5183 ["trpezium"] = 9186,
5184 ["elinters"] = 9191,
5185 ["blank"] = 9251,

```

```

5186 ["circledS"] = 9416,
5187 ["oS"] = 9416,
5188 ["HorizontalLine"] = 9472,
5189 ["boxh"] = 9472,
5190 ["boxv"] = 9474,
5191 ["boxdr"] = 9484,
5192 ["boxdl"] = 9488,
5193 ["boxur"] = 9492,
5194 ["boxul"] = 9496,
5195 ["boxvr"] = 9500,
5196 ["boxvl"] = 9508,
5197 ["boxhd"] = 9516,
5198 ["boxhu"] = 9524,
5199 ["boxvh"] = 9532,
5200 ["boxH"] = 9552,
5201 ["boxV"] = 9553,
5202 ["boxdR"] = 9554,
5203 ["boxDr"] = 9555,
5204 ["boxDR"] = 9556,
5205 ["boxdL"] = 9557,
5206 ["boxDL"] = 9558,
5207 ["boxDL"] = 9559,
5208 ["boxuR"] = 9560,
5209 ["boxUr"] = 9561,
5210 ["boxUR"] = 9562,
5211 ["boxuL"] = 9563,
5212 ["boxUL"] = 9564,
5213 ["boxUL"] = 9565,
5214 ["boxvR"] = 9566,
5215 ["boxVr"] = 9567,
5216 ["boxVR"] = 9568,
5217 ["boxvL"] = 9569,
5218 ["boxVL"] = 9570,
5219 ["boxVL"] = 9571,
5220 ["boxHd"] = 9572,
5221 ["boxhD"] = 9573,
5222 ["boxHD"] = 9574,
5223 ["boxHu"] = 9575,
5224 ["boxhU"] = 9576,
5225 ["boxHU"] = 9577,
5226 ["boxvH"] = 9578,
5227 ["boxVh"] = 9579,
5228 ["boxVH"] = 9580,
5229 ["uhblk"] = 9600,
5230 ["lhblk"] = 9604,
5231 ["block"] = 9608,
5232 ["blk14"] = 9617,

```



```

5233 ["blk12"] = 9618,
5234 ["blk34"] = 9619,
5235 ["Square"] = 9633,
5236 ["squ"] = 9633,
5237 ["square"] = 9633,
5238 ["FilledVerySmallSquare"] = 9642,
5239 ["blacksquare"] = 9642,
5240 ["squarf"] = 9642,
5241 ["squf"] = 9642,
5242 ["EmptyVerySmallSquare"] = 9643,
5243 ["rect"] = 9645,
5244 ["marker"] = 9646,
5245 ["fltns"] = 9649,
5246 ["bigtriangleup"] = 9651,
5247 ["xutri"] = 9651,
5248 ["blacktriangle"] = 9652,
5249 ["utrif"] = 9652,
5250 ["triangle"] = 9653,
5251 ["utri"] = 9653,
5252 ["blacktriangleright"] = 9656,
5253 ["rtrif"] = 9656,
5254 ["rtri"] = 9657,
5255 ["triangleright"] = 9657,
5256 ["bigtriangledown"] = 9661,
5257 ["xdtri"] = 9661,
5258 ["blacktriangledown"] = 9662,
5259 ["dtrif"] = 9662,
5260 ["dtri"] = 9663,
5261 ["triangledown"] = 9663,
5262 ["blacktriangleleft"] = 9666,
5263 ["ltrif"] = 9666,
5264 ["ltri"] = 9667,
5265 ["triangleleft"] = 9667,
5266 ["loz"] = 9674,
5267 ["lozenge"] = 9674,
5268 ["cir"] = 9675,
5269 ["tridot"] = 9708,
5270 ["bigcirc"] = 9711,
5271 ["xcirc"] = 9711,
5272 ["ultri"] = 9720,
5273 ["urtri"] = 9721,
5274 ["lltri"] = 9722,
5275 ["EmptySmallSquare"] = 9723,
5276 ["FilledSmallSquare"] = 9724,
5277 ["bigstar"] = 9733,
5278 ["starf"] = 9733,
5279 ["star"] = 9734,

```

```

5280 ["phone"] = 9742,
5281 ["female"] = 9792,
5282 ["male"] = 9794,
5283 ["spades"] = 9824,
5284 ["spadesuit"] = 9824,
5285 ["clubs"] = 9827,
5286 ["clubsuit"] = 9827,
5287 ["hearts"] = 9829,
5288 ["heartsuit"] = 9829,
5289 ["diamondsuit"] = 9830,
5290 ["diams"] = 9830,
5291 ["sung"] = 9834,
5292 ["flat"] = 9837,
5293 ["natur"] = 9838,
5294 ["natural"] = 9838,
5295 ["sharp"] = 9839,
5296 ["check"] = 10003,
5297 ["checkmark"] = 10003,
5298 ["cross"] = 10007,
5299 ["malt"] = 10016,
5300 ["maltese"] = 10016,
5301 ["sext"] = 10038,
5302 ["VerticalSeparator"] = 10072,
5303 ["lbbbrk"] = 10098,
5304 ["rbbrk"] = 10099,
5305 ["bsolhsub"] = 10184,
5306 ["suphsol"] = 10185,
5307 ["LeftDoubleBracket"] = 10214,
5308 ["lobrk"] = 10214,
5309 ["RightDoubleBracket"] = 10215,
5310 ["robrk"] = 10215,
5311 ["LeftAngleBracket"] = 10216,
5312 ["lang"] = 10216,
5313 ["langle"] = 10216,
5314 ["RightAngleBracket"] = 10217,
5315 ["rang"] = 10217,
5316 ["rangle"] = 10217,
5317 ["Lang"] = 10218,
5318 ["Rang"] = 10219,
5319 ["loang"] = 10220,
5320 ["roang"] = 10221,
5321 ["LongLeftArrow"] = 10229,
5322 ["longleftarrow"] = 10229,
5323 ["xlarr"] = 10229,
5324 ["LongRightArrow"] = 10230,
5325 ["longrightarrow"] = 10230,
5326 ["xrarr"] = 10230,

```

```

5327 ["LongLeftRightArrow"] = 10231,
5328 ["longlefttrightarrow"] = 10231,
5329 ["xharr"] = 10231,
5330 ["DoubleLongLeftArrow"] = 10232,
5331 ["Longleftarrow"] = 10232,
5332 ["xlArr"] = 10232,
5333 ["DoubleLongRightArrow"] = 10233,
5334 ["Longrightarrow"] = 10233,
5335 ["xrArr"] = 10233,
5336 ["DoubleLongLeftRightArrow"] = 10234,
5337 ["longlefttrightarrow"] = 10234,
5338 ["xhArr"] = 10234,
5339 ["longmapsto"] = 10236,
5340 ["xmap"] = 10236,
5341 ["dzigrarr"] = 10239,
5342 ["nvlArr"] = 10498,
5343 ["nvrArr"] = 10499,
5344 ["nvHarr"] = 10500,
5345 ["Map"] = 10501,
5346 ["lbarr"] = 10508,
5347 ["bkarow"] = 10509,
5348 ["rbarr"] = 10509,
5349 ["lBarr"] = 10510,
5350 ["dbkarow"] = 10511,
5351 ["rBarr"] = 10511,
5352 ["RBarr"] = 10512,
5353 ["drbkarow"] = 10512,
5354 ["DDottrahd"] = 10513,
5355 ["UpArrowBar"] = 10514,
5356 ["DownArrowBar"] = 10515,
5357 ["Rarrtl"] = 10518,
5358 ["latail"] = 10521,
5359 ["ratail"] = 10522,
5360 ["lAtail"] = 10523,
5361 ["rAtail"] = 10524,
5362 ["larrfs"] = 10525,
5363 ["rarrfs"] = 10526,
5364 ["larrbfs"] = 10527,
5365 ["rarrbfs"] = 10528,
5366 ["nwarhk"] = 10531,
5367 ["nearhk"] = 10532,
5368 ["hksearow"] = 10533,
5369 ["searhk"] = 10533,
5370 ["hkswarrow"] = 10534,
5371 ["swarhk"] = 10534,
5372 ["nwnear"] = 10535,
5373 ["nesear"] = 10536,

```

```

5374 ["toea"] = 10536,
5375 ["seswar"] = 10537,
5376 ["tosa"] = 10537,
5377 ["swnwar"] = 10538,
5378 ["nrarrc"] = {10547, 824},
5379 ["rarrc"] = 10547,
5380 ["cudarr"] = 10549,
5381 ["ldca"] = 10550,
5382 ["rdca"] = 10551,
5383 ["cudarrr"] = 10552,
5384 ["larrpl"] = 10553,
5385 ["curarrm"] = 10556,
5386 ["cularrp"] = 10557,
5387 ["rarrpl"] = 10565,
5388 ["harrcir"] = 10568,
5389 ["Uarroccir"] = 10569,
5390 ["lurdshar"] = 10570,
5391 ["ldrushar"] = 10571,
5392 ["LeftRightVector"] = 10574,
5393 ["RightUpDownVector"] = 10575,
5394 ["DownLeftRightVector"] = 10576,
5395 ["LeftUpDownVector"] = 10577,
5396 ["LeftVectorBar"] = 10578,
5397 ["RightVectorBar"] = 10579,
5398 ["RightUpVectorBar"] = 10580,
5399 ["RightDownVectorBar"] = 10581,
5400 ["DownLeftVectorBar"] = 10582,
5401 ["DownRightVectorBar"] = 10583,
5402 ["LeftUpVectorBar"] = 10584,
5403 ["LeftDownVectorBar"] = 10585,
5404 ["LeftTeeVector"] = 10586,
5405 ["RightTeeVector"] = 10587,
5406 ["RightUpTeeVector"] = 10588,
5407 ["RightDownTeeVector"] = 10589,
5408 ["DownLeftTeeVector"] = 10590,
5409 ["DownRightTeeVector"] = 10591,
5410 ["LeftUpTeeVector"] = 10592,
5411 ["LeftDownTeeVector"] = 10593,
5412 ["lHar"] = 10594,
5413 ["uHar"] = 10595,
5414 ["rHar"] = 10596,
5415 ["dHar"] = 10597,
5416 ["luruhar"] = 10598,
5417 ["ldrdhar"] = 10599,
5418 ["ruluhar"] = 10600,
5419 ["rdldhar"] = 10601,
5420 ["lharul"] = 10602,

```

```

5421 ["llhard"] = 10603,
5422 ["rharul"] = 10604,
5423 ["lrhard"] = 10605,
5424 ["UpEquilibrium"] = 10606,
5425 ["udhar"] = 10606,
5426 ["ReverseUpEquilibrium"] = 10607,
5427 ["duhar"] = 10607,
5428 ["RoundImplies"] = 10608,
5429 ["erarr"] = 10609,
5430 ["simrarr"] = 10610,
5431 ["larrsim"] = 10611,
5432 ["rarrsim"] = 10612,
5433 ["rarrap"] = 10613,
5434 ["ltlarr"] = 10614,
5435 ["gtrarr"] = 10616,
5436 ["subrarr"] = 10617,
5437 ["suplarr"] = 10619,
5438 ["lfisht"] = 10620,
5439 ["rfisht"] = 10621,
5440 ["ufisht"] = 10622,
5441 ["dfisht"] = 10623,
5442 ["lopar"] = 10629,
5443 ["ropar"] = 10630,
5444 ["lbrke"] = 10635,
5445 ["rbrke"] = 10636,
5446 ["lbrkslu"] = 10637,
5447 ["rbrksld"] = 10638,
5448 ["lbrksld"] = 10639,
5449 ["rbrkslu"] = 10640,
5450 ["langd"] = 10641,
5451 ["rangd"] = 10642,
5452 ["lparlt"] = 10643,
5453 ["rpargt"] = 10644,
5454 ["gtlPar"] = 10645,
5455 ["ltrPar"] = 10646,
5456 ["vzigzag"] = 10650,
5457 ["vangrt"] = 10652,
5458 ["angrtvbd"] = 10653,
5459 ["ange"] = 10660,
5460 ["range"] = 10661,
5461 ["dwangle"] = 10662,
5462 ["uwangle"] = 10663,
5463 ["angmsdaa"] = 10664,
5464 ["angmsdab"] = 10665,
5465 ["angmsdac"] = 10666,
5466 ["angmsdad"] = 10667,
5467 ["angmsdae"] = 10668,

```

```

5468 ["angmsdaf"] = 10669,
5469 ["angmsdag"] = 10670,
5470 ["angmsdah"] = 10671,
5471 ["bemptyv"] = 10672,
5472 ["demptyv"] = 10673,
5473 ["cemptyv"] = 10674,
5474 ["raemptyv"] = 10675,
5475 ["laemptyv"] = 10676,
5476 ["ohbar"] = 10677,
5477 ["omid"] = 10678,
5478 ["opar"] = 10679,
5479 ["operp"] = 10681,
5480 ["olcross"] = 10683,
5481 ["odsold"] = 10684,
5482 ["olcir"] = 10686,
5483 ["ofcir"] = 10687,
5484 ["olt"] = 10688,
5485 ["ogt"] = 10689,
5486 ["cirscir"] = 10690,
5487 ["cirE"] = 10691,
5488 ["solb"] = 10692,
5489 ["bsolb"] = 10693,
5490 ["boxbox"] = 10697,
5491 ["trish"] = 10701,
5492 ["rtriltri"] = 10702,
5493 ["LeftTriangleBar"] = 10703,
5494 ["NotLeftTriangleBar"] = {10703, 824},
5495 ["NotRightTriangleBar"] = {10704, 824},
5496 ["RightTriangleBar"] = 10704,
5497 ["iinfin"] = 10716,
5498 ["infintie"] = 10717,
5499 ["nvinfin"] = 10718,
5500 ["eparsl"] = 10723,
5501 ["smeparsl"] = 10724,
5502 ["eqvparsl"] = 10725,
5503 ["blacklozenge"] = 10731,
5504 ["lozf"] = 10731,
5505 ["RuleDelayed"] = 10740,
5506 ["dsol"] = 10742,
5507 ["bigodot"] = 10752,
5508 ["xodot"] = 10752,
5509 ["bigoplus"] = 10753,
5510 ["xoplus"] = 10753,
5511 ["bigotimes"] = 10754,
5512 ["xotime"] = 10754,
5513 ["biguplus"] = 10756,
5514 ["xuplus"] = 10756,

```

```

5515 ["bigscup"] = 10758,
5516 ["xscup"] = 10758,
5517 ["iiiint"] = 10764,
5518 ["qint"] = 10764,
5519 ["fpartint"] = 10765,
5520 ["cirfnint"] = 10768,
5521 ["awint"] = 10769,
5522 ["rppolint"] = 10770,
5523 ["scpolint"] = 10771,
5524 ["npolint"] = 10772,
5525 ["pointint"] = 10773,
5526 ["quatint"] = 10774,
5527 ["intlarhk"] = 10775,
5528 ["pluscir"] = 10786,
5529 ["plusacir"] = 10787,
5530 ["simplus"] = 10788,
5531 ["plusdu"] = 10789,
5532 ["plussim"] = 10790,
5533 ["plustwo"] = 10791,
5534 ["mcomma"] = 10793,
5535 ["minusdu"] = 10794,
5536 ["loplus"] = 10797,
5537 ["roplus"] = 10798,
5538 ["Cross"] = 10799,
5539 ["timesd"] = 10800,
5540 ["timesbar"] = 10801,
5541 ["smashp"] = 10803,
5542 ["lotimes"] = 10804,
5543 ["rotimes"] = 10805,
5544 ["otimesas"] = 10806,
5545 ["Otimes"] = 10807,
5546 ["odiv"] = 10808,
5547 ["triplus"] = 10809,
5548 ["triminus"] = 10810,
5549 ["tritime"] = 10811,
5550 ["intprod"] = 10812,
5551 ["iproduct"] = 10812,
5552 ["amalg"] = 10815,
5553 ["capdot"] = 10816,
5554 ["ncup"] = 10818,
5555 ["ncap"] = 10819,
5556 ["capand"] = 10820,
5557 ["cupor"] = 10821,
5558 ["cupcap"] = 10822,
5559 ["capcup"] = 10823,
5560 ["cupbrcap"] = 10824,
5561 ["capbrcup"] = 10825,

```

5562 ["cupcup"] = 10826,  
 5563 ["capcap"] = 10827,  
 5564 ["ccups"] = 10828,  
 5565 ["ccaps"] = 10829,  
 5566 ["ccupssm"] = 10832,  
 5567 ["And"] = 10835,  
 5568 ["Or"] = 10836,  
 5569 ["andand"] = 10837,  
 5570 ["oror"] = 10838,  
 5571 ["orslope"] = 10839,  
 5572 ["andslope"] = 10840,  
 5573 ["andv"] = 10842,  
 5574 ["orv"] = 10843,  
 5575 ["andd"] = 10844,  
 5576 ["ord"] = 10845,  
 5577 ["wedbar"] = 10847,  
 5578 ["sdote"] = 10854,  
 5579 ["simdot"] = 10858,  
 5580 ["congdote"] = 10861,  
 5581 ["ncongdot"] = {10861, 824},  
 5582 ["easter"] = 10862,  
 5583 ["apacir"] = 10863,  
 5584 ["apE"] = 10864,  
 5585 ["napE"] = {10864, 824},  
 5586 ["eplus"] = 10865,  
 5587 ["pluse"] = 10866,  
 5588 ["Esim"] = 10867,  
 5589 ["Colone"] = 10868,  
 5590 ["Equal"] = 10869,  
 5591 ["ddotseq"] = 10871,  
 5592 ["eDDot"] = 10871,  
 5593 ["equivDD"] = 10872,  
 5594 ["ltcir"] = 10873,  
 5595 ["gtcir"] = 10874,  
 5596 ["ltquest"] = 10875,  
 5597 ["gtquest"] = 10876,  
 5598 ["LessSlantEqual"] = 10877,  
 5599 ["NotLessSlantEqual"] = {10877, 824},  
 5600 ["leqslant"] = 10877,  
 5601 ["les"] = 10877,  
 5602 ["nleqslant"] = {10877, 824},  
 5603 ["nles"] = {10877, 824},  
 5604 ["GreaterSlantEqual"] = 10878,  
 5605 ["NotGreaterSlantEqual"] = {10878, 824},  
 5606 ["geqslant"] = 10878,  
 5607 ["ges"] = 10878,  
 5608 ["ngeqslant"] = {10878, 824},



```

5609 ["nges"] = {10878, 824},
5610 ["lesdot"] = 10879,
5611 ["gesdot"] = 10880,
5612 ["lesdoto"] = 10881,
5613 ["gesdoto"] = 10882,
5614 ["lesdotor"] = 10883,
5615 ["gesdoto1"] = 10884,
5616 ["lap"] = 10885,
5617 ["lessapprox"] = 10885,
5618 ["gap"] = 10886,
5619 ["gtrapprox"] = 10886,
5620 ["lne"] = 10887,
5621 ["lneq"] = 10887,
5622 ["gne"] = 10888,
5623 ["gneq"] = 10888,
5624 ["lnap"] = 10889,
5625 ["lnapprox"] = 10889,
5626 ["gnap"] = 10890,
5627 ["gnapprox"] = 10890,
5628 ["lEg"] = 10891,
5629 ["lesseqqgtr"] = 10891,
5630 ["gE1"] = 10892,
5631 ["gtreqqless"] = 10892,
5632 ["lsime"] = 10893,
5633 ["gsime"] = 10894,
5634 ["lsimg"] = 10895,
5635 ["gsim1"] = 10896,
5636 ["lgE"] = 10897,
5637 ["glE"] = 10898,
5638 ["lesges"] = 10899,
5639 ["gesles"] = 10900,
5640 ["els"] = 10901,
5641 ["eqslantless"] = 10901,
5642 ["egs"] = 10902,
5643 ["eqslantgtr"] = 10902,
5644 ["elsdot"] = 10903,
5645 ["egsdot"] = 10904,
5646 ["el"] = 10905,
5647 ["eg"] = 10906,
5648 ["sim1"] = 10909,
5649 ["simg"] = 10910,
5650 ["simlE"] = 10911,
5651 ["simgE"] = 10912,
5652 ["LessLess"] = 10913,
5653 ["NotNestedLessLess"] = {10913, 824},
5654 ["GreaterGreater"] = 10914,
5655 ["NotNestedGreaterGreater"] = {10914, 824},

```

```

5656 ["glj"] = 10916,
5657 ["gla"] = 10917,
5658 ["ltcc"] = 10918,
5659 ["gtcc"] = 10919,
5660 ["lescc"] = 10920,
5661 ["gescc"] = 10921,
5662 ["smt"] = 10922,
5663 ["lat"] = 10923,
5664 ["smte"] = 10924,
5665 ["smtes"] = {10924, 65024},
5666 ["late"] = 10925,
5667 ["lates"] = {10925, 65024},
5668 ["bumpE"] = 10926,
5669 ["NotPrecedesEqual"] = {10927, 824},
5670 ["PrecedesEqual"] = 10927,
5671 ["npre"] = {10927, 824},
5672 ["npreceq"] = {10927, 824},
5673 ["pre"] = 10927,
5674 ["preceq"] = 10927,
5675 ["NotSucceedsEqual"] = {10928, 824},
5676 ["SucceedsEqual"] = 10928,
5677 ["nsce"] = {10928, 824},
5678 ["nsucceq"] = {10928, 824},
5679 ["sce"] = 10928,
5680 ["succeq"] = 10928,
5681 ["prE"] = 10931,
5682 ["scE"] = 10932,
5683 ["precneqq"] = 10933,
5684 ["prnE"] = 10933,
5685 ["scnE"] = 10934,
5686 ["succneqq"] = 10934,
5687 ["prap"] = 10935,
5688 ["precapprox"] = 10935,
5689 ["scap"] = 10936,
5690 ["succapprox"] = 10936,
5691 ["precnapprox"] = 10937,
5692 ["prnap"] = 10937,
5693 ["scnap"] = 10938,
5694 ["succnapprox"] = 10938,
5695 ["Pr"] = 10939,
5696 ["Sc"] = 10940,
5697 ["subdot"] = 10941,
5698 ["supdot"] = 10942,
5699 ["subplus"] = 10943,
5700 ["supplus"] = 10944,
5701 ["submult"] = 10945,
5702 ["supmult"] = 10946,

```

```

5703 ["subedot"] = 10947,
5704 ["supedot"] = 10948,
5705 ["nsubE"] = {10949, 824},
5706 ["nsubseteqq"] = {10949, 824},
5707 ["subE"] = 10949,
5708 ["subseteqq"] = 10949,
5709 ["nsupE"] = {10950, 824},
5710 ["nsupseteqq"] = {10950, 824},
5711 ["supE"] = 10950,
5712 ["supseteqq"] = 10950,
5713 ["subsim"] = 10951,
5714 ["supsim"] = 10952,
5715 ["subnE"] = 10955,
5716 ["subsetneqq"] = 10955,
5717 ["varsubsetneqq"] = {10955, 65024},
5718 ["vsubnE"] = {10955, 65024},
5719 ["supnE"] = 10956,
5720 ["supsetneqq"] = 10956,
5721 ["varsupsetneqq"] = {10956, 65024},
5722 ["vsupnE"] = {10956, 65024},
5723 ["csub"] = 10959,
5724 ["csup"] = 10960,
5725 ["csube"] = 10961,
5726 ["csupe"] = 10962,
5727 ["subsup"] = 10963,
5728 ["supsub"] = 10964,
5729 ["subsub"] = 10965,
5730 ["supsup"] = 10966,
5731 ["suphsub"] = 10967,
5732 ["supdsub"] = 10968,
5733 ["forkv"] = 10969,
5734 ["topfork"] = 10970,
5735 ["mlcp"] = 10971,
5736 ["Dashv"] = 10980,
5737 ["DoubleLeftTee"] = 10980,
5738 ["Vdashl"] = 10982,
5739 ["Barv"] = 10983,
5740 ["vBar"] = 10984,
5741 ["vBarv"] = 10985,
5742 ["Vbar"] = 10987,
5743 ["Not"] = 10988,
5744 ["bNot"] = 10989,
5745 ["rnmid"] = 10990,
5746 ["cirmid"] = 10991,
5747 ["midcir"] = 10992,
5748 ["topcir"] = 10993,
5749 ["nhpar"] = 10994,

```

```

5750 ["parsim"] = 10995,
5751 ["nparsl"] = {11005, 8421},
5752 ["parsl"] = 11005,
5753 ["fflig"] = 64256,
5754 ["filig"] = 64257,
5755 ["fllig"] = 64258,
5756 ["ffilig"] = 64259,
5757 ["ffllig"] = 64260,
5758 ["Ascr"] = 119964,
5759 ["Cscr"] = 119966,
5760 ["Dscr"] = 119967,
5761 ["Gscr"] = 119970,
5762 ["Jscr"] = 119973,
5763 ["Kscr"] = 119974,
5764 ["Nscr"] = 119977,
5765 ["Oscr"] = 119978,
5766 ["Pscr"] = 119979,
5767 ["Qscr"] = 119980,
5768 ["Sscr"] = 119982,
5769 ["Tscr"] = 119983,
5770 ["Uscr"] = 119984,
5771 ["Vscr"] = 119985,
5772 ["Wscr"] = 119986,
5773 ["Xscr"] = 119987,
5774 ["Yscr"] = 119988,
5775 ["Zscr"] = 119989,
5776 ["ascr"] = 119990,
5777 ["bscr"] = 119991,
5778 ["cscr"] = 119992,
5779 ["dscr"] = 119993,
5780 ["fscr"] = 119995,
5781 ["hscr"] = 119997,
5782 ["iscr"] = 119998,
5783 ["jscr"] = 119999,
5784 ["kscr"] = 120000,
5785 ["lscr"] = 120001,
5786 ["mscr"] = 120002,
5787 ["nscr"] = 120003,
5788 ["pscr"] = 120005,
5789 ["qscr"] = 120006,
5790 ["rscr"] = 120007,
5791 ["sscr"] = 120008,
5792 ["tscr"] = 120009,
5793 ["uscr"] = 120010,
5794 ["vscr"] = 120011,
5795 ["wscr"] = 120012,
5796 ["xscr"] = 120013,

```

```

5797 ["yscr"] = 120014,
5798 ["zscr"] = 120015,
5799 ["Afr"] = 120068,
5800 ["Bfr"] = 120069,
5801 ["Dfr"] = 120071,
5802 ["Efr"] = 120072,
5803 ["Ffr"] = 120073,
5804 ["Gfr"] = 120074,
5805 ["Jfr"] = 120077,
5806 ["Kfr"] = 120078,
5807 ["Lfr"] = 120079,
5808 ["Mfr"] = 120080,
5809 ["Nfr"] = 120081,
5810 ["Ofr"] = 120082,
5811 ["Pfr"] = 120083,
5812 ["Qfr"] = 120084,
5813 ["Sfr"] = 120086,
5814 ["Tfr"] = 120087,
5815 ["Ufr"] = 120088,
5816 ["Vfr"] = 120089,
5817 ["Wfr"] = 120090,
5818 ["Xfr"] = 120091,
5819 ["Yfr"] = 120092,
5820 ["afr"] = 120094,
5821 ["bfr"] = 120095,
5822 ["cfr"] = 120096,
5823 ["dfr"] = 120097,
5824 ["efr"] = 120098,
5825 ["ffr"] = 120099,
5826 ["gfr"] = 120100,
5827 ["hfr"] = 120101,
5828 ["ifr"] = 120102,
5829 ["jfr"] = 120103,
5830 ["kfr"] = 120104,
5831 ["lfr"] = 120105,
5832 ["mfr"] = 120106,
5833 ["nfr"] = 120107,
5834 ["ofr"] = 120108,
5835 ["pfr"] = 120109,
5836 ["qfr"] = 120110,
5837 ["rfr"] = 120111,
5838 ["sfr"] = 120112,
5839 ["tfr"] = 120113,
5840 ["ufr"] = 120114,
5841 ["vfr"] = 120115,
5842 ["wfr"] = 120116,
5843 ["xfr"] = 120117,

```

```

5844 ["yfr"] = 120118,
5845 ["zfr"] = 120119,
5846 ["Aopf"] = 120120,
5847 ["Bopf"] = 120121,
5848 ["Dopf"] = 120123,
5849 ["Eopf"] = 120124,
5850 ["Fopf"] = 120125,
5851 ["Gopf"] = 120126,
5852 ["Iopf"] = 120128,
5853 ["Jopf"] = 120129,
5854 ["Kopf"] = 120130,
5855 ["Lopf"] = 120131,
5856 ["Mopf"] = 120132,
5857 ["Oopf"] = 120134,
5858 ["Sopf"] = 120138,
5859 ["Topf"] = 120139,
5860 ["Uopf"] = 120140,
5861 ["Vopf"] = 120141,
5862 ["Wopf"] = 120142,
5863 ["Xopf"] = 120143,
5864 ["Yopf"] = 120144,
5865 ["aopf"] = 120146,
5866 ["bopf"] = 120147,
5867 ["copf"] = 120148,
5868 ["dopf"] = 120149,
5869 ["eopf"] = 120150,
5870 ["fopf"] = 120151,
5871 ["gopf"] = 120152,
5872 ["hopf"] = 120153,
5873 ["iopf"] = 120154,
5874 ["jopf"] = 120155,
5875 ["kopf"] = 120156,
5876 ["lopf"] = 120157,
5877 ["mopf"] = 120158,
5878 ["nopf"] = 120159,
5879 ["oopf"] = 120160,
5880 ["popf"] = 120161,
5881 ["qopf"] = 120162,
5882 ["ropf"] = 120163,
5883 ["sopf"] = 120164,
5884 ["topf"] = 120165,
5885 ["uopf"] = 120166,
5886 ["vopf"] = 120167,
5887 ["wopf"] = 120168,
5888 ["xopf"] = 120169,
5889 ["yopf"] = 120170,
5890 ["zopf"] = 120171,

```

```
5891 }
```

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

```
5892 function entities.dec_entity(s)
5893 local n = tonumber(s)
5894 if n == nil then
5895 return "&#" .. s .. ";" -- fallback for unknown entities
5896 end
5897 return unicode.utf8.char(n)
5898 end
```

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

```
5899 function entities.hex_entity(s)
5900 local n = tonumber("0x"..s)
5901 if n == nil then
5902 return "&#x" .. s .. ";" -- fallback for unknown entities
5903 end
5904 return unicode.utf8.char(n)
5905 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.

```
5906 function entities.hex_entity_with_x_char(x, s)
5907 local n = tonumber("0x"..s)
5908 if n == nil then
5909 return "&#" .. x .. s .. ";" -- fallback for unknown entities
5910 end
5911 return unicode.utf8.char(n)
5912 end
```

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

```
5913 function entities.char_entity(s)
5914 local code_points = character_entities[s]
5915 if code_points == nil then
5916 return "&" .. s .. ";"
5917 end
5918 if type(code_points) ~= 'table' then
5919 code_points = {code_points}
5920 end
5921 local char_table = {}
5922 for _, code_point in ipairs(code_points) do
5923 table.insert(char_table, unicode.utf8.char(code_point))
5924 end
5925 return table.concat(char_table)
```

5926 end

### 3.1.3 Plain T<sub>E</sub>X Writer

This section documents the `writer` object, which implements the routines for producing the T<sub>E</sub>X output. The object is an amalgamate of the generic, T<sub>E</sub>X, L<sup>A</sup>T<sub>E</sub>X writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the 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.

```
5927 M.writer = {}
```

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

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

```
5928 function M.writer.new(options)
5929 local self = {}
```

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

```
5930 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.

```
5931 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.

```
5932 local slice_specifiers = {}
5933 for specifier in options.slice:gmatch("[^%s]+") do
5934 table.insert(slice_specifiers, specifier)
5935 end
5936
5937 if #slice_specifiers == 2 then
5938 self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5939 local slice_begin_type = self.slice_begin:sub(1, 1)
5940 if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5941 self.slice_begin = "^" .. self.slice_begin
```



```

5942 end
5943 local slice_end_type = self.slice_end:sub(1, 1)
5944 if slice_end_type ~= "^" and slice_end_type ~= "$" then
5945 self.slice_end = "$" .. self.slice_end
5946 end
5947 elseif #slice_specifiers == 1 then
5948 self.slice_begin = "^" .. slice_specifiers[1]
5949 self.slice_end = "$" .. slice_specifiers[1]
5950 end
5951
5952 self.slice_begin_type = self.slice_begin:sub(1, 1)
5953 self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5954 self.slice_end_type = self.slice_end:sub(1, 1)
5955 self.slice_end_identifier = self.slice_end:sub(2) or ""
5956
5957 if self.slice_begin == "^" and self.slice_end ~= "^" then
5958 self.is_writing = true
5959 else
5960 self.is_writing = false
5961 end

```

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

```

5962 self.space = " "

```

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

```

5963 self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

5964 function self.plain(s)
5965 return s
5966 end

```

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

```

5967 function self.paragraph(s)
5968 if not self.is_writing then return "" end
5969 return s
5970 end

```

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

```

5971 self.interblocksep_text = "\\markdownRendererInterblockSeparator\\n{}"
5972 function self.interblocksep()
5973 if not self.is_writing then return "" end
5974 return self.interblocksep_text
5975 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.

```
5976 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
5977 function self.paragraphsep()
5978 if not self.is_writing then return "" end
5979 return self.paragraphsep_text
5980 end
```

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

```
5981 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
5982 function self.undosep()
5983 if not self.is_writing then return "" end
5984 return self.undosep_text
5985 end
```

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

```
5986 self.soft_line_break = function()
5987 if self.flatten_inlines then return "\n" end
5988 return "\\markdownRendererSoftLineBreak\n{"
5989 end
```

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

```
5990 self.hard_line_break = function()
5991 if self.flatten_inlines then return "\n" end
5992 return "\\markdownRendererHardLineBreak\n{"
5993 end
```

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

```
5994 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
5995 function self.thematic_break()
5996 if not self.is_writing then return "" end
5997 return "\\markdownRendererThematicBreak{"
5998 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.

```
5999 self.escaped_uri_chars = {
6000 [{""] = "\\markdownRendererLeftBrace{",
6001 ["}"] = "\\markdownRendererRightBrace{",
6002 ["\\"] = "\\markdownRendererBackslash{",
6003 ["\r"] = " ",
6004 ["\n"] = " ",
6005 }
6006 self.escaped_minimal_strings = {
6007 ["^^"] = "\\markdownRendererCircumflex"
```

```

6008 .. "\\markdownRendererCircumflex ",
6009 ["☒"] = "\\markdownRendererTickedBox{}",
6010 ["☐"] = "\\markdownRendererHalfTickedBox{}",
6011 ["□"] = "\\markdownRendererUntickedBox{}",
6012 [entities.hex_entity('FFFD')]
6013 = "\\markdownRendererReplacementCharacter{}",
6014 }

```

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

```

6015 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
6016 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.

```

6017 self.escaped_chars = {
6018 ["{"] = "\\markdownRendererLeftBrace{}",
6019 ["}"] = "\\markdownRendererRightBrace{}",
6020 ["%"] = "\\markdownRendererPercentSign{}",
6021 ["\\"] = "\\markdownRendererBackslash{}",
6022 ["#"] = "\\markdownRendererHash{}",
6023 ["$"] = "\\markdownRendererDollarSign{}",
6024 ["&"] = "\\markdownRendererAmpersand{}",
6025 ["_"] = "\\markdownRendererUnderscore{}",
6026 ["^"] = "\\markdownRendererCircumflex{}",
6027 ["~"] = "\\markdownRendererTilde{}",
6028 ["|"] = "\\markdownRendererPipe{}",
6029 [entities.hex_entity('0000')]
6030 = "\\markdownRendererReplacementCharacter{}",
6031 }

```

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

```

6032 local function create_escaper(char_escapes, string_escapes)
6033 local escape = util.escaper(char_escapes, string_escapes)
6034 return function(s)
6035 if self.flatten_inlines then return s end
6036 return escape(s)
6037 end
6038 end
6039 local escape_typographic_text = create_escaper(
6040 self.escaped_chars, self.escaped_strings)
6041 local escape_programmatic_text = create_escaper(
6042 self.escaped_uri_chars, self.escaped_minimal_strings)
6043 local escape_minimal = create_escaper(
6044 {}, 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.

```
6045 self.escape = escape_typographic_text
6046 self.math = escape_minimal
6047 if options.hybrid then
6048 self.identifier = escape_minimal
6049 self.string = escape_minimal
6050 self.uri = escape_minimal
6051 self.infostring = escape_minimal
6052 else
6053 self.identifier = escape_programmatic_text
6054 self.string = escape_typographic_text
6055 self.uri = escape_programmatic_text
6056 self.infostring = escape_programmatic_text
6057 end
```

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

```
6058 function self.warning(t, m)
6059 return {"\\markdownRendererWarning{", self.escape(t), "}{" ,
6060 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6061 escape_minimal(m or ""), "}"}
6062 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.

```
6063 function self.error(t, m)
6064 return {"\\markdownRendererError{", self.escape(t), "}{" ,
6065 escape_minimal(t), "}{" , self.escape(m or ""), "}{" ,
6066 escape_minimal(m or ""), "}"}
6067 end
```

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

```
6068 function self.code(s, attributes)
6069 if self.flatten_inlines then return s end
6070 local buf = {}
6071 if attributes ~= nil then
```

```

6072 table.insert(buf,
6073 "\\markdownRendererCodeSpanAttributeContextBegin\n")
6074 table.insert(buf, self.attributes(attributes))
6075 end
6076 table.insert(buf,
6077 {"\\markdownRendererCodeSpan{", self.escape(s), "}"}))
6078 if attributes ~= nil then
6079 table.insert(buf,
6080 "\\markdownRendererCodeSpanAttributeContextEnd{")
6081 end
6082 return buf
6083 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.

```

6084 function self.link(lab, src, tit, attributes)
6085 if self.flatten_inlines then return lab end
6086 local buf = {}
6087 if attributes ~= nil then
6088 table.insert(buf,
6089 "\\markdownRendererLinkAttributeContextBegin\n")
6090 table.insert(buf, self.attributes(attributes))
6091 end
6092 table.insert(buf, {"\\markdownRendererLink{", lab, "}",
6093 "{", self.escape(src), "}",
6094 "{", self.uri(src), "}",
6095 "{", self.string(tit or ""), "}"})
6096 if attributes ~= nil then
6097 table.insert(buf,
6098 "\\markdownRendererLinkAttributeContextEnd{")
6099 end
6100 return buf
6101 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.

```

6102 function self.image(lab, src, tit, attributes)
6103 if self.flatten_inlines then return lab end
6104 local buf = {}
6105 if attributes ~= nil then
6106 table.insert(buf,
6107 "\\markdownRendererImageAttributeContextBegin\n")
6108 table.insert(buf, self.attributes(attributes))
6109 end
6110 table.insert(buf, {"\\markdownRendererImage{", lab, "}",

```

```

6111 {"",self.string(src),""},
6112 {"",self.uri(src),""},
6113 {"",self.string(tit or ""),""})
6114 if attributes ~= nil then
6115 table.insert(buf,
6116 "\\markdownRendererImageAttributeContextEnd{}")
6117 end
6118 return buf
6119 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.

```

6120 function self.bulletlist(items,tight)
6121 if not self.is_writing then return "" end
6122 local buffer = {}
6123 for _,item in ipairs(items) do
6124 if item ~= "" then
6125 buffer[#buffer + 1] = self.bulletitem(item)
6126 end
6127 end
6128 local contents = util.intersperse(buffer,"\n")
6129 if tight and options.tightLists then
6130 return {"\\markdownRendererUlBeginTight\n",contents,
6131 "\n\\markdownRendererUlEndTight "}
6132 else
6133 return {"\\markdownRendererUlBegin\n",contents,
6134 "\n\\markdownRendererUlEnd "}
6135 end
6136 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.

```

6137 function self.bulletitem(s)
6138 return {"\\markdownRendererUlItem ",s,
6139 "\\markdownRendererUlItemEnd "}
6140 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.

```

6141 function self.orderedlist(items,tight,startnum)
6142 if not self.is_writing then return "" end
6143 local buffer = {}
6144 local num = startnum
6145 for _,item in ipairs(items) do
6146 if item ~= "" then

```

```

6147 buffer[#buffer + 1] = self.ordereditem(item,num)
6148 end
6149 if num ~= nil and item ~= "" then
6150 num = num + 1
6151 end
6152 end
6153 local contents = util.intersperse(buffer,"\n")
6154 if tight and options.tightLists then
6155 return {"\\markdownRenderer01BeginTight\n",contents,
6156 "\\n\\markdownRenderer01EndTight "}
6157 else
6158 return {"\\markdownRenderer01Begin\n",contents,
6159 "\\n\\markdownRenderer01End "}
6160 end
6161 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.

```

6162 function self.ordereditem(s,num)
6163 if num ~= nil then
6164 return {"\\markdownRenderer01ItemWithNumber{",num,"}",s,
6165 "\\n\\markdownRenderer01ItemEnd "}
6166 else
6167 return {"\\markdownRenderer01Item ",s,
6168 "\\n\\markdownRenderer01ItemEnd "}
6169 end
6170 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.

```

6171 function self.inline_html_comment(contents)
6172 if self.flatten_inlines then return contents end
6173 return {"\\markdownRendererInlineHtmlComment{",contents,""}
6174 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.

```

6175 function self.inline_html_tag(contents)
6176 if self.flatten_inlines then return contents end
6177 return {"\\markdownRendererInlineHtmlTag{",
6178 self.string(contents),""}
6179 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.

```
6180 function self.block_html_element(s)
6181 if not self.is_writing then return "" end
6182 local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6183 return {"\\markdownRendererInputBlockHtmlElement{" ,name,""} }
6184 end
```

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

```
6185 function self.emphasis(s)
6186 if self.flatten_inlines then return s end
6187 return {"\\markdownRendererEmphasis{" ,s,""} }
6188 end
```

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

```
6189 function self.tickbox(f)
6190 if f == 1.0 then
6191 return "☒ "
6192 elseif f == 0.0 then
6193 return "☐ "
6194 else
6195 return "◻ "
6196 end
6197 end
```

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

```
6198 function self.strong(s)
6199 if self.flatten_inlines then return s end
6200 return {"\\markdownRendererStrongEmphasis{" ,s,""} }
6201 end
```

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

```
6202 function self.blockquote(s)
6203 if not self.is_writing then return "" end
6204 return {"\\markdownRendererBlockQuoteBegin\\n",s,
6205 "\\markdownRendererBlockQuoteEnd "}
6206 end
```

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

```
6207 function self.verbatim(s)
6208 if not self.is_writing then return "" end
6209 s = s:gsub("\\n$", "")
```



```

6210 local name = util.cache_verbatim(options.cacheDir, s)
6211 return {"\\markdownRendererInputVerbatim{" ,name,""} }
6212 end

```

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

```

6213 function self.document(d)
6214 local buf = {"\\markdownRendererDocumentBegin\n"}
6215
6216 -- warn against the `hybrid` option
6217 if options.hybrid then
6218 local text = "The `hybrid` option has been soft-deprecated."
6219 local more = "Consider using one of the following better options "
6220 .. "for mixing TeX and markdown: `contentBlocks`, "
6221 .. "`rawAttribute`, `texComments`, `texMathDollars`, "
6222 .. "`texMathSingleBackslash`, and "
6223 .. "`texMathDoubleBackslash`. "
6224 .. "For more information, see the user manual at "
6225 .. "<https://witiko.github.io/markdown/>."
6226 table.insert(buf, self.warning(text, more))
6227 end
6228
6229 -- insert the text of the document
6230 table.insert(buf, d)
6231
6232 -- pop all attributes
6233 table.insert(buf, self.pop_attributes())
6234
6235 table.insert(buf, "\\markdownRendererDocumentEnd")
6236
6237 return buf
6238 end

```

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

```

6239 local seen_identifiers = {}
6240 local key_value_regex = "([~=]+)%s*=%s*(.*)"
6241 local function normalize_attributes(attributes, auto_identifiers)
6242 -- normalize attributes
6243 local normalized_attributes = {}
6244 local has_explicit_identifiers = false
6245 local key, value
6246 for _, attribute in ipairs(attributes or {}) do
6247 if attribute:sub(1, 1) == "#" then
6248 table.insert(normalized_attributes, attribute)
6249 has_explicit_identifiers = true
6250 seen_identifiers[attribute:sub(2)] = true
6251 elseif attribute:sub(1, 1) == "." then

```

```

6252 table.insert(normalized_attributes, attribute)
6253 else
6254 key, value = attribute:match(key_value_regex)
6255 if key:lower() == "id" then
6256 table.insert(normalized_attributes, "#" .. value)
6257 elseif key:lower() == "class" then
6258 local classes = {}
6259 for class in value:gmatch("%S+") do
6260 table.insert(classes, class)
6261 end
6262 table.sort(classes)
6263 for _, class in ipairs(classes) do
6264 table.insert(normalized_attributes, "." .. class)
6265 end
6266 else
6267 table.insert(normalized_attributes, attribute)
6268 end
6269 end
6270 end
6271
6272 -- if no explicit identifiers exist, add auto identifiers
6273 if not has_explicit_identifiers and auto_identifiers ~= nil then
6274 local seen_auto_identifiers = {}
6275 for _, auto_identifier in ipairs(auto_identifiers) do
6276 if seen_auto_identifiers[auto_identifier] == nil then
6277 seen_auto_identifiers[auto_identifier] = true
6278 if seen_identifiers[auto_identifier] == nil then
6279 seen_identifiers[auto_identifier] = true
6280 table.insert(normalized_attributes,
6281 "#" .. auto_identifier)
6282 else
6283 local auto_identifier_number = 1
6284 while true do
6285 local numbered_auto_identifier = auto_identifier .. "-"
6286 .. auto_identifier_number
6287 if seen_identifiers[numbered_auto_identifier] == nil then
6288 seen_identifiers[numbered_auto_identifier] = true
6289 table.insert(normalized_attributes,
6290 "#" .. numbered_auto_identifier)
6291 break
6292 end
6293 auto_identifier_number = auto_identifier_number + 1
6294 end
6295 end
6296 end
6297 end
6298 end

```

```

6299
6300 -- sort and deduplicate normalized attributes
6301 table.sort(normalized_attributes)
6302 local seen_normalized_attributes = {}
6303 local deduplicated_normalized_attributes = {}
6304 for _, attribute in ipairs(normalized_attributes) do
6305 if seen_normalized_attributes[attribute] == nil then
6306 seen_normalized_attributes[attribute] = true
6307 table.insert(deduplicated_normalized_attributes, attribute)
6308 end
6309 end
6310
6311 return deduplicated_normalized_attributes
6312 end
6313
6314 function self.attributes(attributes, should_normalize_attributes)
6315 local normalized_attributes
6316 if should_normalize_attributes == false then
6317 normalized_attributes = attributes
6318 else
6319 normalized_attributes = normalize_attributes(attributes)
6320 end
6321
6322 local buf = {}
6323 local key, value
6324 for _, attribute in ipairs(normalized_attributes) do
6325 if attribute:sub(1, 1) == "#" then
6326 table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
6327 attribute:sub(2), "}"}))
6328 elseif attribute:sub(1, 1) == "." then
6329 table.insert(buf, {"\\markdownRendererAttributeName{",
6330 attribute:sub(2), "}"}))
6331 else
6332 key, value = attribute:match(key_value_regex)
6333 table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
6334 key, "}{" , value, "}"}))
6335 end
6336 end
6337
6338 return buf
6339 end

```

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

```

6340 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`.

```

6341 self.attribute_type_levels = {}
6342 setmetatable(self.attribute_type_levels,
6343 { __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`.

```

6344 local function apply_attributes()
6345 local buf = {}
6346 for i = 1, #self.active_attributes do
6347 local start_output = self.active_attributes[i][3]
6348 if start_output ~= nil then
6349 table.insert(buf, start_output)
6350 end
6351 end
6352 return buf
6353 end
6354
6355 local function tear_down_attributes()
6356 local buf = {}
6357 for i = #self.active_attributes, 1, -1 do
6358 local end_output = self.active_attributes[i][4]
6359 if end_output ~= nil then
6360 table.insert(buf, end_output)
6361 end
6362 end
6363 return buf
6364 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.

```

6365 function self.push_attributes(attribute_type, attributes,
6366 start_output, end_output)
6367 local attribute_type_level
6368 = self.attribute_type_levels[attribute_type]
6369 self.attribute_type_levels[attribute_type]
6370 = attribute_type_level + 1
6371
6372 -- index attributes in a hash table for easy lookup
6373 attributes = attributes or {}
6374 for i = 1, #attributes do
6375 attributes[attributes[i]] = true
6376 end
6377

```

```

6378 local buf = {}
6379 -- handle slicing
6380 if attributes["#" .. self.slice_end_identifier] ~= nil and
6381 self.slice_end_type == "^" then
6382 if self.is_writing then
6383 table.insert(buf, self.undosep())
6384 table.insert(buf, tear_down_attributes())
6385 end
6386 self.is_writing = false
6387 end
6388 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6389 self.slice_begin_type == "^" then
6390 table.insert(buf, apply_attributes())
6391 self.is_writing = true
6392 end
6393 if self.is_writing and start_output ~= nil then
6394 table.insert(buf, start_output)
6395 end
6396 table.insert(self.active_attributes,
6397 {attribute_type, attributes,
6398 start_output, end_output})
6399 return buf
6400 end
6401

```

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`).

```

6402 function self.pop_attributes(attribute_type)
6403 local buf = {}
6404 -- pop attributes until we find attributes of correct type
6405 -- or until no attributes remain
6406 local current_attribute_type = false
6407 while current_attribute_type ~= attribute_type and
6408 #self.active_attributes > 0 do
6409 local attributes, _, end_output
6410 current_attribute_type, attributes, _, end_output = table.unpack(
6411 self.active_attributes[#self.active_attributes])
6412 local attribute_type_level
6413 = self.attribute_type_levels[current_attribute_type]
6414 self.attribute_type_levels[current_attribute_type]
6415 = attribute_type_level - 1
6416 if self.is_writing and end_output ~= nil then
6417 table.insert(buf, end_output)

```

```

6418 end
6419 table.remove(self.active_attributes, #self.active_attributes)
6420 -- handle slicing
6421 if attributes["#" .. self.slice_end_identifier] ~= nil
6422 and self.slice_end_type == "$" then
6423 if self.is_writing then
6424 table.insert(buf, self.undosep())
6425 table.insert(buf, tear_down_attributes())
6426 end
6427 self.is_writing = false
6428 end
6429 if attributes["#" .. self.slice_begin_identifier] ~= nil and
6430 self.slice_begin_type == "$" then
6431 self.is_writing = true
6432 table.insert(buf, apply_attributes())
6433 end
6434 end
6435 return buf
6436 end

```

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

```

6437 local function create_auto_identifier(s)
6438 local buffer = {}
6439 local prev_space = false
6440 local letter_found = false
6441 local normalized_s = s
6442 if not options.unicodeNormalization
6443 or options.unicodeNormalizationForm ~= "nfc" then
6444 normalized_s = uni_algos.normalize.NFC(normalized_s)
6445 end
6446
6447 for _, code in utf8.codes(normalized_s) do
6448 local char = utf8.char(code)
6449
6450 -- Remove everything up to the first letter.
6451 if not letter_found then
6452 local is_letter = unicode.utf8.match(char, "%a")
6453 if is_letter then
6454 letter_found = true
6455 else
6456 goto continue
6457 end
6458 end
6459
6460 -- Remove all non-alphanumeric characters, except underscores,
6461 -- hyphens, and periods.
6462 if not unicode.utf8.match(char, "[%w_-%.%s]") then
6463 goto continue
6464 end
6465 end

```

```

6464 end
6465
6466 -- Replace all spaces and newlines with hyphens.
6467 if unicode.utf8.match(char, "[%s\\n]") then
6468 char = "-"
6469 if prev_space then
6470 goto continue
6471 else
6472 prev_space = true
6473 end
6474 else
6475 -- Convert all alphabetic characters to lowercase.
6476 char = unicode.utf8.lower(char)
6477 prev_space = false
6478 end
6479
6480 table.insert(buffer, char)
6481
6482 ::continue::
6483 end
6484
6485 if prev_space then
6486 table.remove(buffer)
6487 end
6488
6489 local identifier = #buffer == 0 and "section"
6490 or table.concat(buffer, "")
6491 return identifier
6492 end

```

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

```

6493 local function create_gfm_auto_identifier(s)
6494 local buffer = {}
6495 local prev_space = false
6496 local letter_found = false
6497 local normalized_s = s
6498 if not options.unicodeNormalization
6499 or options.unicodeNormalizationForm ~= "nfc" then
6500 normalized_s = uni_algos.normalize.NFC(normalized_s)
6501 end
6502
6503 for _, code in utf8.codes(normalized_s) do
6504 local char = utf8.char(code)
6505
6506 -- Remove everything up to the first non-space.
6507 if not letter_found then

```

```

6508 local is_letter = unicode.utf8.match(char, "%S")
6509 if is_letter then
6510 letter_found = true
6511 else
6512 goto continue
6513 end
6514 end
6515
6516 -- Remove all non-alphanumeric characters, except underscores
6517 -- and hyphens.
6518 if not unicode.utf8.match(char, "[%w_-%s]") then
6519 prev_space = false
6520 goto continue
6521 end
6522
6523 -- Replace all spaces and newlines with hyphens.
6524 if unicode.utf8.match(char, "[%s\\n]") then
6525 char = "-"
6526 if prev_space then
6527 goto continue
6528 else
6529 prev_space = true
6530 end
6531 else
6532 -- Convert all alphabetic characters to lowercase.
6533 char = unicode.utf8.lower(char)
6534 prev_space = false
6535 end
6536
6537 table.insert(buffer, char)
6538
6539 ::continue::
6540 end
6541
6542 if prev_space then
6543 table.remove(buffer)
6544 end
6545
6546 local identifier = #buffer == 0 and "section"
6547 or table.concat(buffer, "")
6548 return identifier
6549 end

```

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

```

6550 self.secbegin_text = "\\markdownRendererSectionBegin\\n"
6551 self.seccend_text = "\\n\\markdownRendererSectionEnd "

```



```

6552 function self.heading(s, level, attributes)
6553 local buf = {}
6554 local flat_text, inlines = table.unpack(s)
6555
6556 -- push empty attributes for implied sections
6557 while self.attribute_type_levels["heading"] < level - 1 do
6558 table.insert(buf,
6559 self.push_attributes("heading",
6560 nil,
6561 self.secbegin_text,
6562 self.secend_text))
6563 end
6564
6565 -- pop attributes for sections that have ended
6566 while self.attribute_type_levels["heading"] >= level do
6567 table.insert(buf, self.pop_attributes("heading"))
6568 end
6569
6570 -- construct attributes for the new section
6571 local auto_identifiers = {}
6572 if self.options.autoIdentifiers then
6573 table.insert(auto_identifiers, create_auto_identifier(flat_text))
6574 end
6575 if self.options.gfmAutoIdentifiers then
6576 table.insert(auto_identifiers,
6577 create_gfm_auto_identifier(flat_text))
6578 end
6579 local normalized_attributes = normalize_attributes(attributes,
6580 auto_identifiers)
6581
6582 -- push attributes for the new section
6583 local start_output = {}
6584 local end_output = {}
6585 table.insert(start_output, self.secbegin_text)
6586 table.insert(end_output, self.secend_text)
6587
6588 table.insert(buf, self.push_attributes("heading",
6589 normalized_attributes,
6590 start_output,
6591 end_output))
6592 assert(self.attribute_type_levels["heading"] == level)
6593
6594 -- render the heading and its attributes
6595 if self.is_writing and #normalized_attributes > 0 then
6596 table.insert(buf,
6597 "\\markdownRendererHeaderAttributeContextBegin\\n")
6598 table.insert(buf, self.attributes(normalized_attributes, false))

```

```

6599 end
6600
6601 local cmd
6602 level = level + options.shiftHeadings
6603 if level <= 1 then
6604 cmd = "\\markdownRendererHeadingOne"
6605 elseif level == 2 then
6606 cmd = "\\markdownRendererHeadingTwo"
6607 elseif level == 3 then
6608 cmd = "\\markdownRendererHeadingThree"
6609 elseif level == 4 then
6610 cmd = "\\markdownRendererHeadingFour"
6611 elseif level == 5 then
6612 cmd = "\\markdownRendererHeadingFive"
6613 elseif level >= 6 then
6614 cmd = "\\markdownRendererHeadingSix"
6615 else
6616 cmd = ""
6617 end
6618 if self.is_writing then
6619 table.insert(buf, {cmd, "{", inlines, "}"})
6620 end
6621
6622 if self.is_writing and #normalized_attributes > 0 then
6623 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6624 end
6625
6626 return buf
6627 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.

```

6628 function self.get_state()
6629 return {
6630 is_writing=self.is_writing,
6631 flatten_inlines=self.flatten_inlines,
6632 active_attributes={table.unpack(self.active_attributes)},
6633 }
6634 end

```

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

```

6635 function self.set_state(s)
6636 local previous_state = self.get_state()
6637 for key, value in pairs(s) do
6638 self[key] = value
6639 end
6640 return previous_state

```

```
6641 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`.

```
6642 function self.defer_call(f)
6643 local previous_state = self.get_state()
6644 return function(...)
6645 local state = self.set_state(previous_state)
6646 local return_value = f(...)
6647 self.set_state(state)
6648 return return_value
6649 end
6650 end
6651
6652 return self
6653 end
```

### 3.1.4 Parsers

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

```
6654 local parsers = {}
```

#### 3.1.4.1 Basic Parsers

```
6655 parsers.percent = P("%")
6656 parsers.at = P("@")
6657 parsers.comma = P(",")
6658 parsers.asterisk = P("*")
6659 parsers.dash = P("-")
6660 parsers.plus = P("+")
6661 parsers.underscore = P("_")
6662 parsers.period = P(".")
6663 parsers.hash = P("#")
6664 parsers.dollar = P("$")
6665 parsers.ampersand = P("&")
6666 parsers.backtick = P("`")
6667 parsers.less = P("<")
6668 parsers.more = P(">")
6669 parsers.space = P(" ")
6670 parsers.squote = P("'")
6671 parsers.dquote = P('"')
6672 parsers.lparent = P("(")
6673 parsers.rparent = P(")")
6674 parsers.lbracket = P("[")
6675 parsers.rbracket = P("]")
```

```

6676 parsers.lbrace = P("{")
6677 parsers.rbrace = P("}")
6678 parsers.circumflex = P("^")
6679 parsers.slash = P("/")
6680 parsers.equal = P("=")
6681 parsers.colon = P(":")
6682 parsers.semicolon = P(";")
6683 parsers.exclamation = P("!")
6684 parsers.pipe = P("|")
6685 parsers.tilde = P("~")
6686 parsers.backslash = P("\\")
6687 parsers.tab = P("\t")
6688 parsers.newline = P("\n")
6689
6690 parsers.digit = R("09")
6691 parsers.hexdigit = R("09","af","AF")
6692 parsers.letter = R("AZ","az")
6693 parsers.alphanumeric = R("AZ","az","09")
6694 parsers.keyword = parsers.letter
6695 * (parsers.alphanumeric + parsers.dash)^0
6696
6697 parsers.doubleasterisks = P("**")
6698 parsers.doubleunderscores = P("__")
6699 parsers.doubletildes = P("~")
6700 parsers.fourspace = P(" ")
6701
6702 parsers.any = P(1)
6703 parsers.succeed = P(true)
6704 parsers.fail = P(false)
6705
6706 parsers.internal_punctuation = S(",:;.?.")
6707 parsers.ascii_punctuation = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
6708

```

### 3.1.5 Unicode punctuation

This section documents the Unicode punctuation<sup>33</sup> recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

(CommonMark Spec, Version 0.31.2 (2024-01-28))

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

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.

```

6709 ;(function()
6710 local pathname = assert(kpse.find_file("UnicodeData.txt"),
6711 [[Could not locate file "UnicodeData.txt"]])
6712 local file = assert(io.open(pathname, "r"),
6713 [[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.

```

6714 local prefix_trees = {}
6715 for line in file:lines() do
6716 local codepoint, major_category = line:match("^(%x+);[^\s]*;(%a)")
6717 if major_category == "P" or major_category == "S" then
6718 local code = unicode.utf8.char(tonumber(codepoint, 16))
6719 if prefix_trees[#code] == nil then
6720 prefix_trees[#code] = {}
6721 end
6722 local node = prefix_trees[#code]
6723 for i = 1, #code do
6724 local byte = code:sub(i, i)
6725 if i < #code then
6726 if node[byte] == nil then
6727 node[byte] = {}
6728 end
6729 node = node[byte]
6730 else
6731 table.insert(node, byte)
6732 end
6733 end
6734 end
6735 end
6736 assert(file:close())
6737

```

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

```

6738 local function depth_first_search(node, path, visit, leave)
6739 visit(node, path)
6740 for label, child in pairs(node) do
6741 if type(child) == "table" then
6742 depth_first_search(child, path .. label, visit, leave)
6743 else
6744 visit(child, path)
6745 end
6746 end
6747 leave(node, path)

```

```

6748 end
6749
6750 print("M.punctuation = {}")
6751 print("local S = lpeg.S")
6752 print("-- luacheck: push no max line length")
6753 for length, prefix_tree in pairs(prefix_trees) do
6754 local subparsers = {}
6755 depth_first_search(prefix_tree, "", function(node, path)
6756 if type(node) == "string" then
6757 local suffix
6758 if node == "]" then
6759 suffix = "S('" .. node .. "')"
6760 else
6761 suffix = "S([" .. node .. "])"
6762 end
6763 if subparsers[path] ~= nil then
6764 subparsers[path] = subparsers[path] .. " + " .. suffix
6765 else
6766 subparsers[path] = suffix
6767 end
6768 end
6769 end, function(_, path)
6770 if #path > 0 then
6771 local byte = path:sub(#path, #path)
6772 local parent_path = path:sub(1, #path-1)
6773 if subparsers[path] ~= nil then
6774 local suffix
6775 if byte == "]" then
6776 suffix = "S('" .. byte .. "')"
6777 else
6778 suffix = "S([" .. byte .. "])"
6779 end
6780 suffix = suffix .. " * (" .. subparsers[path] .. ")"
6781 if subparsers[parent_path] ~= nil then
6782 subparsers[parent_path] = subparsers[parent_path]
6783 .. " + " .. suffix
6784 else
6785 subparsers[parent_path] = suffix
6786 end
6787 end
6788 else
6789 print("M.punctuation[" .. length .. "] = " .. subparsers[path])
6790 end
6791 end)
6792 end
6793 print("-- luacheck: pop")
6794 end)()

```

```
6795 print("return M")
```

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

```
6796 local unicode_data = require("markdown-unicode-data")
6797 if metadata.version ~= unicode_data.metadata.version then
6798 util.warning(
6799 "markdown.lua " .. metadata.version .. " used with " ..
6800 "markdown-unicode-data.lua " .. unicode_data.metadata.version .. ".")
6801)
6802 end
6803 parsers.punctuation = unicode_data.punctuation
6804
6805 parsers.escapable = parsers.ascii_punctuation
6806 parsers.anyescaped = parsers.backslash / ""
6807 * parsers.escapable
6808 + parsers.any
6809
6810 parsers.spacechar = S("\t ")
6811 parsers.spacing = S(" \n\r\t")
6812 parsers.nonspacechar = parsers.any - parsers.spacing
6813 parsers.optionalspace = parsers.spacechar^0
6814
6815 parsers.normalchar = parsers.any - (V("SpecialChar")
6816 + parsers.spacing)
6817 parsers.eof = -parsers.any
6818 parsers.nonindentSPACE = parsers.space^-3 * - parsers.spacechar
6819 parsers.indent = parsers.space^-3 * parsers.tab
6820 + parsers.fourspace / ""
6821 parsers.linechar = P(1 - parsers.newline)
6822
6823 parsers.blankline = parsers.optionalspace
6824 * parsers.newline / "\n"
6825 parsers.blanklines = parsers.blankline^0
6826 parsers.skipblanklines = (parsers.optionalspace
6827 * parsers.newline)^0
6828 parsers.indentedline = parsers.indent / ""
6829 * C(parsers.linechar^1
6830 * parsers.newline^1)
6831 parsers.optionallyindentedline = parsers.indent^-1 / ""
6832 * C(parsers.linechar^1
6833 * parsers.newline^1)
6834 parsers.sp = parsers.spacing^0
6835 parsers.spnl = parsers.optionalspace
6836 * (parsers.newline
6837 * parsers.optionalspace)^1
6838 parsers.line = parsers.linechar^0 * parsers.newline
```

```
6839 parsers.nonemptyline = parsers.line - parsers.blankline
```

### 3.1.5.1 Parsers Used for Indentation

```
6840
6841 parsers.leader = parsers.space^-3
6842
```

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

```
6843 local function has_trail(indent_table)
6844 return indent_table ~= nil and
6845 indent_table.trail ~= nil and
6846 next(indent_table.trail) ~= nil
6847 end
6848
```

Check if indent table `indent_table` has any indents.

```
6849 local function has_indents(indent_table)
6850 return indent_table ~= nil and
6851 indent_table.indents ~= nil and
6852 next(indent_table.indents) ~= nil
6853 end
6854
```

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

```
6855 local function add_trail(indent_table, trail_info)
6856 indent_table.trail = trail_info
6857 return indent_table
6858 end
6859
```

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

```
6860 local function remove_trail(indent_table)
6861 indent_table.trail = nil
6862 return indent_table
6863 end
6864
```

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

```
6865 local function update_indent_table(indent_table, new_indent, add)
6866 indent_table = remove_trail(indent_table)
6867
6868 if not has_indents(indent_table) then
6869 indent_table.indents = {}
6870 end
6871
6872
6873 if add then
6874 indent_table.indents[#indent_table.indents + 1] = new_indent
```



```

6875 else
6876 if indent_table.indents[#indent_table.indents].name
6877 == new_indent.name then
6878 indent_table.indents[#indent_table.indents] = nil
6879 end
6880 end
6881
6882 return indent_table
6883 end
6884

```

Remove an indent by its name `name`.

```

6885 local function remove_indent(name)
6886 local remove_indent_level =
6887 function(s, i, indent_table) -- luacheck: ignore s i
6888 indent_table = update_indent_table(indent_table, {name=name},
6889 false)
6890 return true, indent_table
6891 end
6892
6893 return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6894 end
6895

```

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`.

```

6896 local function process_starter_spacing(indent, spacing,
6897 minimum, left_strip_length)
6898 left_strip_length = left_strip_length or 0
6899
6900 local count = 0
6901 local tab_value = 4 - (indent) % 4
6902
6903 local code_started, minimum_found = false, false
6904 local code_start, minimum_remainder = "", ""
6905
6906 local left_total_stripped = 0
6907 local full_remainder = ""
6908
6909 if spacing ~= nil then
6910 for i = 1, #spacing do
6911 local character = spacing:sub(i, i)
6912
6913 if character == "\t" then

```

```

6914 count = count + tab_value
6915 tab_value = 4
6916 elseif character == " " then
6917 count = count + 1
6918 tab_value = 4 - (1 - tab_value) % 4
6919 end
6920
6921 if (left_strip_length ~= 0) then
6922 local possible_to_strip = math.min(count, left_strip_length)
6923 count = count - possible_to_strip
6924 left_strip_length = left_strip_length - possible_to_strip
6925 left_total_stripped = left_total_stripped + possible_to_strip
6926 else
6927 full_remainder = full_remainder .. character
6928 end
6929
6930 if (minimum_found) then
6931 minimum_remainder = minimum_remainder .. character
6932 elseif (count >= minimum) then
6933 minimum_found = true
6934 minimum_remainder = minimum_remainder
6935 .. string.rep(" ", count - minimum)
6936 end
6937
6938 if (code_started) then
6939 code_start = code_start .. character
6940 elseif (count >= minimum + 4) then
6941 code_started = true
6942 code_start = code_start
6943 .. string.rep(" ", count - (minimum + 4))
6944 end
6945 end
6946 end
6947
6948 local remainder
6949 if (code_started) then
6950 remainder = code_start
6951 else
6952 remainder = string.rep(" ", count - minimum)
6953 end
6954
6955 local is_minimum = count >= minimum
6956 return {
6957 is_code = code_started,
6958 remainder = remainder,
6959 left_total_stripped = left_total_stripped,
6960 is_minimum = is_minimum,

```

```

6961 minimum_remainder = minimum_remainder,
6962 total_length = count,
6963 full_remainder = full_remainder
6964 }
6965 end
6966

```

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

```

6967 local function count_indent_tab_level(indent_table)
6968 local count = 0
6969 if not has_indents(indent_table) then
6970 return count
6971 end
6972
6973 for i=1, #indent_table.indents do
6974 count = count + indent_table.indents[i].length
6975 end
6976 return count
6977 end
6978

```

Count the total width of a delimiter `delimiter`.

```

6979 local function total_delimiter_length(delimiter)
6980 local count = 0
6981 if type(delimiter) == "string" then return #delimiter end
6982 for _, value in pairs(delimiter) do
6983 count = count + total_delimiter_length(value)
6984 end
6985 return count
6986 end
6987

```

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`.

```

6988 local function process_starter_indent(_, _, indent_table, starter,
6989 is_blank, indent_type, breakable)
6990 local last_trail = starter[1]
6991 local delimiter = starter[2]
6992 local raw_new_trail = starter[3]
6993
6994 if indent_type == "bq" and not breakable then
6995 indent_table.ignore_blockquote_blank = true
6996 end
6997
6998 if has_trail(indent_table) then
6999 local trail = indent_table.trail
7000 if trail.is_code then
7001 return false

```

```

7002 end
7003 last_trail = trail.remainder
7004 else
7005 local sp = process_starter_spacing(0, last_trail, 0, 0)
7006
7007 if sp.is_code then
7008 return false
7009 end
7010 last_trail = sp.remainder
7011 end
7012
7013 local preceding_indentation = count_indent_tab_level(indent_table) % 4
7014 local last_trail_length = #last_trail
7015 local delimiter_length = total_delimiter_length(delimiter)
7016
7017 local total_indent_level = preceding_indentation + last_trail_length
7018 + delimiter_length
7019
7020 local sp = {}
7021 if not is_blank then
7022 sp = process_starter_spacing(total_indent_level, raw_new_trail,
7023 0, 1)
7024 end
7025
7026 local del_trail_length = sp.left_total_stripped
7027 if is_blank then
7028 del_trail_length = 1
7029 elseif not sp.is_code then
7030 del_trail_length = del_trail_length + #sp.remainder
7031 end
7032
7033 local indent_length = last_trail_length + delimiter_length
7034 + del_trail_length
7035 local new_indent_info = {name=indent_type, length=indent_length}
7036
7037 indent_table = update_indent_table(indent_table, new_indent_info,
7038 true)
7039 indent_table = add_trail(indent_table,
7040 {is_code=sp.is_code,
7041 remainder=sp.remainder,
7042 total_length=sp.total_length,
7043 full_remainder=sp.full_remainder})
7044
7045 return true, indent_table
7046 end
7047

```

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

```
7048 local function decode_pattern(name)
7049 local delimiter = parsers.succeed
7050 if name == "bq" then
7051 delimiter = parsers.more
7052 end
7053
7054 return C(parsers.optionalspace) * C(delimiter)
7055 * C(parsers.optionalspace) * Cp()
7056 end
7057
```

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

```
7058 local function left_blank_starter(indent_table)
7059 local blank_starter_index
7060
7061 if not has_indents(indent_table) then
7062 return
7063 end
7064
7065 for i = #indent_table.indents,1,-1 do
7066 local value = indent_table.indents[i]
7067 if value.name == "li" then
7068 blank_starter_index = i
7069 else
7070 break
7071 end
7072 end
7073
7074 return blank_starter_index
7075 end
7076
```

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.

```
7077 local function traverse_indent(s, i, indent_table, is_optional,
7078 is_blank, current_line_indents)
7079 local new_index = i
7080
7081 local preceding_indentation = 0
7082 local current_trail = {}
7083
7084 local blank_starter = left_blank_starter(indent_table)
```

```

7085
7086 if current_line_indents == nil then
7087 current_line_indents = {}
7088 end
7089
7090 for index = 1,#indent_table.indents do
7091 local value = indent_table.indents[index]
7092 local pattern = decode_pattern(value.name)
7093
7094 -- match decoded pattern
7095 local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
7096 if new_indent_info == nil then
7097 local blankline_end = lpeg.match(
7098 Ct(parsers.blankline * Cg(Cp(), "pos")), s, new_index)
7099 if is_optional or not indent_table.ignore_blockquote_blank
7100 or not blankline_end then
7101 return is_optional, new_index, current_trail,
7102 current_line_indents
7103 end
7104
7105 return traverse_indent(s, tonumber(blankline_end.pos),
7106 indent_table, is_optional, is_blank,
7107 current_line_indents)
7108 end
7109
7110 local raw_last_trail = new_indent_info[1]
7111 local delimiter = new_indent_info[2]
7112 local raw_new_trail = new_indent_info[3]
7113 local next_index = new_indent_info[4]
7114
7115 local space_only = delimiter == ""
7116
7117 -- check previous trail
7118 if not space_only and next(current_trail) == nil then
7119 local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
7120 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7121 total_length=sp.total_length,
7122 full_remainder=sp.full_remainder}
7123 end
7124
7125 if next(current_trail) ~= nil then
7126 if not space_only and current_trail.is_code then
7127 return is_optional, new_index, current_trail,
7128 current_line_indents
7129 end
7130 if current_trail.internal_remainder ~= nil then
7131 raw_last_trail = current_trail.internal_remainder

```

```

7132 end
7133 end
7134
7135 local raw_last_trail_length = 0
7136 local delimiter_length = 0
7137
7138 if not space_only then
7139 delimiter_length = #delimiter
7140 raw_last_trail_length = #raw_last_trail
7141 end
7142
7143 local total_indent_level = preceding_indentation
7144 + raw_last_trail_length + delimiter_length
7145
7146 local spacing_to_process
7147 local minimum = 0
7148 local left_strip_length = 0
7149
7150 if not space_only then
7151 spacing_to_process = raw_new_trail
7152 left_strip_length = 1
7153 else
7154 spacing_to_process = raw_last_trail
7155 minimum = value.length
7156 end
7157
7158 local sp = process_starter_spacing(total_indent_level,
7159 spacing_to_process, minimum,
7160 left_strip_length)
7161
7162 if space_only and not sp.is_minimum then
7163 return is_optional or (is_blank and blank_starter <= index),
7164 new_index, current_trail, current_line_indents
7165 end
7166
7167 local indent_length = raw_last_trail_length + delimiter_length
7168 + sp.left_total_stripped
7169
7170 -- update info for the next pattern
7171 if not space_only then
7172 preceding_indentation = preceding_indentation + indent_length
7173 else
7174 preceding_indentation = preceding_indentation + value.length
7175 end
7176
7177 current_trail = {is_code=sp.is_code, remainder=sp.remainder,
7178 internal_remainder=sp.minimum_remainder,

```

```

7179 total_length=sp.total_length,
7180 full_remainder=sp.full_remainder}
7181
7182 current_line_indents[#current_line_indents + 1] = new_indent_info
7183 new_index = next_index
7184 end
7185
7186 return true, new_index, current_trail, current_line_indents
7187 end
7188

```

Check if a code trail is expected.

```

7189 local function check_trail(expect_code, is_code)
7190 return (expect_code and is_code) or (not expect_code and not is_code)
7191 end
7192

```

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](#).

```

7193 local check_trail_joined =
7194 function(s, i, indent_table, -- luacheck: ignore s i
7195 spacing, expect_code, omit_remainder)
7196 local is_code
7197 local remainder
7198
7199 if has_trail(indent_table) then
7200 local trail = indent_table.trail
7201 is_code = trail.is_code
7202 if is_code then
7203 remainder = trail.remainder
7204 else
7205 remainder = trail.full_remainder
7206 end
7207 else
7208 local sp = process_starter_spacing(0, spacing, 0, 0)
7209 is_code = sp.is_code
7210 if is_code then
7211 remainder = sp.remainder
7212 else
7213 remainder = sp.full_remainder
7214 end
7215 end
7216
7217 local result = check_trail(expect_code, is_code)
7218 if omit_remainder then
7219 return result
7220 end

```



```

7221 return result, remainder
7222 end
7223

```

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

```

7224 local check_trail_length =
7225 function(s, i, indent_table, -- luacheck: ignore s i
7226 spacing, min, max)
7227 local trail
7228
7229 if has_trail(indent_table) then
7230 trail = indent_table.trail
7231 else
7232 trail = process_starter_spacing(0, spacing, 0, 0)
7233 end
7234
7235 local total_length = trail.total_length
7236 if total_length == nil then
7237 return false
7238 end
7239
7240 return min <= total_length and total_length <= max
7241 end
7242

```

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

```

7243 local function check_continuation_indentation(s, i, indent_table,
7244 is_optional, is_blank)
7245 if not has_indents(indent_table) then
7246 return true
7247 end
7248
7249 local passes, new_index, current_trail, current_line_indents =
7250 traverse_indent(s, i, indent_table, is_optional, is_blank)
7251
7252 if passes then
7253 indent_table.current_line_indents = current_line_indents
7254 indent_table = add_trail(indent_table, current_trail)
7255 return new_index, indent_table
7256 end
7257 return false
7258 end
7259

```

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

```

7260 local function get_last_indent_name(indent_table)
7261 if has_indents(indent_table) then

```

```

7262 return indent_table.indents[#indent_table.indents].name
7263 end
7264 end
7265

```

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

```

7266 local function remove_remainder_if_blank(indent_table, remainder)
7267 if get_last_indent_name(indent_table) == "li" then
7268 return ""
7269 end
7270 return remainder
7271 end
7272

```

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.

```

7273 local check_trail_type =
7274 function(s, i, -- luacheck: ignore s i
7275 trail, spacing, trail_type)
7276 if trail == nil then
7277 trail = process_starter_spacing(0, spacing, 0, 0)
7278 end
7279
7280 if trail_type == "non-code" then
7281 return check_trail(false, trail.is_code)
7282 end
7283 if trail_type == "code" then
7284 return check_trail(true, trail.is_code)
7285 end
7286 if trail_type == "full-code" then
7287 if (trail.is_code) then
7288 return i, trail.remainder
7289 end
7290 return i, ""
7291 end
7292 if trail_type == "full-any" then
7293 return i, trail.internal_remainder
7294 end
7295 end
7296

```

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

```

7297 local trail_freezing =
7298 function(s, i, -- luacheck: ignore s i
7299 indent_table, is_freezing)
7300 if is_freezing then
7301 if indent_table.is_trail_frozen then

```

```

7302 indent_table.trail = indent_table.frozen_trail
7303 else
7304 indent_table.frozen_trail = indent_table.trail
7305 indent_table.is_trail_frozen = true
7306 end
7307 else
7308 indent_table.frozen_trail = nil
7309 indent_table.is_trail_frozen = false
7310 end
7311 return true, indent_table
7312 end
7313

```

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`.

```

7314 local check_continuation_indentation_and_trail =
7315 function (s, i, indent_table, is_optional, is_blank, trail_type,
7316 reset_rem, omit_remainder)
7317 if not has_indents(indent_table) then
7318 local spacing, new_index = lpeg.match(C(parsers.spacechar^0)
7319 * Cp(), s, i)
7320 local result, remainder = check_trail_type(s, i,
7321 indent_table.trail, spacing, trail_type)
7322 if remainder == nil then
7323 if result then
7324 return new_index
7325 end
7326 return false
7327 end
7328 if result then
7329 return new_index, remainder
7330 end
7331 return false
7332 end
7333
7334 local passes, new_index, current_trail = traverse_indent(s, i,
7335 indent_table, is_optional, is_blank)
7336
7337 if passes then
7338 local spacing
7339 if current_trail == nil then
7340 local newer_spacing, newer_index = lpeg.match(
7341 C(parsers.spacechar^0) * Cp(), s, i)
7342 current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
7343 new_index = newer_index
7344 spacing = newer_spacing

```

```

7345 else
7346 spacing = current_trail.remainder
7347 end
7348 local result, remainder = check_trail_type(s, new_index,
7349 current_trail, spacing, trail_type)
7350 if remainder == nil or omit_remainder then
7351 if result then
7352 return new_index
7353 end
7354 return false
7355 end
7356
7357 if is_blank and reset_rem then
7358 remainder = remove_remainder_if_blank(indent_table, remainder)
7359 end
7360 if result then
7361 return new_index, remainder
7362 end
7363 return false
7364 end
7365 return false
7366 end
7367

```

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

```

7368 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0)
7369 * Cc(false), check_trail_joined)
7370
7371 parsers.check_trail_no_rem = Cmt(Cb("indent_info")
7372 * C(parsers.spacechar^0) * Cc(false)
7373 * Cc(true), check_trail_joined)
7374
7375 parsers.check_code_trail = Cmt(Cb("indent_info")
7376 * C(parsers.spacechar^0)
7377 * Cc(true), check_trail_joined)
7378
7379 parsers.check_trail_length_range = function(min, max)
7380 return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min)
7381 * Cc(max), check_trail_length)
7382 end
7383
7384 parsers.check_trail_length = function(n)
7385 return parsers.check_trail_length_range(n, n)
7386 end
7387

```

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

```

7388 parsers.freeze_trail = Cg(Cmt(Cb("indent_info")
7389 * Cc(true), trail_freezing), "indent_info")
7390
7391 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false),
7392 trail_freezing), "indent_info")
7393

```

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

```

7394 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false),
7395 check_continuation_indentation)
7396
7397 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true),
7398 check_continuation_indentation)
7399
7400 parsers.check_minimal_blank_indent
7401 = Cmt(Cb("indent_info") * Cc(false)
7402 * Cc(true)
7403 , check_continuation_indentation)
7404

```

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

```

7405
7406 parsers.check_minimal_indent_and_trail =
7407 Cmt(Cb("indent_info")
7408 * Cc(false) * Cc(false) * Cc("non-code") * Cc(true)
7409 , check_continuation_indentation_and_trail)
7410
7411 parsers.check_minimal_indent_and_code_trail =
7412 Cmt(Cb("indent_info")
7413 * Cc(false) * Cc(false) * Cc("code") * Cc(false)
7414 , check_continuation_indentation_and_trail)
7415
7416 parsers.check_minimal_blank_indent_and_full_code_trail =
7417 Cmt(Cb("indent_info")
7418 * Cc(false) * Cc(true) * Cc("full-code") * Cc(true)
7419 , check_continuation_indentation_and_trail)
7420
7421 parsers.check_minimal_indent_and_any_trail =
7422 Cmt(Cb("indent_info")
7423 * Cc(false) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7424 , check_continuation_indentation_and_trail)
7425
7426 parsers.check_minimal_blank_indent_and_any_trail =
7427 Cmt(Cb("indent_info")
7428 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7429 , check_continuation_indentation_and_trail)

```

```

7430
7431 parsers.check_minimal_blank_indent_and_any_trail_no_rem =
7432 Cmt(Cb("indent_info")
7433 * Cc(false) * Cc(true) * Cc("full-any") * Cc(true) * Cc(true)
7434 , check_continuation_indentation_and_trail)
7435
7436 parsers.check_optional_indent_and_any_trail =
7437 Cmt(Cb("indent_info")
7438 * Cc(true) * Cc(false) * Cc("full-any") * Cc(true) * Cc(false)
7439 , check_continuation_indentation_and_trail)
7440
7441 parsers.check_optional_blank_indent_and_any_trail =
7442 Cmt(Cb("indent_info")
7443 * Cc(true) * Cc(true) * Cc("full-any") * Cc(true) * Cc(false)
7444 , check_continuation_indentation_and_trail)
7445

```

The following patterns specify behaviour around newlines.

```

7446
7447 parsers.spnlc_noexc = parsers.optionalspace
7448 * (parsers.newline
7449 * parsers.check_minimal_indent_and_any_trail)^-1
7450
7451 parsers.spnlc = parsers.optionalspace
7452 * (V("EndlineNoSub"))^-1
7453
7454 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
7455 + parsers.spacechar^1
7456
7457 parsers.only_blank = parsers.spacechar^0
7458 * (parsers.newline + parsers.eof)
7459
7460 % \end{macrocode}
7461 % \begin{figure}
7462 % \hspace*{-0.1\textwidth}
7463 % \begin{minipage}{1.2\textwidth}
7464 % \centering
7465 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
7466 % \node[state, initial by diamond, accepting] (noop) {initial};
7467 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
7468 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
7469 % \node[state] (comment) [below=of noop] {comment};
7470 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
7471 % \node[state] (blank_line) [below right=of comment] {blank line};
7472 % \path[->]
7473 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^
7474 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
7475 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl

```

```

7476 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge\wedge\drsh$}
7477 % edge [bend left=10] node {match \drsh} (leading_spaces)
7478 % (leading_spaces) edge [loop below] node {match [\textvisiblespace\rightleftarrows}
7479 % edge [bend right=90] node [right] {match \textbackslash} (odd_backslash)
7480 % edge [bend left=10] node {match \%} (comment)
7481 % edge [bend right=10] node {ϵ} (blank_line)
7482 % edge [bend left=10] node [align=center, right=0.3cm] {match [$^\wedge\wedge\drsh$}
7483 % (blank_line) edge [loop below] node {match [\textvisiblespace\rightleftarrows}
7484 % edge [bend left=90] node [align=center, below=1.2cm] {match \drsh}
7485 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2cm]
7486 % edge [bend right=10] node [align=center, above left=-0.3cm, xshift=0.1cm] {match [$^\wedge\wedge\textbackslash$}
7487 % (even_backslash) edge [bend left=10] node {ϵ} (noop);
7488 % \end{tikzpicture}
7489 % \caption{A pushdown automaton that recognizes \TeX{} comments}
7490 % \label{fig:commented_line}
7491 % \end{minipage}
7492 % \end{figure}
7493 % \begin{markdown}
7494 %
7495 % The \luamdef{parsers.comment_line} parser recognizes the regular
7496 % language of \TeX{} comments, see an equivalent finite automaton in Figure
7497 % <#fig:commented_line>.
7498 %
7499 % \end{markdown}
7500 % \begin{macrocode}
7501 parsers.comment_line_letter = parsers.linechar
7502 + parsers.newline
7503 - parsers.backslash
7504 - parsers.percent
7505 parsers.comment_line = Cg(Cc(""), "backslashes")
7506 * ((#(parsers.comment_line_letter
7507 - parsers.newline)
7508 * Cb("backslashes")
7509 * Cs(parsers.comment_line_letter
7510 - parsers.newline)^1 -- initial
7511 * Cg(Cc(""), "backslashes"))
7512 + #(parsers.backslash * parsers.backslash)
7513 * Cg((parsers.backslash -- even backslash
7514 * parsers.backslash)^1, "backslashes")
7515 + (parsers.backslash
7516 * (#parsers.percent
7517 * Cb("backslashes")
7518 / function(backslashes)
7519 return string.rep("\\", #backslashes / 2)
7520 end
7521 * C(parsers.percent)

```

```

7522 + #parsers.commented_line_letter
7523 * Cb("backslashes")
7524 * Cc("\\")
7525 * C(parsers.commented_line_letter))
7526 * Cg(Cc(""), "backslashes"))^0
7527 * (#parsers.percent
7528 * Cb("backslashes")
7529 / function(backslashes)
7530 return string.rep("\\", #backslashes / 2)
7531 end
7532 * ((parsers.percent -- comment
7533 * parsers.line
7534 * #parsers.blankline) -- blank line
7535 / "\n"
7536 + parsers.percent -- comment
7537 * parsers.line
7538 * parsers.optionalspace) -- leading spaces
7539 + #(parsers.newline)
7540 * Cb("backslashes")
7541 * C(parsers.newline))
7542
7543 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7544 - parsers.blankline)^0
7545
7546 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7547 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7548 parsers.attribute_key = (parsers.attribute_key_char
7549 - parsers.dash - parsers.digit)
7550 * parsers.attribute_key_char^0
7551 parsers.attribute_value = ((parsers.dquote / "\"")
7552 * (parsers.anyescaped - parsers.dquote)^0
7553 * (parsers.dquote / "\""))
7554 + ((parsers.squote / "\"")
7555 * (parsers.anyescaped - parsers.squote)^0
7556 * (parsers.squote / "\""))
7557 + (parsers.anyescaped
7558 - parsers.dquote
7559 - parsers.rbrace
7560 - parsers.space)^0
7561 parsers.attribute_identfier = parsers.attribute_key_char^1
7562 parsers.attribute_classname = parsers.letter
7563 * parsers.attribute_key_char^0
7564 parsers.attribute_raw = parsers.attribute_raw_char^1
7565
7566 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7567 + C(parsers.hash
7568 * parsers.attribute_identfier)

```



```

7569 + C(parsers.period
7570 * parsers.attribute_classname)
7571 + Cs(parsers.attribute_key
7572 * parsers.optionalspace
7573 * parsers.equal
7574 * parsers.optionalspace
7575 * parsers.attribute_value)
7576 parsers.attributes = parsers.lbrace
7577 * parsers.optionalspace
7578 * parsers.attribute
7579 * (parsers.spacechar^1
7580 * parsers.attribute)^0
7581 * parsers.optionalspace
7582 * parsers.rbrace
7583
7584 parsers.raw_attribute = parsers.lbrace
7585 * parsers.optionalspace
7586 * parsers.equal
7587 * C(parsers.attribute_raw)
7588 * parsers.optionalspace
7589 * parsers.rbrace
7590
7591 -- block followed by 0 or more optionally
7592 -- indented blocks with first line indented.
7593 parsers.indented_blocks = function(bl)
7594 return Cs(bl
7595 * (parsers.blankline^1
7596 * parsers.indent
7597 * -parsers.blankline
7598 * bl)^0
7599 * (parsers.blankline^1 + parsers.eof))
7600 end

```

### 3.1.5.2 Parsers Used for HTML Entities

```

7601 local function repeat_between(pattern, min, max)
7602 return -pattern^(max + 1) * pattern^min
7603 end
7604
7605 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7606 * C(repeat_between(parsers.hexdigit, 1, 6))
7607 * parsers.semicolon
7608 parsers.decentity = parsers.ampersand * parsers.hash
7609 * C(repeat_between(parsers.digit, 1, 7))
7610 * parsers.semicolon
7611 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7612 * parsers.semicolon

```

```

7613
7614 parsers.html_entities
7615 = parsers.hexentity / entities.hex_entity_with_x_char
7616 + parsers.decentity / entities.dec_entity
7617 + parsers.tagentity / entities.char_entity

```

### 3.1.5.3 Parsers Used for Markdown Lists

```

7618 parsers.bullet = function(bullet_char, interrupting)
7619 local allowed_end
7620 if interrupting then
7621 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7622 else
7623 allowed_end = C(parsers.spacechar^1)
7624 + #(parsers.newline + parsers.eof)
7625 end
7626 return parsers.check_trail
7627 * Ct(C(bullet_char) * Cc(""))
7628 * allowed_end
7629 end
7630
7631 local function tickbox(interior)
7632 return parsers.optionalspace * parsers.lbracket
7633 * interior * parsers.rbracket * parsers.spacechar^1
7634 end
7635
7636 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7637 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7638 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7639

```

### 3.1.5.4 Parsers Used for Markdown Code Spans

```

7640 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7641
7642 local function captures_equal_length(_,i,a,b)
7643 return #a == #b and i
7644 end
7645
7646 parsers.closeticks = Cmt(C(parsers.backtick^1)
7647 * Cb("ticks"), captures_equal_length)
7648
7649 parsers.intickschar = (parsers.any - S("\n\r`"))
7650 + V("NoSoftLineBreakEndline")
7651 + (parsers.backtick^1 - parsers.closeticks)
7652
7653 local function process_inticks(s)
7654 s = s:gsub("\n", " ")

```

```

7655 s = s:gsub("^ (.*) $", "%1")
7656 return s
7657 end
7658
7659 parsers.inticks = parsers.openticks
7660 * C(parsers.space^0)
7661 * parsers.closeticks
7662 + parsers.openticks
7663 * Cs(Cs(parsers.intickschar^0) / process_inticks)
7664 * parsers.closeticks
7665

```

### 3.1.5.5 Parsers Used for HTML

```

7666 -- case-insensitive match (we assume s is lowercase)
7667 -- must be single byte encoding
7668 parsers.keyword_exact = function(s)
7669 local parser = P(0)
7670 for i=1,#s do
7671 local c = s:sub(i,i)
7672 local m = c .. upper(c)
7673 parser = parser * S(m)
7674 end
7675 return parser
7676 end
7677
7678 parsers.special_block_keyword =
7679 parsers.keyword_exact("pre") +
7680 parsers.keyword_exact("script") +
7681 parsers.keyword_exact("style") +
7682 parsers.keyword_exact("textarea")
7683
7684 parsers.block_keyword =
7685 parsers.keyword_exact("address") +
7686 parsers.keyword_exact("article") +
7687 parsers.keyword_exact("aside") +
7688 parsers.keyword_exact("base") +
7689 parsers.keyword_exact("basefont") +
7690 parsers.keyword_exact("blockquote") +
7691 parsers.keyword_exact("body") +
7692 parsers.keyword_exact("caption") +
7693 parsers.keyword_exact("center") +
7694 parsers.keyword_exact("col") +
7695 parsers.keyword_exact("colgroup") +
7696 parsers.keyword_exact("dd") +
7697 parsers.keyword_exact("details") +
7698 parsers.keyword_exact("dialog") +

```

```
7699 parsers.keyword_exact("dir") +
7700 parsers.keyword_exact("div") +
7701 parsers.keyword_exact("dl") +
7702 parsers.keyword_exact("dt") +
7703 parsers.keyword_exact("fieldset") +
7704 parsers.keyword_exact("figcaption") +
7705 parsers.keyword_exact("figure") +
7706 parsers.keyword_exact("footer") +
7707 parsers.keyword_exact("form") +
7708 parsers.keyword_exact("frame") +
7709 parsers.keyword_exact("frameset") +
7710 parsers.keyword_exact("h1") +
7711 parsers.keyword_exact("h2") +
7712 parsers.keyword_exact("h3") +
7713 parsers.keyword_exact("h4") +
7714 parsers.keyword_exact("h5") +
7715 parsers.keyword_exact("h6") +
7716 parsers.keyword_exact("head") +
7717 parsers.keyword_exact("header") +
7718 parsers.keyword_exact("hr") +
7719 parsers.keyword_exact("html") +
7720 parsers.keyword_exact("iframe") +
7721 parsers.keyword_exact("legend") +
7722 parsers.keyword_exact("li") +
7723 parsers.keyword_exact("link") +
7724 parsers.keyword_exact("main") +
7725 parsers.keyword_exact("menu") +
7726 parsers.keyword_exact("menuitem") +
7727 parsers.keyword_exact("nav") +
7728 parsers.keyword_exact("noframes") +
7729 parsers.keyword_exact("ol") +
7730 parsers.keyword_exact("optgroup") +
7731 parsers.keyword_exact("option") +
7732 parsers.keyword_exact("p") +
7733 parsers.keyword_exact("param") +
7734 parsers.keyword_exact("section") +
7735 parsers.keyword_exact("source") +
7736 parsers.keyword_exact("summary") +
7737 parsers.keyword_exact("table") +
7738 parsers.keyword_exact("tbody") +
7739 parsers.keyword_exact("td") +
7740 parsers.keyword_exact("tfoot") +
7741 parsers.keyword_exact("th") +
7742 parsers.keyword_exact("thead") +
7743 parsers.keyword_exact("title") +
7744 parsers.keyword_exact("tr") +
7745 parsers.keyword_exact("track") +
```

```

7746 parsers.keyword_exact("ul")
7747
7748 -- end conditions
7749 parsers.html_blankline_end_condition
7750 = parsers.linechar^0
7751 * (parsers.newline
7752 * (parsers.check_minimal_blank_indent_and_any_trail
7753 * #parsers.blankline
7754 + parsers.check_minimal_indent_and_any_trail)
7755 * parsers.linechar^1)^0
7756 * (parsers.newline^-1 / "")
7757
7758 local function remove_trailing_blank_lines(s)
7759 return s:gsub("[\n\r]+%s*$", "")
7760 end
7761
7762 parsers.html_until_end = function(end_marker)
7763 return Cs(Cs((parsers.newline
7764 * (parsers.check_minimal_blank_indent_and_any_trail
7765 * #parsers.blankline
7766 + parsers.check_minimal_indent_and_any_trail)
7767 + parsers.linechar - end_marker)^0
7768 * parsers.linechar^0 * parsers.newline^-1)
7769 / remove_trailing_blank_lines)
7770 end
7771
7772 -- attributes
7773 parsers.html_attribute_spacing = parsers.optionalspace
7774 * V("NoSoftLineBreakEndline")
7775 * parsers.optionalspace
7776 + parsers.spacechar^1
7777
7778 parsers.html_attribute_name = (parsers.letter
7779 + parsers.colon
7780 + parsers.underscore)
7781 * (parsers.alphanumeric
7782 + parsers.colon
7783 + parsers.underscore
7784 + parsers.period
7785 + parsers.dash)^0
7786
7787 parsers.html_attribute_value = parsers.squote
7788 * (parsers.linechar - parsers.squote)^0
7789 * parsers.squote
7790 + parsers.dquote
7791 * (parsers.linechar - parsers.dquote)^0
7792 * parsers.dquote

```

```

7793 + (parsers.any
7794 - parsers.spacechar
7795 - parsers.newline
7796 - parsers.dquote
7797 - parsers.squote
7798 - parsers.backtick
7799 - parsers.equal
7800 - parsers.less
7801 - parsers.more)^1
7802
7803 parsers.html_inline_attribute_value = parsers.squote
7804 * (V("NoSoftLineBreakEndline")
7805 + parsers.any
7806 - parsers.blankline^2
7807 - parsers.squote)^0
7808 * parsers.squote
7809 + parsers.dquote
7810 * (V("NoSoftLineBreakEndline")
7811 + parsers.any
7812 - parsers.blankline^2
7813 - parsers.dquote)^0
7814 * parsers.dquote
7815 + (parsers.any
7816 - parsers.spacechar
7817 - parsers.newline
7818 - parsers.dquote
7819 - parsers.squote
7820 - parsers.backtick
7821 - parsers.equal
7822 - parsers.less
7823 - parsers.more)^1
7824
7825 parsers.html_attribute_value_specification
7826 = parsers.optionalspace
7827 * parsers.equal
7828 * parsers.optionalspace
7829 * parsers.html_attribute_value
7830
7831 parsers.html_spnl = parsers.optionalspace
7832 * (V("NoSoftLineBreakEndline")
7833 * parsers.optionalspace)^-1
7834
7835 parsers.html_inline_attribute_value_specification
7836 = parsers.html_spnl
7837 * parsers.equal
7838 * parsers.html_spnl
7839 * parsers.html_inline_attribute_value

```

```

7840
7841 parsers.html_attribute
7842 = parsers.html_attribute_spacing
7843 * parsers.html_attribute_name
7844 * parsers.html_inline_attribute_value_specification^-1
7845
7846 parsers.html_non_newline_attribute
7847 = parsers.spacechar^1
7848 * parsers.html_attribute_name
7849 * parsers.html_attribute_value_specification^-1
7850
7851 parsers.nested_breaking_blank = parsers.newline
7852 * parsers.check_minimal_blank_indent
7853 * parsers.blankline
7854
7855 parsers.html_comment_start = P("<!--")
7856
7857 parsers.html_comment_end = P("-->")
7858
7859 parsers.html_comment
7860 = Cs(parsers.html_comment_start
7861 * parsers.html_until_end(parsers.html_comment_end))
7862
7863 parsers.html_inline_comment = (parsers.html_comment_start / "")
7864 * -P(">") * -P("-->")
7865 * Cs((V("NoSoftLineBreakEndline")
7866 + parsers.any
7867 - parsers.nested_breaking_blank
7868 - parsers.html_comment_end)^0)
7869 * (parsers.html_comment_end / "")
7870
7871 parsers.html_cdatasection_start = P("<![CDATA[")
7872
7873 parsers.html_cdatasection_end = P("]]>")
7874
7875 parsers.html_cdatasection
7876 = Cs(parsers.html_cdatasection_start
7877 * parsers.html_until_end(parsers.html_cdatasection_end))
7878
7879 parsers.html_inline_cdatasection
7880 = parsers.html_cdatasection_start
7881 * Cs(V("NoSoftLineBreakEndline") + parsers.any
7882 - parsers.nested_breaking_blank - parsers.html_cdatasection_end)^0
7883 * parsers.html_cdatasection_end
7884
7885 parsers.html_declaration_start = P("<!") * parsers.letter
7886

```

```

7887 parsers.html_declaration_end = P(">")
7888
7889 parsers.html_declaration
7890 = Cs(parsers.html_declaration_start
7891 * parsers.html_until_end(parsers.html_declaration_end))
7892
7893 parsers.html_inline_declaration
7894 = parsers.html_declaration_start
7895 * Cs(V("NoSoftLineBreakEndline") + parsers.any
7896 - parsers.nested_breaking_blank - parsers.html_declaration_end)^0
7897 * parsers.html_declaration_end
7898
7899 parsers.html_instruction_start = P("<?")
7900
7901 parsers.html_instruction_end = P("?>")
7902
7903 parsers.html_instruction
7904 = Cs(parsers.html_instruction_start
7905 * parsers.html_until_end(parsers.html_instruction_end))
7906
7907 parsers.html_inline_instruction = parsers.html_instruction_start
7908 * Cs(V("NoSoftLineBreakEndline")
7909 + parsers.any
7910 - parsers.nested_breaking_blank
7911 - parsers.html_instruction_end)^0
7912 * parsers.html_instruction_end
7913
7914 parsers.html_blankline = parsers.newline
7915 * parsers.optionalspace
7916 * parsers.newline
7917
7918 parsers.html_tag_start = parsers.less
7919
7920 parsers.html_tag_closing_start = parsers.less
7921 * parsers.slash
7922
7923 parsers.html_tag_end = parsers.html_spnl
7924 * parsers.more
7925
7926 parsers.html_empty_tag_end = parsers.html_spnl
7927 * parsers.slash
7928 * parsers.more
7929
7930 -- opening tags
7931 parsers.html_any_open_inline_tag = parsers.html_tag_start
7932 * parsers.keyword
7933 * parsers.html_attribute^0

```



```

7934 * parsers.html_tag_end
7935
7936 parsers.html_any_open_tag = parsers.html_tag_start
7937 * parsers.keyword
7938 * parsers.html_non_newline_attribute^0
7939 * parsers.html_tag_end
7940
7941 parsers.html_open_tag = parsers.html_tag_start
7942 * parsers.block_keyword
7943 * parsers.html_attribute^0
7944 * parsers.html_tag_end
7945
7946 parsers.html_open_special_tag = parsers.html_tag_start
7947 * parsers.special_block_keyword
7948 * parsers.html_attribute^0
7949 * parsers.html_tag_end
7950
7951 -- incomplete tags
7952 parsers.incomplete_tag_following = parsers.spacechar
7953 + parsers.more
7954 + parsers.slash * parsers.more
7955 + #(parsers.newline + parsers.eof)
7956
7957 parsers.incomplete_special_tag_following = parsers.spacechar
7958 + parsers.more
7959 + #(parsers.newline
7960 + parsers.eof)
7961
7962 parsers.html_incomplete_open_tag = parsers.html_tag_start
7963 * parsers.block_keyword
7964 * parsers.incomplete_tag_following
7965
7966 parsers.html_incomplete_open_special_tag
7967 = parsers.html_tag_start
7968 * parsers.special_block_keyword
7969 * parsers.incomplete_special_tag_following
7970
7971 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7972 * parsers.block_keyword
7973 * parsers.incomplete_tag_following
7974
7975 parsers.html_incomplete_close_special_tag
7976 = parsers.html_tag_closing_start
7977 * parsers.special_block_keyword
7978 * parsers.incomplete_tag_following
7979
7980 -- closing tags

```

```

7981 parsers.html_close_tag = parsers.html_tag_closing_start
7982 * parsers.block_keyword
7983 * parsers.html_tag_end
7984
7985 parsers.html_any_close_tag = parsers.html_tag_closing_start
7986 * parsers.keyword
7987 * parsers.html_tag_end
7988
7989 parsers.html_close_special_tag = parsers.html_tag_closing_start
7990 * parsers.special_block_keyword
7991 * parsers.html_tag_end
7992
7993 -- empty tags
7994 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7995 * parsers.keyword
7996 * parsers.html_attribute^0
7997 * parsers.html_empty_tag_end
7998
7999 parsers.html_any_empty_tag = parsers.html_tag_start
8000 * parsers.keyword
8001 * parsers.html_non_newline_attribute^0
8002 * parsers.optionalspace
8003 * parsers.slash
8004 * parsers.more
8005
8006 parsers.html_empty_tag = parsers.html_tag_start
8007 * parsers.block_keyword
8008 * parsers.html_attribute^0
8009 * parsers.html_empty_tag_end
8010
8011 parsers.html_empty_special_tag = parsers.html_tag_start
8012 * parsers.special_block_keyword
8013 * parsers.html_attribute^0
8014 * parsers.html_empty_tag_end
8015
8016 parsers.html_incomplete_blocks
8017 = parsers.html_incomplete_open_tag
8018 + parsers.html_incomplete_open_special_tag
8019 + parsers.html_incomplete_close_tag
8020
8021 -- parse special html blocks
8022 parsers.html_blankline_ending_special_block_opening
8023 = (parsers.html_close_special_tag
8024 + parsers.html_empty_special_tag)
8025 * #(parsers.optionalspace
8026 * (parsers.newline + parsers.eof))
8027

```

```

8028 parsers.html_blankline_ending_special_block
8029 = parsers.html_blankline_ending_special_block_opening
8030 * parsers.html_blankline_end_condition
8031
8032 parsers.html_special_block_opening
8033 = parsers.html_incomplete_open_special_tag
8034 - parsers.html_empty_special_tag
8035
8036 parsers.html_closing_special_block
8037 = parsers.html_special_block_opening
8038 * parsers.html_until_end(parsers.html_close_special_tag)
8039
8040 parsers.html_special_block
8041 = parsers.html_blankline_ending_special_block
8042 + parsers.html_closing_special_block
8043
8044 -- parse html blocks
8045 parsers.html_block_opening = parsers.html_incomplete_open_tag
8046 + parsers.html_incomplete_close_tag
8047
8048 parsers.html_block = parsers.html_block_opening
8049 * parsers.html_blankline_end_condition
8050
8051 -- parse any html blocks
8052 parsers.html_any_block_opening
8053 = (parsers.html_any_open_tag
8054 + parsers.html_any_close_tag
8055 + parsers.html_any_empty_tag)
8056 * #(parsers.optionalspace * (parsers.newline + parsers.eof))
8057
8058 parsers.html_any_block = parsers.html_any_block_opening
8059 * parsers.html_blankline_end_condition
8060
8061 parsers.html_inline_comment_full = parsers.html_comment_start
8062 * -P(">") * -P("->")
8063 * Cs((V("NoSoftLineBreakEndline")
8064 + parsers.any - P("--")
8065 - parsers.nested_breaking_blank
8066 - parsers.html_comment_end)^0)
8067 * parsers.html_comment_end
8068
8069 parsers.html_inline_tags = parsers.html_inline_comment_full
8070 + parsers.html_any_empty_inline_tag
8071 + parsers.html_inline_instruction
8072 + parsers.html_inline_cdatasection
8073 + parsers.html_inline_declaration
8074 + parsers.html_any_open_inline_tag

```

```

8075 + parsers.html_any_close_tag
8076

```

### 3.1.5.6 Parsers Used for Markdown Tags and Links

```

8077 parsers.urlchar = parsers.anyescaped
8078 - parsers.newline
8079 - parsers.more
8080
8081 parsers.auto_link_scheme_part = parsers.alphanumeric
8082 + parsers.plus
8083 + parsers.period
8084 + parsers.dash
8085
8086 parsers.auto_link_scheme = parsers.letter
8087 * parsers.auto_link_scheme_part
8088 * parsers.auto_link_scheme_part^-30
8089
8090 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
8091 * (parsers.any - parsers.spacing
8092 - parsers.less - parsers.more)^0
8093
8094 parsers.printable_characters = S(" !#$%&'*/+=?^_`{|}~--")
8095
8096 parsers.email_address_local_part_char = parsers.alphanumeric
8097 + parsers.printable_characters
8098
8099 parsers.email_address_local_part
8100 = parsers.email_address_local_part_char^1
8101
8102 parsers.email_address_dns_label = parsers.alphanumeric
8103 * (parsers.alphanumeric
8104 + parsers.dash)^-62
8105 * B(parsers.alphanumeric)
8106
8107 parsers.email_address_domain = parsers.email_address_dns_label
8108 * (parsers.period
8109 * parsers.email_address_dns_label)^0
8110
8111 parsers.email_address = parsers.email_address_local_part
8112 * parsers.at
8113 * parsers.email_address_domain
8114
8115 parsers.auto_link_url = parsers.less
8116 * C(parsers.absolute_uri)
8117 * parsers.more
8118

```

```

8119 parsers.auto_link_email = parsers.less
8120 * C(parsers.email_address)
8121 * parsers.more
8122
8123 parsers.auto_link_relative_reference = parsers.less
8124 * C(parsers.urlchar^1)
8125 * parsers.more
8126
8127 parsers.autolink = parsers.auto_link_url
8128 + parsers.auto_link_email
8129
8130 -- content in balanced brackets, parentheses, or quotes:
8131 parsers.bracketed = P{ parsers.lbracket
8132 * ((parsers.backslash / "\"" * parsers.rbracket
8133 + parsers.any - (parsers.lbracket
8134 + parsers.rbracket
8135 + parsers.blankline^2)
8136) + V(1))^0
8137 * parsers.rbracket }
8138
8139 parsers.inparens = P{ parsers.lparent
8140 * ((parsers.anyescaped - (parsers.lparent
8141 + parsers.rparent
8142 + parsers.blankline^2)
8143) + V(1))^0
8144 * parsers.rparent }
8145
8146 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
8147 * ((parsers.anyescaped - (parsers.squote
8148 + parsers.blankline^2)
8149) + V(1))^0
8150 * parsers.squote }
8151
8152 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
8153 * ((parsers.anyescaped - (parsers.dquote
8154 + parsers.blankline^2)
8155) + V(1))^0
8156 * parsers.dquote }
8157
8158 parsers.link_text = parsers.lbracket
8159 * Cs((parsers.alphanumeric^1
8160 + parsers.bracketed
8161 + parsers.inticks
8162 + parsers.autolink
8163 + V("InlineHtml"))
8164 + (parsers.backslash * parsers.backslash)
8165 + (parsers.backslash

```

```

8166 * (parsers.lbracket
8167 + parsers.rbracket)
8168 + V("NoSoftLineBreakSpace")
8169 + V("NoSoftLineBreakEndline")
8170 + (parsers.any
8171 - (parsers.newline
8172 + parsers.lbracket
8173 + parsers.rbracket
8174 + parsers.blankline^2))))^0)
8175 * parsers.rbracket
8176
8177 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
8178 * #((parsers.any
8179 - parsers.rbracket)^~999
8180 * parsers.rbracket)
8181 * Cs((parsers.alphanumeric^1
8182 + parsers.inticks
8183 + parsers.autolink
8184 + V("InlineHtml")
8185 + (parsers.backslash * parsers.backslash)
8186 + (parsers.backslash
8187 * (parsers.lbracket
8188 + parsers.rbracket)
8189 + V("NoSoftLineBreakSpace")
8190 + V("NoSoftLineBreakEndline")
8191 + (parsers.any
8192 - (parsers.newline
8193 + parsers.lbracket
8194 + parsers.rbracket
8195 + parsers.blankline^2))))^1)
8196
8197 parsers.link_label = parsers.lbracket
8198 * parsers.link_label_body
8199 * parsers.rbracket
8200
8201 parsers.inparens_url = P{ parsers.lparent
8202 * ((parsers.anyescaped - (parsers.lparent
8203 + parsers.rparent
8204 + parsers.spacing)
8205) + V(1))^0
8206 * parsers.rparent }
8207
8208 -- url for markdown links, allowing nested brackets:
8209 parsers.url = parsers.less * Cs((parsers.anyescaped
8210 - parsers.newline
8211 - parsers.less
8212 - parsers.more)^0)

```

```

8213 * parsers.more
8214 + -parsers.less
8215 * Cs((parsers.inparens_url + (parsers.anyescaped
8216 - parsers.spacing
8217 - parsers.lparent
8218 - parsers.rparent))^1)
8219
8220 -- quoted text:
8221 parsers.title_s = parsers.squote
8222 * Cs((parsers.html_entities
8223 + V("NoSoftLineBreakSpace")
8224 + V("NoSoftLineBreakEndline")
8225 + (parsers.anyescaped
8226 - parsers.newline
8227 - parsers.squote
8228 - parsers.blankline^2))^0)
8229 * parsers.squote
8230
8231 parsers.title_d = parsers.dquote
8232 * Cs((parsers.html_entities
8233 + V("NoSoftLineBreakSpace")
8234 + V("NoSoftLineBreakEndline")
8235 + (parsers.anyescaped
8236 - parsers.newline
8237 - parsers.dquote
8238 - parsers.blankline^2))^0)
8239 * parsers.dquote
8240
8241 parsers.title_p = parsers.lparent
8242 * Cs((parsers.html_entities
8243 + V("NoSoftLineBreakSpace")
8244 + V("NoSoftLineBreakEndline")
8245 + (parsers.anyescaped
8246 - parsers.newline
8247 - parsers.lparent
8248 - parsers.rparent
8249 - parsers.blankline^2))^0)
8250 * parsers.rparent
8251
8252 parsers.title
8253 = parsers.title_d + parsers.title_s + parsers.title_p
8254
8255 parsers.optionaltitle
8256 = parsers.spnlc * parsers.title * parsers.spacechar^0 + Cc("")
8257

```

### 3.1.5.7 Helpers for Links and Link Reference Definitions

```
8258 -- parse a reference definition: [foo]: /bar "title"
8259 parsers.define_reference_parser = (parsers.check_trail / "")
8260 * parsers.link_label * parsers.colon
8261 * parsers.spnlc * parsers.url
8262 * (parsers.spnlc_sep * parsers.title
8263 * parsers.only_blank
8264 + Cc("") * parsers.only_blank)
```

### 3.1.5.8 Inline Elements

```
8265 parsers.Inline = V("Inline")
8266
8267 -- parse many p between starter and ender
8268 parsers.between = function(p, starter, ender)
8269 local ender2 = B(parsers.nonspacechar) * ender
8270 return (starter
8271 * #parsers.nonspacechar
8272 * Ct(p * (p - ender2)^0)
8273 * ender2)
8274 end
8275
```

### 3.1.5.9 Block Elements

```
8276 parsers.lineof = function(c)
8277 return (parsers.check_trail_no_rem
8278 * (P(c) * parsers.optionalspace)^3
8279 * (parsers.newline + parsers.eof))
8280 end
8281
8282 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
8283 + parsers.lineof(parsers.dash)
8284 + parsers.lineof(parsers.underscore)
```

### 3.1.5.10 Headings

```
8285 -- parse Atx heading start and return level
8286 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
8287 * -parsers.hash / length
8288
8289 -- parse setext header ending and return level
8290 parsers.heading_level
8291 = parsers.nonindentSPACE * parsers.equal^1
8292 * parsers.optionalspace * #parsers.newline * Cc(1)
8293 + parsers.nonindentSPACE * parsers.dash^1
8294 * parsers.optionalspace * #parsers.newline * Cc(2)
8295
```



```

8296 local function strip_atx_end(s)
8297 return s:gsub("%s+#+%s*\n$", "")
8298 end
8299
8300 parsers.atx_heading = parsers.check_trail_no_rem
8301 * Cg(parsers.heading_start, "level")
8302 * (C(parsers.optionalspace
8303 * parsers.hash^0
8304 * parsers.optionalspace
8305 * parsers.newline)
8306 + parsers.spacechar^1
8307 * 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`.

```

8308 M.reader = {}
8309 function M.reader.new(writer, options)
8310 local self = {}

```

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

```

8311 self.writer = writer
8312 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.

```

8313 self.parsers = {}
8314 (function(parsers)
8315 setmetatable(self.parsers, {
8316 __index = function (_, key)
8317 return parsers[key]
8318 end
8319 })
8320 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.

```
8321 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.

```
8322 function self.normalize_tag(tag)
8323 tag = util.ropetostring(tag)
8324 tag = tag:gsub("[\n\r\t]+", " ")
8325 tag = tag:gsub("^ ", " "):gsub(" $", "")
8326 tag = uni_algos.case.casefold(tag, true, false)
8327 return tag
8328 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.

```
8329 local function iterlines(s, f)
8330 local rope = lpeg.match(Ct((parsers.line / f)^1), s)
8331 return util.ropetostring(rope)
8332 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.

```
8333 if options.preserveTabs then
8334 self.expandtabs = function(s) return s end
8335 else
8336 self.expandtabs = function(s)
8337 if s:find("\t") then
8338 return iterlines(s, util.expand_tabs_in_line)
8339 else
8340 return s
8341 end
8342 end
8343 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 `oplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
8344 self.parser_functions = {}
```

```

8345 self.create_parser = function(name, grammar, toplevel)
8346 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.

```

8347 if toplevel and options.stripIndent then
8348 local min_prefix_length, min_prefix = nil, ''
8349 str = iterlines(str, function(line)
8350 if lpeg.match(parsers.nonemptyline, line) == nil then
8351 return line
8352 end
8353 line = util.expand_tabs_in_line(line)
8354 local prefix = lpeg.match(C(parsers.optionalspace), line)
8355 local prefix_length = #prefix
8356 local is_shorter = min_prefix_length == nil
8357 if not is_shorter then
8358 is_shorter = prefix_length < min_prefix_length
8359 end
8360 if is_shorter then
8361 min_prefix_length, min_prefix = prefix_length, prefix
8362 end
8363 return line
8364 end)
8365 str = str:gsub('^' .. min_prefix, '')
8366 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.

```

8367 if toplevel and (options.texComments or options.hybrid) then
8368 str = lpeg.match(Ct(parsers.commented_line^1), str)
8369 str = util.rope_to_string(str)
8370 end
8371 local res = lpeg.match(grammar(), str)
8372 if res == nil then
8373 return writer.error(
8374 format("Parser `%s` failed to process the input text.", name),
8375 format("Here are the first 20 characters of the remaining "
8376 .. "unprocessed text: `%s`.", str:sub(1,20))
8377)
8378 else
8379 return res
8380 end
8381 end
8382 end

```

```

8383
8384 self.create_parser("parse_blocks",
8385 function()
8386 return parsers.blocks
8387 end, true)
8388
8389 self.create_parser("parse_blocks_nested",
8390 function()
8391 return parsers.blocks_nested
8392 end, false)
8393
8394 self.create_parser("parse_inlines",
8395 function()
8396 return parsers.inlines
8397 end, false)
8398
8399 self.create_parser("parse_inlines_no_inline_note",
8400 function()
8401 return parsers.inlines_no_inline_note
8402 end, false)
8403
8404 self.create_parser("parse_inlines_no_html",
8405 function()
8406 return parsers.inlines_no_html
8407 end, false)
8408
8409 self.create_parser("parse_inlines_nbsp",
8410 function()
8411 return parsers.inlines_nbsp
8412 end, false)
8413 self.create_parser("parse_inlines_no_link_or_emphasis",
8414 function()
8415 return parsers.inlines_no_link_or_emphasis
8416 end, false)

```

### 3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

8417 parsers.minimally_indented_blankline
8418 = parsers.check_minimal_indent * (parsers.blankline / "")
8419
8420 parsers.minimally_indented_block
8421 = parsers.check_minimal_indent * V("Block")
8422
8423 parsers.minimally_indented_block_or_paragraph
8424 = parsers.check_minimal_indent * V("BlockOrParagraph")
8425

```

```

8426 parsers.minimally_indented_paragraph
8427 = parsers.check_minimal_indent * V("Paragraph")
8428
8429 parsers.minimally_indented_plain
8430 = parsers.check_minimal_indent * V("Plain")
8431
8432 parsers.minimally_indented_par_or_plain
8433 = parsers.minimally_indented_paragraph
8434 + parsers.minimally_indented_plain
8435
8436 parsers.minimally_indented_par_or_plain_no_blank
8437 = parsers.minimally_indented_par_or_plain
8438 - parsers.minimally_indented_blankline
8439
8440 parsers.minimally_indented_ref
8441 = parsers.check_minimal_indent * V("Reference")
8442
8443 parsers.minimally_indented_blank
8444 = parsers.check_minimal_indent * V("Blank")
8445
8446 parsers.conditionally_indented_blankline
8447 = parsers.check_minimal_blank_indent * (parsers.blankline / "")
8448
8449 parsers.minimally_indented_ref_or_block
8450 = parsers.minimally_indented_ref
8451 + parsers.minimally_indented_block
8452 - parsers.minimally_indented_blankline
8453
8454 parsers.minimally_indented_ref_or_block_or_par
8455 = parsers.minimally_indented_ref
8456 + parsers.minimally_indented_block_or_paragraph
8457 - parsers.minimally_indented_blankline
8458

```

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

```

8459
8460 function parsers.separator_loop(separated_block, paragraph,
8461 block_separator, paragraph_separator)
8462 return separated_block
8463 + block_separator
8464 * paragraph
8465 * separated_block
8466 + paragraph_separator
8467 * paragraph
8468 end
8469

```

```

8470 function parsers.create_loop_body_pair(separated_block, paragraph,
8471 block_separator,
8472 paragraph_separator)
8473 return {
8474 block = parsers.separator_loop(separated_block, paragraph,
8475 block_separator, block_separator),
8476 par = parsers.separator_loop(separated_block, paragraph,
8477 block_separator, paragraph_separator)
8478 }
8479 end
8480
8481 parsers.block_sep_group = function(blank)
8482 return blank^0 * parsers.eof
8483 + (blank^2 / writer.paragraphsep
8484 + blank^0 / writer.interblocksep
8485)
8486 end
8487
8488 parsers.par_sep_group = function(blank)
8489 return blank^0 * parsers.eof
8490 + blank^0 / writer.paragraphsep
8491 end
8492
8493 parsers.sep_group_no_output = function(blank)
8494 return blank^0 * parsers.eof
8495 + blank^0
8496 end
8497
8498 parsers.content_blank = parsers.minimally_indented_blankline
8499
8500 parsers.ref_or_block_separated
8501 = parsers.sep_group_no_output(parsers.content_blank)
8502 * (parsers.minimally_indented_ref
8503 - parsers.content_blank)
8504 + parsers.block_sep_group(parsers.content_blank)
8505 * (parsers.minimally_indented_block
8506 - parsers.content_blank)
8507
8508 parsers.loop_body_pair =
8509 parsers.create_loop_body_pair(
8510 parsers.ref_or_block_separated,
8511 parsers.minimally_indented_par_or_plain_no_blank,
8512 parsers.block_sep_group(parsers.content_blank),
8513 parsers.par_sep_group(parsers.content_blank))
8514
8515 parsers.content_loop = (V("Block")
8516 * parsers.loop_body_pair.block^0

```

```

8517 + (V("Paragraph") + V("Plain"))
8518 * parsers.ref_or_block_separated
8519 * parsers.loop_body_pair.block^0
8520 + (V("Paragraph") + V("Plain"))
8521 * parsers.loop_body_pair.par^0)
8522 * parsers.content_blank^0
8523
8524 parsers.indented_content = function()
8525 return Ct((V("Reference") + (parsers.blankline / ""))
8526 * parsers.content_blank^0
8527 * parsers.check_minimal_indent
8528 * parsers.content_loop
8529 + (V("Reference") + (parsers.blankline / ""))
8530 * parsers.content_blank^0
8531 + parsers.content_loop)
8532 end
8533
8534 parsers.add_indent = function(pattern, name, breakable)
8535 return Cg(Cmt(Cb("indent_info")
8536 * Ct(pattern)
8537 * (#parsers.linechar -- check if starter is blank
8538 * Cc(false) + Cc(true))
8539 * Cc(name)
8540 * Cc(breakable),
8541 process_starter_indent), "indent_info")
8542 end
8543

```

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

```

8544 if options.hashEnumerators then
8545 parsers.dig = parsers.digit + parsers.hash
8546 else
8547 parsers.dig = parsers.digit
8548 end
8549
8550 parsers.enumerator = function(delimiter_type, interrupting)
8551 local delimiter_range
8552 local allowed_end
8553 if interrupting then
8554 delimiter_range = P("1")
8555 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
8556 else
8557 delimiter_range = parsers.dig * parsers.dig^-8
8558 allowed_end = C(parsers.spacechar^1)
8559 + #(parsers.newline + parsers.eof)
8560 end

```

```

8561
8562 return parsers.check_trail
8563 * Ct(C(delimiter_range) * C(delimiter_type))
8564 * allowed_end
8565 end
8566
8567 parsers.starter = parsers.bullet(parsers.dash)
8568 + parsers.bullet(parsers.asterisk)
8569 + parsers.bullet(parsers.plus)
8570 + parsers.enumerator(parsers.period)
8571 + parsers.enumerator(parsers.rparent)
8572

```

### 3.1.6.5 Parsers Used for Blockquotes (local)

```

8573 parsers.blockquote_start
8574 = parsers.check_trail
8575 * C(parsers.more)
8576 * C(parsers.spacechar^0)
8577
8578 parsers.blockquote_body
8579 = parsers.add_indent(parsers.blockquote_start, "bq", true)
8580 * parsers.indented_content()
8581 * remove_indent("bq")
8582
8583 if not options.breakableBlockquotes then
8584 parsers.blockquote_body
8585 = parsers.add_indent(parsers.blockquote_start, "bq", false)
8586 * parsers.indented_content()
8587 * remove_indent("bq")
8588 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.

```

8589 local function parse_content_part(content_part)
8590 local rope = util.rope_to_string(content_part)
8591 local parsed
8592 = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
8593 parsed.indent_info = nil
8594 return parsed
8595 end
8596

```

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

```

8597 local collect_emphasis_content =

```



```

8598 function(t, opening_index, closing_index)
8599 local content = {}
8600
8601 local content_part = {}
8602 for i = opening_index, closing_index do
8603 local value = t[i]
8604
8605 if value.rendered ~= nil then
8606 content[#content + 1] = parse_content_part(content_part)
8607 content_part = {}
8608 content[#content + 1] = value.rendered
8609 value.rendered = nil
8610 else
8611 if value.type == "delimiter"
8612 and value.element == "emphasis" then
8613 if value.is_active then
8614 content_part[#content_part + 1]
8615 = string.rep(value.character, value.current_count)
8616 end
8617 else
8618 content_part[#content_part + 1] = value.content
8619 end
8620 value.content = ''
8621 value.is_active = false
8622 end
8623 end
8624
8625 if next(content_part) ~= nil then
8626 content[#content + 1] = parse_content_part(content_part)
8627 end
8628
8629 return content
8630 end
8631

```

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

```

8632 local function fill_emph(t, opening_index, closing_index)
8633 local content
8634 = collect_emphasis_content(t, opening_index + 1,
8635 closing_index - 1)
8636 t[opening_index + 1].is_active = true
8637 t[opening_index + 1].rendered = writer.emphasis(content)
8638 end
8639

```

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

```

8640 local function fill_strong(t, opening_index, closing_index)
8641 local content
8642 = collect_emphasis_content(t, opening_index + 1,
8643 closing_index - 1)
8644 t[opening_index + 1].is_active = true
8645 t[opening_index + 1].rendered = writer.strong(content)
8646 end
8647

```

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

```

8648 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8649 return (opening_delimiter.is_closing
8650 or closing_delimiter.is_opening)
8651 and ((opening_delimiter.original_count
8652 + closing_delimiter.original_count) % 3 == 0)
8653 and (opening_delimiter.original_count % 3 ~= 0
8654 or closing_delimiter.original_count % 3 ~= 0)
8655 end
8656

```

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`.

```

8657 local find_emphasis_opener = function(t, bottom_index, latest_index,
8658 character, closing_delimiter)
8659 for i = latest_index, bottom_index, -1 do
8660 local value = t[i]
8661 if value.is_active and
8662 value.is_opening and
8663 value.type == "delimiter" and
8664 value.element == "emphasis" and
8665 (value.character == character) and
8666 (value.current_count > 0) then
8667 if not breaks_three_rule(value, closing_delimiter) then
8668 return i
8669 end
8670 end
8671 end
8672 end
8673

```

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

```

8674 local function process_emphasis(t, opening_index, closing_index)
8675 for i = opening_index, closing_index do
8676 local value = t[i]
8677 if value.type == "delimiter" and value.element == "emphasis" then

```

```

8678 local delimiter_length = string.len(value.content)
8679 value.character = string.sub(value.content, 1, 1)
8680 value.current_count = delimiter_length
8681 value.original_count = delimiter_length
8682 end
8683 end
8684
8685 local openers_bottom = {
8686 ['*'] = {
8687 [true] = {opening_index, opening_index, opening_index},
8688 [false] = {opening_index, opening_index, opening_index}
8689 },
8690 ['_'] = {
8691 [true] = {opening_index, opening_index, opening_index},
8692 [false] = {opening_index, opening_index, opening_index}
8693 }
8694 }
8695
8696 local current_position = opening_index
8697 local max_position = closing_index
8698
8699 while current_position <= max_position do
8700 local value = t[current_position]
8701
8702 if value.type ~= "delimiter" or
8703 value.element ~= "emphasis" or
8704 not value.is_active or
8705 not value.is_closing or
8706 (value.current_count <= 0) then
8707 current_position = current_position + 1
8708 goto continue
8709 end
8710
8711 local character = value.character
8712 local is_opening = value.is_opening
8713 local closing_length_modulo_three = value.original_count % 3
8714
8715 local current_openers_bottom
8716 = openers_bottom[character][is_opening]
8717 [closing_length_modulo_three + 1]
8718
8719 local opener_position
8720 = find_emphasis_opener(t, current_openers_bottom,
8721 current_position - 1, character, value)
8722
8723 if (opener_position == nil) then
8724 openers_bottom[character][is_opening]

```

```

8725 [closing_length_modulo_three + 1]
8726 = current_position
8727 current_position = current_position + 1
8728 goto continue
8729 end
8730
8731 local opening_delimiter = t[opener_position]
8732
8733 local current_opening_count = opening_delimiter.current_count
8734 local current_closing_count = t[current_position].current_count
8735
8736 if (current_opening_count >= 2)
8737 and (current_closing_count >= 2) then
8738 opening_delimiter.current_count = current_opening_count - 2
8739 t[current_position].current_count = current_closing_count - 2
8740 fill_strong(t, opener_position, current_position)
8741 else
8742 opening_delimiter.current_count = current_opening_count - 1
8743 t[current_position].current_count = current_closing_count - 1
8744 fill_emph(t, opener_position, current_position)
8745 end
8746
8747 ::continue::
8748 end
8749 end
8750
8751 local cont = lpeg.R("\128\191") -- continuation byte
8752

```

Match a UTF-8 character of byte length [n](#).

```

8753 local function utf8_by_byte_count(n)
8754 if (n == 1) then
8755 return lpeg.R("\0\127")
8756 end
8757 if (n == 2) then
8758 return lpeg.R("\194\223") * cont
8759 end
8760 if (n == 3) then
8761 return lpeg.R("\224\239") * cont * cont
8762 end
8763 if (n == 4) then
8764 return lpeg.R("\240\244") * cont * cont * cont
8765 end
8766 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](#).

```

8767 local function check_unicode_type(s, i, start_pos, end_pos, chartype)

```

```

8768 local c
8769 local char_length
8770 for pos = start_pos, end_pos, 1 do
8771 if (start_pos < 0) then
8772 char_length = -pos
8773 else
8774 char_length = pos + 1
8775 end
8776
8777 if (chartype == "punctuation") then
8778 if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8779 return i
8780 end
8781 else
8782 c = lpeg.match({ C(utf8_by_byte_count(char_length)) }, s, i+pos)
8783 if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8784 return i
8785 end
8786 end
8787 end
8788 end
8789
8790 local function check_preceding_unicode_punctuation(s, i)
8791 return check_unicode_type(s, i, -4, -1, "punctuation")
8792 end
8793
8794 local function check_preceding_unicode_whitespace(s, i)
8795 return check_unicode_type(s, i, -4, -1, "%s")
8796 end
8797
8798 local function check_following_unicode_punctuation(s, i)
8799 return check_unicode_type(s, i, 0, 3, "punctuation")
8800 end
8801
8802 local function check_following_unicode_whitespace(s, i)
8803 return check_unicode_type(s, i, 0, 3, "%s")
8804 end
8805
8806 parsers.unicode_preceding_punctuation
8807 = B(parsers.escapable)
8808 + Cmt(parsers.succeed, check_preceding_unicode_punctuation)
8809
8810 parsers.unicode_preceding_whitespace
8811 = Cmt(parsers.succeed, check_preceding_unicode_whitespace)
8812
8813 parsers.unicode_following_punctuation
8814 = #parsers.escapable

```

```

8815 + Cmt(parsers.succeed, check_following_unicode_punctuation)
8816
8817 parsers.unicode_following_whitespace
8818 = Cmt(parsers.succeed, check_following_unicode_whitespace)
8819
8820 parsers.delimiter_run = function(character)
8821 return (B(parsers.backslash * character) + -B(character))
8822 * character~1
8823 * -#character
8824 end
8825
8826 parsers.left_flanking_delimiter_run = function(character)
8827 return (B(parsers.any)
8828 * (parsers.unicode_preceding_punctuation
8829 + parsers.unicode_preceding_whitespace)
8830 + -B(parsers.any))
8831 * parsers.delimiter_run(character)
8832 * parsers.unicode_following_punctuation
8833 + parsers.delimiter_run(character)
8834 * -(parsers.unicode_following_punctuation
8835 + parsers.unicode_following_whitespace
8836 + parsers.eof)
8837 end
8838
8839 parsers.right_flanking_delimiter_run = function(character)
8840 return parsers.unicode_preceding_punctuation
8841 * parsers.delimiter_run(character)
8842 * (parsers.unicode_following_punctuation
8843 + parsers.unicode_following_whitespace
8844 + parsers.eof)
8845 + (B(parsers.any)
8846 * -(parsers.unicode_preceding_punctuation
8847 + parsers.unicode_preceding_whitespace))
8848 * parsers.delimiter_run(character)
8849 end
8850
8851 if options.underscores then
8852 parsers.emph_start
8853 = parsers.left_flanking_delimiter_run(parsers.asterisk)
8854 + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
8855 + (parsers.unicode_preceding_punctuation
8856 * #parsers.right_flanking_delimiter_run(parsers.underscore)))
8857 * parsers.left_flanking_delimiter_run(parsers.underscore)
8858
8859 parsers.emph_end
8860 = parsers.right_flanking_delimiter_run(parsers.asterisk)
8861 + (-#parsers.left_flanking_delimiter_run(parsers.underscore)

```

```

8862 + #(parsers.left_flanking_delimiter_run(parsers.underscore)
8863 * parsers.unicode_following_punctuation))
8864 * parsers.right_flanking_delimiter_run(parsers.underscore)
8865 else
8866 parsers.emph_start
8867 = parsers.left_flanking_delimiter_run(parsers.asterisk)
8868
8869 parsers.emph_end
8870 = parsers.right_flanking_delimiter_run(parsers.asterisk)
8871 end
8872
8873 parsers.emph_capturing_open_and_close
8874 = #parsers.emph_start * #parsers.emph_end
8875 * Ct(Cg(Cc("delimiter"), "type")
8876 * Cg(Cc("emphasis"), "element")
8877 * Cg(C(parsers.emph_start), "content")
8878 * Cg(Cc(true), "is_opening")
8879 * Cg(Cc(true), "is_closing"))
8880
8881 parsers.emph_capturing_open = Ct(Cg(Cc("delimiter"), "type")
8882 * Cg(Cc("emphasis"), "element")
8883 * Cg(C(parsers.emph_start), "content")
8884 * Cg(Cc(true), "is_opening")
8885 * Cg(Cc(false), "is_closing"))
8886
8887 parsers.emph_capturing_close = Ct(Cg(Cc("delimiter"), "type")
8888 * Cg(Cc("emphasis"), "element")
8889 * Cg(C(parsers.emph_end), "content")
8890 * Cg(Cc(false), "is_opening")
8891 * Cg(Cc(true), "is_closing"))
8892
8893 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8894 + parsers.emph_capturing_open
8895 + parsers.emph_capturing_close
8896
8897 parsers.emph_open = parsers.emph_capturing_open_and_close
8898 + parsers.emph_capturing_open
8899
8900 parsers.emph_close = parsers.emph_capturing_open_and_close
8901 + parsers.emph_capturing_close
8902

```

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

```

8903 -- List of references defined in the document
8904 local references
8905

```

```

8906 -- List of note references defined in the document
8907 parsers.rawnotes = {}
8908

```

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.

```

8909 function self.register_link(_, tag, url, title,
8910 attributes)
8911 local normalized_tag = self.normalize_tag(tag)
8912 if references[normalized_tag] == nil then
8913 references[normalized_tag] = {
8914 url = url,
8915 title = title,
8916 attributes = attributes
8917 }
8918 end
8919 return ""
8920 end
8921

```

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

```

8922 function self.lookup_reference(tag)
8923 return references[self.normalize_tag(tag)]
8924 end
8925

```

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

```

8926 function self.lookup_note_reference(tag)
8927 return parsers.rawnotes[self.normalize_tag(tag)]
8928 end
8929
8930 parsers.title_s_direct_ref = parsers.squote
8931 * Cs((parsers.html_entities
8932 + (parsers.anyescaped
8933 - parsers.squote
8934 - parsers.blankline^2))^0)
8935 * parsers.squote
8936
8937 parsers.title_d_direct_ref = parsers.dquote
8938 * Cs((parsers.html_entities
8939 + (parsers.anyescaped
8940 - parsers.dquote
8941 - parsers.blankline^2))^0)
8942 * parsers.dquote
8943
8944 parsers.title_p_direct_ref = parsers.lparent

```



```

8945 * Cs((parsers.html_entities
8946 + (parsers.anyescaped
8947 - parsers.lparent
8948 - parsers.rparent
8949 - parsers.blankline^2))^0)
8950 * parsers.rparent
8951
8952 parsers.title_direct_ref = parsers.title_s_direct_ref
8953 + parsers.title_d_direct_ref
8954 + parsers.title_p_direct_ref
8955
8956 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8957 * Cg(parsers.url + Cc(""), "url")
8958 * parsers.spnl
8959 * Cg(parsers.title_direct_ref
8960 + Cc(""), "title")
8961 * parsers.spnl * parsers.rparent
8962
8963 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8964 * Cg(parsers.url + Cc(""), "url")
8965 * parsers.spnlc
8966 * Cg(parsers.title + Cc(""), "title")
8967 * parsers.spnlc * parsers.rparent
8968
8969 parsers.empty_link = parsers.lbracket
8970 * parsers.rbracket
8971
8972 parsers.inline_link = parsers.link_text
8973 * parsers.inline_direct_ref
8974
8975 parsers.full_link = parsers.link_text
8976 * parsers.link_label
8977
8978 parsers.shortcut_link = parsers.link_label
8979 * -(parsers.empty_link + parsers.link_label)
8980
8981 parsers.collapsed_link = parsers.link_label
8982 * parsers.empty_link
8983
8984 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8985 * Cg(Cc("inline"), "link_type")
8986 + #(parsers.exclamation * parsers.full_link)
8987 * Cg(Cc("full"), "link_type")
8988 + #(parsers.exclamation
8989 * parsers.collapsed_link)
8990 * Cg(Cc("collapsed"), "link_type")
8991 + #(parsers.exclamation * parsers.shortcut_link)

```

```

8992 * Cg(Cc("shortcut"), "link_type")
8993 + #(parsers.exclamation * parsers.empty_link)
8994 * Cg(Cc("empty"), "link_type")
8995
8996 parsers.link_opening = #parsers.inline_link
8997 * Cg(Cc("inline"), "link_type")
8998 + #parsers.full_link
8999 * Cg(Cc("full"), "link_type")
9000 + #parsers.collapsed_link
9001 * Cg(Cc("collapsed"), "link_type")
9002 + #parsers.shortcut_link
9003 * Cg(Cc("shortcut"), "link_type")
9004 + #parsers.empty_link
9005 * Cg(Cc("empty_link"), "link_type")
9006 + #parsers.link_text
9007 * Cg(Cc("link_text"), "link_type")
9008
9009 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
9010 * Cg(Cc("note_inline"), "link_type")
9011
9012 parsers.raw_note_opening = #(parsers.lbracket
9013 * parsers.circumflex
9014 * parsers.link_label_body
9015 * parsers.rbracket)
9016 * Cg(Cc("raw_note"), "link_type")
9017
9018 local inline_note_element = Cg(Cc("note"), "element")
9019 * parsers.note_opening
9020 * Cg(parsers.circumflex
9021 * parsers.lbracket, "content")
9022
9023 local image_element = Cg(Cc("image"), "element")
9024 * parsers.image_opening
9025 * Cg(parsers.exclamation
9026 * parsers.lbracket, "content")
9027
9028 local note_element = Cg(Cc("note"), "element")
9029 * parsers.raw_note_opening
9030 * Cg(parsers.lbracket
9031 * parsers.circumflex, "content")
9032
9033 local link_element = Cg(Cc("link"), "element")
9034 * parsers.link_opening
9035 * Cg(parsers.lbracket, "content")
9036
9037 local opening_elements = parsers.fail
9038

```

```

9039 if options.inlineNotes then
9040 opening_elements = opening_elements + inline_note_element
9041 end
9042
9043 opening_elements = opening_elements + image_element
9044
9045 if options.notes then
9046 opening_elements = opening_elements + note_element
9047 end
9048
9049 opening_elements = opening_elements + link_element
9050
9051 parsers.link_image_opening = Ct(Cg(Cc("delimiter"), "type")
9052 * Cg(Cc(true), "is_opening")
9053 * Cg(Cc(false), "is_closing")
9054 * opening_elements)
9055
9056 parsers.link_image_closing = Ct(Cg(Cc("delimiter"), "type")
9057 * Cg(Cc("link"), "element")
9058 * Cg(Cc(false), "is_opening")
9059 * Cg(Cc(true), "is_closing")
9060 * (Cg(Cc(true), "is_direct")
9061 * Cg(parsers.rbracket
9062 * #parsers.inline_direct_ref,
9063 "content")
9064 + Cg(Cc(false), "is_direct")
9065 * Cg(parsers.rbracket, "content")))
9066
9067 parsers.link_image_open_or_close = parsers.link_image_opening
9068 + parsers.link_image_closing
9069
9070 if options.html then
9071 parsers.link_emph_precedence = parsers.inticks
9072 + parsers.autolink
9073 + parsers.html_inline_tags
9074 else
9075 parsers.link_emph_precedence = parsers.inticks
9076 + parsers.autolink
9077 end
9078
9079 parsers.link_and_emph_endline = parsers.newline
9080 * ((parsers.check_minimal_indent
9081 * -V("EndlineExceptions")
9082 + parsers.check_optional_indent
9083 * -V("EndlineExceptions")
9084 * -parsers.starter) / "")
9085 * parsers.spacechar^0 / "\n"

```

```

9086
9087 parsers.link_and_emph_content
9088 = Ct(Cg(Cc("content"), "type")
9089 * Cg(Cs((parsers.link_emph_precedence
9090 + parsers.backslash * parsers.linechar
9091 + parsers.link_and_emph_endline
9092 + (parsers.linechar
9093 - parsers.blankline^2
9094 - parsers.link_image_open_or_close
9095 - parsers.emph_open_or_close))^0), "content"))
9096
9097 parsers.link_and_emph_table
9098 = (parsers.link_image_opening + parsers.emph_open)
9099 * parsers.link_and_emph_content
9100 * ((parsers.link_image_open_or_close + parsers.emph_open_or_close)
9101 * parsers.link_and_emph_content)^1
9102

```

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

```

9103 local function collect_link_content(t, opening_index, closing_index)
9104 local content = {}
9105 for i = opening_index, closing_index do
9106 content[#content + 1] = t[i].content
9107 end
9108 return util.rope_to_string(content)
9109 end
9110

```

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

```

9111 local function find_link_opener(t, bottom_index, latest_index)
9112 for i = latest_index, bottom_index, -1 do
9113 local value = t[i]
9114 if value.type == "delimiter" and
9115 value.is_opening and
9116 (value.element == "link"
9117 or value.element == "image"
9118 or value.element == "note")
9119 and not value.removed then
9120 if value.is_active then
9121 return i
9122 end
9123 value.removed = true
9124 return nil
9125 end
9126 end
9127 end

```

9128

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

```
9129 local function find_next_link_closing_index(t, latest_index)
9130 for i = latest_index, #t do
9131 local value = t[i]
9132 if value.is_closing and
9133 value.element == "link" and
9134 not value.removed then
9135 return i
9136 end
9137 end
9138 end
9139
```

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.

```
9140 local function disable_previous_link_openers(t, opening_index)
9141 if t[opening_index].element == "image" then
9142 return
9143 end
9144
9145 for i = opening_index, 1, -1 do
9146 local value = t[i]
9147 if value.is_active and
9148 value.type == "delimiter" and
9149 value.is_opening and
9150 value.element == "link" then
9151 value.is_active = false
9152 end
9153 end
9154 end
9155
```

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

```
9156 local function disable_range(t, opening_index, closing_index)
9157 for i = opening_index, closing_index do
9158 local value = t[i]
9159 if value.is_active then
9160 value.is_active = false
9161 if value.type == "delimiter" then
9162 value.removed = true
9163 end
9164 end
9165 end
9166 end
```

9167

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

```
9168 local delete_parsed_content_in_range =
9169 function(t, opening_index, closing_index)
9170 for i = opening_index, closing_index do
9171 t[i].rendered = nil
9172 end
9173 end
9174
```

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

```
9175 local function empty_content_in_range(t, opening_index, closing_index)
9176 for i = opening_index, closing_index do
9177 t[i].content = ''
9178 end
9179 end
9180
```

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

```
9181 local function join_attributes(reference_attributes, own_attributes)
9182 local merged_attributes = {}
9183 for _, attribute in ipairs(reference_attributes or {}) do
9184 table.insert(merged_attributes, attribute)
9185 end
9186 for _, attribute in ipairs(own_attributes or {}) do
9187 table.insert(merged_attributes, attribute)
9188 end
9189 if next(merged_attributes) == nil then
9190 merged_attributes = nil
9191 end
9192 return merged_attributes
9193 end
9194
```

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

```
9195 local render_link_or_image =
9196 function(t, opening_index, closing_index, content_end_index,
9197 reference)
9198 process_emphasis(t, opening_index, content_end_index)
9199 local mapped = collect_emphasis_content(t, opening_index + 1,
9200 content_end_index - 1)
9201
9202 local rendered = {}

```

```

9203 if (t[opening_index].element == "link") then
9204 rendered = writer.link(mapped, reference.url,
9205 reference.title, reference.attributes)
9206 end
9207
9208 if (t[opening_index].element == "image") then
9209 rendered = writer.image(mapped, reference.url, reference.title,
9210 reference.attributes)
9211 end
9212
9213 if (t[opening_index].element == "note") then
9214 if (t[opening_index].link_type == "note_inline") then
9215 rendered = writer.note(mapped)
9216 end
9217 if (t[opening_index].link_type == "raw_note") then
9218 rendered = writer.note(reference)
9219 end
9220 end
9221
9222 t[opening_index].rendered = rendered
9223 delete_parsed_content_in_range(t, opening_index + 1,
9224 closing_index)
9225 empty_content_in_range(t, opening_index, closing_index)
9226 disable_previous_link_openers(t, opening_index)
9227 disable_range(t, opening_index, closing_index)
9228 end
9229

```

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.

```

9230 local resolve_inline_following_content =
9231 function(t, closing_index, match_reference, match_link_attributes)
9232 local content = ""
9233 for i = closing_index + 1, #t do
9234 content = content .. t[i].content
9235 end
9236
9237 local matching_content = parsers.succeed
9238
9239 if match_reference then
9240 matching_content = matching_content
9241 * parsers.inline_direct_ref_inside
9242 end
9243
9244 if match_link_attributes then
9245 matching_content = matching_content

```

```

9246 * Cg(Ct(parsers.attributes~-1), "attributes")
9247 end
9248
9249 local matched = lpeg.match(Ct(matching_content
9250 * Cg(Cp(), "end_position")), content)
9251
9252 local matched_count = matched.end_position - 1
9253 for i = closing_index + 1, #t do
9254 local value = t[i]
9255
9256 local chars_left = matched_count
9257 matched_count = matched_count - #value.content
9258
9259 if matched_count <= 0 then
9260 value.content = value.content:sub(chars_left + 1)
9261 break
9262 end
9263
9264 value.content = ''
9265 value.is_active = false
9266 end
9267
9268 local attributes = matched.attributes
9269 if attributes == nil or next(attributes) == nil then
9270 attributes = nil
9271 end
9272
9273 return {
9274 url = matched.url or "",
9275 title = matched.title or "",
9276 attributes = attributes
9277 }
9278 end
9279

```

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.

```

9280 local function resolve_inline_link(t, opening_index, closing_index)
9281 local inline_content
9282 = resolve_inline_following_content(t, closing_index, true,
9283 t.match_link_attributes)
9284 render_link_or_image(t, opening_index, closing_index,
9285 closing_index, inline_content)
9286 end
9287

```



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

```

9288 local resolve_note_inline_link =
9289 function(t, opening_index, closing_index)
9290 local inline_content
9291 = resolve_inline_following_content(t, closing_index,
9292 false, false)
9293 render_link_or_image(t, opening_index, closing_index,
9294 closing_index, inline_content)
9295 end
9296
```

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.

```

9297 local function resolve_shortcut_link(t, opening_index, closing_index)
9298 local content
9299 = collect_link_content(t, opening_index + 1, closing_index - 1)
9300 local r = self.lookup_reference(content)
9301
9302 if r then
9303 local inline_content
9304 = resolve_inline_following_content(t, closing_index, false,
9305 t.match_link_attributes)
9306 r.attributes
9307 = join_attributes(r.attributes, inline_content.attributes)
9308 render_link_or_image(t, opening_index, closing_index,
9309 closing_index, r)
9310 end
9311 end
9312
```

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.

```

9313 local function resolve_raw_note_link(t, opening_index, closing_index)
9314 local content
9315 = collect_link_content(t, opening_index + 1, closing_index - 1)
9316 local r = self.lookup_note_reference(content)
9317
9318 if r then
9319 local parsed_ref = self.parser_functions.parse_blocks_nested(r)
9320 render_link_or_image(t, opening_index, closing_index,
9321 closing_index, parsed_ref)
9322 end
9323 end
9324
```

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.

```

9325 local function resolve_full_link(t, opening_index, closing_index)
9326 local next_link_closing_index
9327 = find_next_link_closing_index(t, closing_index + 4)
9328 local next_link_content
9329 = collect_link_content(t, closing_index + 3,
9330 next_link_closing_index - 1)
9331 local r = self.lookup_reference(next_link_content)
9332
9333 if r then
9334 local inline_content
9335 = resolve_inline_following_content(t, next_link_closing_index,
9336 false,
9337 t.match_link_attributes)
9338 r.attributes
9339 = join_attributes(r.attributes, inline_content.attributes)
9340 render_link_or_image(t, opening_index, next_link_closing_index,
9341 closing_index, r)
9342 end
9343 end
9344
```

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.

```

9345 local function resolve_collapsed_link(t, opening_index, closing_index)
9346 local next_link_closing_index
9347 = find_next_link_closing_index(t, closing_index + 4)
9348 local content
9349 = collect_link_content(t, opening_index + 1, closing_index - 1)
9350 local r = self.lookup_reference(content)
9351
9352 if r then
9353 local inline_content
9354 = resolve_inline_following_content(t, closing_index, false,
9355 t.match_link_attributes)
9356 r.attributes
9357 = join_attributes(r.attributes, inline_content.attributes)
9358 render_link_or_image(t, opening_index, next_link_closing_index,
9359 closing_index, r)
9360 end
9361 end
9362
```

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.

```
9363 local function process_links_and_emphasis(t)
9364 for _,value in ipairs(t) do
9365 value.is_active = true
9366 end
9367
9368 for i,value in ipairs(t) do
9369 if not value.is_closing
9370 or value.type ~= "delimiter"
9371 or not (value.element == "link"
9372 or value.element == "image"
9373 or value.element == "note")
9374 or value.removed then
9375 goto continue
9376 end
9377
9378 local opener_position = find_link_opener(t, 1, i - 1)
9379 if (opener_position == nil) then
9380 goto continue
9381 end
9382
9383 local opening_delimiter = t[opener_position]
9384 opening_delimiter.removed = true
9385
9386 local link_type = opening_delimiter.link_type
9387
9388 if (link_type == "inline") then
9389 resolve_inline_link(t, opener_position, i)
9390 end
9391 if (link_type == "shortcut") then
9392 resolve_shortcut_link(t, opener_position, i)
9393 end
9394 if (link_type == "full") then
9395 resolve_full_link(t, opener_position, i)
9396 end
9397 if (link_type == "collapsed") then
9398 resolve_collapsed_link(t, opener_position, i)
9399 end
9400 if (link_type == "note_inline") then
9401 resolve_note_inline_link(t, opener_position, i)
9402 end
9403 if (link_type == "raw_note") then
9404 resolve_raw_note_link(t, opener_position, i)
9405 end
9406
9407 ::continue::
```

```

9408 end
9409
9410 t[#t].content = t[#t].content:gsub("%s*$","")
9411
9412 process_emphasis(t, 1, #t)
9413 local final_result = collect_emphasis_content(t, 1, #t)
9414 return final_result
9415 end
9416
9417 function self.defer_link_and_emphasis_processing(delimiter_table)
9418 return writer.defer_call(function()
9419 return process_links_and_emphasis(delimiter_table)
9420 end)
9421 end
9422

```

### 3.1.6.8 Inline Elements (local)

```

9423 parsers.Str = (parsers.normalchar
9424 * (parsers.normalchar + parsers.at)^0)
9425 / writer.string
9426
9427 parsers.Symbol = (parsers.backtick^1 + V("SpecialChar"))
9428 / writer.string
9429
9430 parsers.Ellipsis = P("...") / writer.ellipsis
9431
9432 parsers.Smart = parsers.Ellipsis
9433
9434 parsers.Code = parsers.inticks / writer.code
9435
9436 if options.blankBeforeBlockquote then
9437 parsers.bqstart = parsers.fail
9438 else
9439 parsers.bqstart = parsers.blockquote_start
9440 end
9441
9442 if options.blankBeforeHeading then
9443 parsers.headerstart = parsers.fail
9444 else
9445 parsers.headerstart = parsers.atx_heading
9446 end
9447
9448 if options.blankBeforeList then
9449 parsers.interrupting_bullets = parsers.fail
9450 parsers.interrupting_enumerators = parsers.fail
9451 else

```

```

9452 parsers.interrupting_bullets
9453 = parsers.bullet(parsers.dash, true)
9454 + parsers.bullet(parsers.asterisk, true)
9455 + parsers.bullet(parsers.plus, true)
9456
9457 parsers.interrupting_enumerators
9458 = parsers.enumerator(parsers.period, true)
9459 + parsers.enumerator(parsers.rparent, true)
9460 end
9461
9462 if options.html then
9463 parsers.html_interrupting
9464 = parsers.check_trail
9465 * (parsers.html_incomplete_open_tag
9466 + parsers.html_incomplete_close_tag
9467 + parsers.html_incomplete_open_special_tag
9468 + parsers.html_comment_start
9469 + parsers.html_cdatasection_start
9470 + parsers.html_declaration_start
9471 + parsers.html_instruction_start
9472 - parsers.html_close_special_tag
9473 - parsers.html_empty_special_tag)
9474 else
9475 parsers.html_interrupting = parsers.fail
9476 end
9477
9478 parsers.EndlineExceptions
9479 = parsers.blankline -- paragraph break
9480 + parsers.eof -- end of document
9481 + parsers.bqstart
9482 + parsers.thematic_break_lines
9483 + parsers.interrupting_bullets
9484 + parsers.interrupting_enumerators
9485 + parsers.headerstart
9486 + parsers.html_interrupting
9487
9488 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
9489
9490 parsers.endline = parsers.newline
9491 * (parsers.check_minimal_indent
9492 * -V("EndlineExceptions")
9493 + parsers.check_optional_indent
9494 * -V("EndlineExceptions")
9495 * -parsers.starter) / function(_) return end
9496 * parsers.spacechar^0
9497
9498 parsers.Endline = parsers.endline

```

```

9499 / writer.soft_line_break
9500
9501 parsers.EndlineNoSub = parsers.endline
9502
9503 parsers.NoSoftLineBreakEndline
9504 = parsers.newline
9505 * (parsers.check_minimal_indent
9506 * -V("NoSoftLineBreakEndlineExceptions")
9507 + parsers.check_optional_indent
9508 * -V("NoSoftLineBreakEndlineExceptions")
9509 * -parsers.starter)
9510 * parsers.spacechar^0
9511 / writer.space
9512
9513 parsers.EndlineBreak = parsers.backslash * parsers.Endline
9514 / writer.hard_line_break
9515
9516 parsers.OptionalIndent
9517 = parsers.spacechar^1 / writer.space
9518
9519 parsers.Space = parsers.spacechar^2 * parsers.Endline
9520 / writer.hard_line_break
9521 + parsers.spacechar^1
9522 * parsers.Endline~-1
9523 * parsers.eof / self.expandtabs
9524 + parsers.spacechar^1 * parsers.Endline
9525 / writer.soft_line_break
9526 + parsers.spacechar^1
9527 * -parsers.newline / self.expandtabs
9528
9529 parsers.NoSoftLineBreakSpace
9530 = parsers.spacechar^2 * parsers.Endline
9531 / writer.hard_line_break
9532 + parsers.spacechar^1
9533 * parsers.Endline~-1
9534 * parsers.eof / self.expandtabs
9535 + parsers.spacechar^1 * parsers.Endline
9536 / writer.soft_line_break
9537 + parsers.spacechar^1
9538 * -parsers.newline / self.expandtabs
9539
9540 parsers.NonbreakingEndline
9541 = parsers.endline
9542 / writer.nbsp
9543
9544 parsers.NonbreakingSpace
9545 = parsers.spacechar^2 * parsers.endline

```

```

9546 / writer.nbsp
9547 + parsers.spacechar^1
9548 * parsers.endline^-1 * parsers.eof / ""
9549 + parsers.spacechar^1 * parsers.endline
9550 * parsers.optionalspace
9551 / writer.nbsp
9552 + parsers.spacechar^1 * parsers.optionalspace
9553 / writer.nbsp
9554

```

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.

```

9555 function self.auto_link_url(url, attributes)
9556 return writer.link(writer.escape(url),
9557 url, nil, attributes)
9558 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.

```

9559 function self.auto_link_email(email, attributes)
9560 return writer.link(writer.escape(email),
9561 "mailto:".email,
9562 nil, attributes)
9563 end
9564
9565 parsers.AutoLinkUrl = parsers.auto_link_url
9566 / self.auto_link_url
9567
9568 parsers.AutoLinkEmail
9569 = parsers.auto_link_email
9570 / self.auto_link_email
9571
9572 parsers.AutoLinkRelativeReference
9573 = parsers.auto_link_relative_reference
9574 / self.auto_link_url
9575
9576 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
9577 / self.defer_link_and_emphasis_processing
9578
9579 parsers.EscapedChar = parsers.backslash
9580 * C(parsers.escapable) / writer.string
9581
9582 parsers.InlineHtml = Cs(parsers.html_inline_comment)
9583 / writer.inline_html_comment
9584 + Cs(parsers.html_any_empty_inline_tag

```

```

9585 + parsers.html_inline_instruction
9586 + parsers.html_inline_cdatasection
9587 + parsers.html_inline_declaration
9588 + parsers.html_any_open_inline_tag
9589 + parsers.html_any_close_tag)
9590 / writer.inline_html_tag
9591
9592 parsers.HtmlEntity = parsers.html_entities / writer.string

```

### 3.1.6.9 Block Elements (local)

```

9593 parsers.DisplayHtml = Cs(parsers.check_trail
9594 * (parsers.html_comment
9595 + parsers.html_special_block
9596 + parsers.html_block
9597 + parsers.html_any_block
9598 + parsers.html_instruction
9599 + parsers.html_cdatasection
9600 + parsers.html_declaration))
9601 / writer.block_html_element
9602
9603 parsers.indented_non_blank_line = parsers.indentedline
9604 - parsers.blankline
9605
9606 parsers.Verbatim
9607 = Cs(parsers.check_code_trail
9608 * (parsers.line - parsers.blankline)
9609 * ((parsers.check_minimal_blank_indent_and_full_code_trail
9610 * parsers.blankline)^0
9611 * ((parsers.check_minimal_indent / "")
9612 * parsers.check_code_trail
9613 * (parsers.line - parsers.blankline))^1)^0)
9614 / self.expandtabs / writer.verbatim
9615
9616 parsers.Blockquote = parsers.blockquote_body
9617 / writer.blockquote
9618
9619 parsers.ThematicBreak = parsers.thematic_break_lines
9620 / writer.thematic_break
9621
9622 parsers.Reference = parsers.define_reference_parser
9623 / self.register_link
9624
9625 parsers.Paragraph = parsers.freeze_trail
9626 * (Ct((parsers.Inline)^1)
9627 * (parsers.newline + parsers.eof)
9628 * parsers.unfreeze_trail

```



```

9629 / writer.paragraph)
9630
9631 parsers.Plain = parsers.nonindentSpace * Ct(parsers.Inline~1)
9632 / writer.plain

```

### 3.1.6.10 Lists (local)

```

9633
9634 if options.taskLists then
9635 parsers.tickbox = (parsers.ticked_box
9636 + parsers.halfTicked_box
9637 + parsers.unticked_box
9638) / writer.tickbox
9639 else
9640 parsers.tickbox = parsers.fail
9641 end
9642
9643 parsers.list_blank = parsers.conditionally_indented_blankline
9644
9645 parsers.ref_or_block_list_separated
9646 = parsers.sep_group_no_output(parsers.list_blank)
9647 * parsers.minimally_indented_ref
9648 + parsers.block_sep_group(parsers.list_blank)
9649 * parsers.minimally_indented_block
9650
9651 parsers.ref_or_block_non_separated
9652 = parsers.minimally_indented_ref
9653 + (parsers.succeed / writer.interblocksep)
9654 * parsers.minimally_indented_block
9655 - parsers.minimally_indented_blankline
9656
9657 parsers.tight_list_loop_body_pair =
9658 parsers.create_loop_body_pair(
9659 parsers.ref_or_block_non_separated,
9660 parsers.minimally_indented_par_or_plain_no_blank,
9661 (parsers.succeed / writer.interblocksep),
9662 (parsers.succeed / writer.paragraphsep))
9663
9664 parsers.loose_list_loop_body_pair =
9665 parsers.create_loop_body_pair(
9666 parsers.ref_or_block_list_separated,
9667 parsers.minimally_indented_par_or_plain,
9668 parsers.block_sep_group(parsers.list_blank),
9669 parsers.par_sep_group(parsers.list_blank))
9670
9671 parsers.tight_list_content_loop
9672 = V("Block")

```

```

9673 * parsers.tight_list_loop_body_pair.block^0
9674 + (V("Paragraph") + V("Plain"))
9675 * parsers.ref_or_block_non_separated
9676 * parsers.tight_list_loop_body_pair.block^0
9677 + (V("Paragraph") + V("Plain"))
9678 * parsers.tight_list_loop_body_pair.par^0
9679
9680 parsers.loose_list_content_loop
9681 = V("Block")
9682 * parsers.loose_list_loop_body_pair.block^0
9683 + (V("Paragraph") + V("Plain"))
9684 * parsers.ref_or_block_list_separated
9685 * parsers.loose_list_loop_body_pair.block^0
9686 + (V("Paragraph") + V("Plain"))
9687 * parsers.loose_list_loop_body_pair.par^0
9688
9689 parsers.list_item_tightness_condition
9690 = -#(parsers.list_blank^0
9691 * parsers.minimally_indented_ref_or_block_or_par)
9692 * remove_indent("li")
9693 + remove_indent("li")
9694 * parsers.fail
9695
9696 parsers.indented_content_tight
9697 = Ct((parsers.blankline / "")
9698 * #parsers.list_blank
9699 * remove_indent("li")
9700 + ((V("Reference") + (parsers.blankline / ""))
9701 * parsers.check_minimal_indent
9702 * parsers.tight_list_content_loop
9703 + (V("Reference") + (parsers.blankline / ""))
9704 + (parsers.tickbox^-1 / writer.escape)
9705 * parsers.tight_list_content_loop
9706)
9707 * parsers.list_item_tightness_condition)
9708
9709 parsers.indented_content_loose
9710 = Ct((parsers.blankline / "")
9711 * #parsers.list_blank
9712 + ((V("Reference") + (parsers.blankline / ""))
9713 * parsers.check_minimal_indent
9714 * parsers.loose_list_content_loop
9715 + (V("Reference") + (parsers.blankline / ""))
9716 + (parsers.tickbox^-1 / writer.escape)
9717 * parsers.loose_list_content_loop))
9718
9719 parsers.TightListItem = function(starter)

```

```

9720 return -parsers.ThematicBreak
9721 * parsers.add_indent(starter, "li")
9722 * parsers.indented_content_tight
9723 end
9724
9725 parsers.LooseListItem = function(starter)
9726 return -parsers.ThematicBreak
9727 * parsers.add_indent(starter, "li")
9728 * parsers.indented_content_loose
9729 * remove_indent("li")
9730 end
9731
9732 parsers.BulletListOfType = function(bullet_type)
9733 local bullet = parsers.bullet(bullet_type)
9734 return (Ct(parsers.TightListItem(bullet)
9735 * ((parsers.check_minimal_indent / "")
9736 * parsers.TightListItem(bullet)
9737)^0
9738)
9739 * Cc(true)
9740 * -#((parsers.list_blank^0 / "")
9741 * parsers.check_minimal_indent
9742 * (bullet - parsers.ThematicBreak)
9743)
9744 + Ct(parsers.LooseListItem(bullet)
9745 * ((parsers.list_blank^0 / "")
9746 * (parsers.check_minimal_indent / "")
9747 * parsers.LooseListItem(bullet)
9748)^0
9749)
9750 * Cc(false)
9751) / writer.bulletlist
9752 end
9753
9754 parsers.BulletList = parsers.BulletListOfType(parsers.dash)
9755 + parsers.BulletListOfType(parsers.asterisk)
9756 + parsers.BulletListOfType(parsers.plus)
9757
9758 local function ordered_list(items,tight,starter)
9759 local startnum = starter[2][1]
9760 if options.startNumber then
9761 startnum = tonumber(startnum) or 1 -- fallback for '#'
9762 if startnum ~= nil then
9763 startnum = math.floor(startnum)
9764 end
9765 else
9766 startnum = nil

```

```

9767 end
9768 return writer.orderedlist(items,tight,startnum)
9769 end
9770
9771 parsers.OrderedListOfType = function(delimiter_type)
9772 local enumerator = parsers.enumerator(delimiter_type)
9773 return Cg(enumerator, "listtype")
9774 * (Ct(parsers.TightListItem(Cb("listtype"))
9775 * ((parsers.check_minimal_indent / "")
9776 * parsers.TightListItem(enumerator))^0)
9777 * Cc(true)
9778 * -#((parsers.list_blank^0 / "")
9779 * parsers.check_minimal_indent * enumerator)
9780 + Ct(parsers.LooseListItem(Cb("listtype"))
9781 * ((parsers.list_blank^0 / "")
9782 * (parsers.check_minimal_indent / "")
9783 * parsers.LooseListItem(enumerator))^0)
9784 * Cc(false)
9785) * Ct(Cb("listtype")) / ordered_list
9786 end
9787
9788 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
9789 + parsers.OrderedListOfType(parsers.rparent)

```

### 3.1.6.11 Blank (local)

```

9790 parsers.Blank = parsers.blankline / ""
9791 + V("Reference")

```

### 3.1.6.12 Headings (local)

```

9792 function parsers.parse_heading_text(s)
9793 local inlines = self.parser_functions.parse_inlines(s)
9794 local flatten_inlines = self.writer.flatten_inlines
9795 self.writer.flatten_inlines = true
9796 local flat_text = self.parser_functions.parse_inlines(s)
9797 flat_text = util.rope_to_string(flat_text)
9798 self.writer.flatten_inlines = flatten_inlines
9799 return {flat_text, inlines}
9800 end
9801
9802 -- parse atx header
9803 parsers.AtxHeading = parsers.check_trail_no_rem
9804 * Cg(parsers.heading_start, "level")
9805 * ((C(parsers.optionalspace
9806 * parsers.hash^0
9807 * parsers.optionalspace
9808 * parsers.newline)

```

```

9809 + parsers.spacechar~1
9810 * C(parsers.line))
9811 / strip_atx_end
9812 / parsers.parse_heading_text)
9813 * Cb("level")
9814 / writer.heading
9815
9816 parsers.heading_line = parsers.linechar~1
9817 - parsers.thematic_break_lines
9818
9819 parsers.heading_text = parsers.heading_line
9820 * ((V("Endline") / "\n")
9821 * (parsers.heading_line
9822 - parsers.heading_level))^0
9823 * parsers.newline~1
9824
9825 parsers.SettextHeading = parsers.freeze_trail
9826 * parsers.check_trail_no_rem
9827 * #(parsers.heading_text
9828 * parsers.check_minimal_indent
9829 * parsers.check_trail
9830 * parsers.heading_level)
9831 * Cs(parsers.heading_text)
9832 / parsers.parse_heading_text
9833 * parsers.check_minimal_indent_and_trail
9834 * parsers.heading_level
9835 * parsers.newline
9836 * parsers.unfreeze_trail
9837 / writer.heading
9838
9839 parsers.Heading = parsers.AtHeading + parsers.SettextHeading

```

### 3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output.

```

9840 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.

```

9841 local walkable_syntax = (function(global_walkable_syntax)
9842 local local_walkable_syntax = {}
9843 for lhs, rule in pairs(global_walkable_syntax) do

```

```

9844 local_walkable_syntax[lhs] = util.table_copy(rule)
9845 end
9846 return local_walkable_syntax
9847 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.

```

9848 local current_extension_name = nil
9849 self.insert_pattern = function(selector, pattern, pattern_name)
9850 assert(pattern_name == nil or type(pattern_name) == "string")
9851 local _, _, lhs, pos, rhs
9852 = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9853 assert(lhs ~= nil,
9854 [[Expected selector in form]]
9855 .. [["LHS (before|after|instead of) RHS", not "]]
9856 .. selector .. [["]]])
9857 assert(walkable_syntax[lhs] ~= nil,
9858 [[Rule]] .. lhs
9859 .. [[-> ... does not exist in markdown grammar]])
9860 assert(pos == "before" or pos == "after" or pos == "instead of",
9861 [[Expected positional specifier "before", "after",]]
9862 .. [[or "instead of", not "]]
9863 .. pos .. [["]])
9864 local rule = walkable_syntax[lhs]
9865 local index = nil
9866 for current_index, current_rhs in ipairs(rule) do
9867 if type(current_rhs) == "string" and current_rhs == rhs then
9868 index = current_index
9869 if pos == "after" then
9870 index = index + 1
9871 end
9872 break
9873 end
9874 end
9875 assert(index ~= nil,
9876 [[Rule]] .. lhs .. [[->]] .. rhs
9877 .. [[does not exist in markdown grammar]])
9878 local accountable_pattern
9879 if current_extension_name then
9880 accountable_pattern
9881 = {pattern, current_extension_name, pattern_name}
9882 else
9883 assert(type(pattern) == "string",
9884 [[reader->insert_pattern() was called outside]]
9885 .. [[an extension with]]
9886 .. [[a PEG pattern instead of a rule name]])

```

```

9887 accountable_pattern = pattern
9888 end
9889 if pos == "instead of" then
9890 rule[index] = accountable_pattern
9891 else
9892 table.insert(rule, index, accountable_pattern)
9893 end
9894 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.

```

9895 local syntax =
9896 { "Blocks",
9897
9898 Blocks = V("InitializeState")
9899 * V("ExpectedJekyllData")
9900 * 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.

```

9901 * (V("Block")
9902 * (V("Blank")^0 * parsers.eof
9903 + (V("Blank")^2 / writer.paragraphsep
9904 + V("Blank")^0 / writer.interblocksep
9905)
9906)
9907 + (V("Paragraph") + V("Plain"))
9908 * (V("Blank")^0 * parsers.eof
9909 + (V("Blank")^2 / writer.paragraphsep
9910 + V("Blank")^0 / writer.interblocksep
9911)
9912)
9913 * V("Block")
9914 * (V("Blank")^0 * parsers.eof
9915 + (V("Blank")^2 / writer.paragraphsep
9916 + V("Blank")^0 / writer.interblocksep
9917)
9918)
9919 + (V("Paragraph") + V("Plain"))
9920 * (V("Blank")^0 * parsers.eof
9921 + V("Blank")^0 / writer.paragraphsep
9922)
9923)^0,
9924
9925 ExpectedJekyllData = parsers.succeed,
9926
9927 Blank = parsers.Blank,

```

```

9928 Reference = parsers.Reference,
9929
9930 Blockquote = parsers.Blockquote,
9931 Verbatim = parsers.Verbatim,
9932 ThematicBreak = parsers.ThematicBreak,
9933 BulletList = parsers.BulletList,
9934 OrderedList = parsers.OrderedList,
9935 DisplayHtml = parsers.DisplayHtml,
9936 Heading = parsers.Heading,
9937 Paragraph = parsers.Paragraph,
9938 Plain = parsers.Plain,
9939
9940 EndlineExceptions = parsers.EndlineExceptions,
9941 NoSoftLineBreakEndlineExceptions
9942 = parsers.NoSoftLineBreakEndlineExceptions,
9943
9944 Str = parsers.Str,
9945 Space = parsers.Space,
9946 NoSoftLineBreakSpace
9947 = parsers.NoSoftLineBreakSpace,
9948 OptionalIndent = parsers.OptionalIndent,
9949 Endline = parsers.Endline,
9950 EndlineNoSub = parsers.EndlineNoSub,
9951 NoSoftLineBreakEndline
9952 = parsers.NoSoftLineBreakEndline,
9953 EndlineBreak = parsers.EndlineBreak,
9954 LinkAndEmph = parsers.LinkAndEmph,
9955 Code = parsers.Code,
9956 AutoLinkUrl = parsers.AutoLinkUrl,
9957 AutoLinkEmail = parsers.AutoLinkEmail,
9958 AutoLinkRelativeReference
9959 = parsers.AutoLinkRelativeReference,
9960 InlineHtml = parsers.InlineHtml,
9961 HtmlEntity = parsers.HtmlEntity,
9962 EscapedChar = parsers.EscapedChar,
9963 Smart = parsers.Smart,
9964 Symbol = parsers.Symbol,
9965 SpecialChar = parsers.fail,
9966 InitializeState = parsers.succeed,
9967 }

```

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]`.

```

9968 self.update_rule = function(rule_name, get_pattern)

```



```

9969 assert(current_extension_name ~= nil)
9970 assert(syntax[rule_name] ~= nil,
9971 [[Rule]] .. rule_name
9972 .. [[-> ... does not exist in markdown grammar]])
9973 local previous_pattern
9974 local extension_name
9975 if walkable_syntax[rule_name] then
9976 local previous_accountable_pattern
9977 = walkable_syntax[rule_name][1]
9978 previous_pattern = previous_accountable_pattern[1]
9979 extension_name
9980 = previous_accountable_pattern[2]
9981 .. ", " .. current_extension_name
9982 else
9983 previous_pattern = nil
9984 extension_name = current_extension_name
9985 end
9986 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

```

```

9987 if type(get_pattern) == "function" then
9988 pattern = get_pattern(previous_pattern)
9989 else
9990 assert(previous_pattern == nil,
9991 [[Rule]] .. rule_name ..
9992 [[has already been updated by]] .. extension_name)
9993 pattern = get_pattern
9994 end
9995 local accountable_pattern = { pattern, extension_name, rule_name }
9996 walkable_syntax[rule_name] = { accountable_pattern }
9997 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.

```

9998 local special_characters = {}
9999 self.add_special_character = function(c)

```

```

10000 table.insert(special_characters, c)
10001 syntax.SpecialChar = S(table.concat(special_characters, ""))
10002 end
10003
10004 self.add_special_character("*")
10005 self.add_special_character("[")
10006 self.add_special_character("]")
10007 self.add_special_character("<")
10008 self.add_special_character("!")
10009 self.add_special_character("\\")

```

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

```

10010 self.initialize_named_group = function(name, value)
10011 local pattern = Ct("")
10012 if value ~= nil then
10013 pattern = pattern / value
10014 end
10015 syntax.InitializeState = syntax.InitializeState
10016 * Cg(pattern, name)
10017 end

```

Add a named group for indentation.

```

10018 self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

10019 for _, extension in ipairs(extensions) do
10020 current_extension_name = extension.name
10021 extension.extend_writer(writer)
10022 extension.extend_reader(self)
10023 end
10024 current_extension_name = nil

```

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

```

10025 if options.debugExtensions then
10026 local sorted_lhs = {}
10027 for lhs, _ in pairs(walkable_syntax) do
10028 table.insert(sorted_lhs, lhs)
10029 end
10030 table.sort(sorted_lhs)
10031
10032 local output_lines = {"{"}
10033 for lhs_index, lhs in ipairs(sorted_lhs) do
10034 local encoded_lhs = util.encode_json_string(lhs)
10035 table.insert(output_lines, [[]] .. encoded_lhs .. [[:]])
10036 local rule = walkable_syntax[lhs]
10037 for rhs_index, rhs in ipairs(rule) do

```

```

10038 local human_readable_rhs
10039 if type(rhs) == "string" then
10040 human_readable_rhs = rhs
10041 else
10042 local pattern_name
10043 if rhs[3] then
10044 pattern_name = rhs[3]
10045 else
10046 pattern_name = "Anonymous Pattern"
10047 end
10048 local extension_name = rhs[2]
10049 human_readable_rhs = pattern_name .. [[(]]
10050 .. extension_name .. [[]]]
10051 end
10052 local encoded_rhs
10053 = util.encode_json_string(human_readable_rhs)
10054 local output_line = [[]] .. encoded_rhs
10055 if rhs_index < #rule then
10056 output_line = output_line .. ", "
10057 end
10058 table.insert(output_lines, output_line)
10059 end
10060 local output_line = "]"
10061 if lhs_index < #sorted_lhs then
10062 output_line = output_line .. ", "
10063 end
10064 table.insert(output_lines, output_line)
10065 end
10066 table.insert(output_lines, "}")
10067
10068 local output = table.concat(output_lines, "\n")
10069 local output_filename = options.debugExtensionsFileName
10070 local output_file = assert(io.open(output_filename, "w"),
10071 [[Could not open file]] .. output_filename
10072 .. [[for writing]])
10073 assert(output_file:write(output))
10074 assert(output_file:close())
10075 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](#).

```

10076 for lhs, rule in pairs(walkable_syntax) do
10077 syntax[lhs] = parsers.fail
10078 for _, rhs in ipairs(rule) do
10079 local pattern

```

Although the interface of the [reader->insert\\_pattern](#) method does not doc-

ument 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.

```
10080 if type(rhs) == "string" then
10081 pattern = V(rhs)
10082 else
10083 pattern = rhs[1]
10084 if type(pattern) == "string" then
10085 pattern = V(pattern)
10086 end
10087 end
10088 syntax[lhs] = syntax[lhs] + pattern
10089 end
10090 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.

```
10091 if options.underscores then
10092 self.add_special_character("_")
10093 end
10094
10095 if not options.codeSpans then
10096 syntax.Code = parsers.fail
10097 else
10098 self.add_special_character("`")
10099 end
10100
10101 if not options.html then
10102 syntax.DisplayHtml = parsers.fail
10103 syntax.InlineHtml = parsers.fail
10104 syntax.HtmlEntity = parsers.fail
10105 else
10106 self.add_special_character("&")
10107 end
10108
10109 if options.preserveTabs then
10110 options.stripIndent = false
10111 end
10112
10113 if not options.smartEllipses then
10114 syntax.Smart = parsers.fail
10115 else
10116 self.add_special_character(".")
10117 end
10118
10119 if not options.relativeReferences then
```

```

10120 syntax.AutoLinkRelativeReference = parsers.fail
10121 end
10122
10123 if options.contentLevel == "inline" then
10124 syntax[1] = "Inlines"
10125 syntax.Inlines = V("InitializeState")
10126 * parsers.Inline^0
10127 * (parsers.spacing^0
10128 * parsers.eof / "")
10129 syntax.Space = parsers.Space + parsers.blankline / writer.space
10130 end
10131
10132 local blocks_nested_t = util.table_copy(syntax)
10133 blocks_nested_t.ExpectedJekyllData = parsers.succeed
10134 parsers.blocks_nested = Ct(blocks_nested_t)
10135
10136 parsers.blocks = Ct(syntax)
10137
10138 local inlines_t = util.table_copy(syntax)
10139 inlines_t[1] = "Inlines"
10140 inlines_t.Inlines = V("InitializeState")
10141 * parsers.Inline^0
10142 * (parsers.spacing^0
10143 * parsers.eof / "")
10144 parsers.inlines = Ct(inlines_t)
10145
10146 local inlines_no_inline_note_t = util.table_copy(inlines_t)
10147 inlines_no_inline_note_t.InlineNote = parsers.fail
10148 parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
10149
10150 local inlines_no_html_t = util.table_copy(inlines_t)
10151 inlines_no_html_t.DisplayHtml = parsers.fail
10152 inlines_no_html_t.InlineHtml = parsers.fail
10153 inlines_no_html_t.HtmlEntity = parsers.fail
10154 parsers.inlines_no_html = Ct(inlines_no_html_t)
10155
10156 local inlines_nbsp_t = util.table_copy(inlines_t)
10157 inlines_nbsp_t.Endline = parsers.NonbreakingEndline
10158 inlines_nbsp_t.Space = parsers.NonbreakingSpace
10159 parsers.inlines_nbsp = Ct(inlines_nbsp_t)
10160
10161 local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
10162 inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
10163 inlines_no_link_or_emphasis_t.EndlineExceptions
10164 = parsers.EndlineExceptions - parsers.eof
10165 parsers.inlines_no_link_or_emphasis
10166 = Ct(inlines_no_link_or_emphasis_t)

```

Return a function that converts markdown string `input` into a plain  $\text{\TeX}$  output and returns it..

```
10167 return function(input)
```

Unicode-normalize the input.

```
10168 if options.unicodeNormalization then
10169 local form = options.unicodeNormalizationForm
10170 if form == "nfc" then
10171 input = uni_algos.normalize.NFC(input)
10172 elseif form == "nfd" then
10173 input = uni_algos.normalize.NFD(input)
10174 elseif form == "nfkc" then
10175 input = uni_algos.normalize.NFKC(input)
10176 elseif form == "nfkd" then
10177 input = uni_algos.normalize.NFKD(input)
10178 else
10179 return writer.error(
10180 format("Unknown normalization form %s.", form))
10181 end
10182 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.

```
10183 input = input:gsub("\r\n?", "\n")
10184 if input:sub(-1) ~= "\n" then
10185 input = input .. "\n"
10186 end
```

Clear the table of references.

```
10187 references = {}
10188 local document = self.parser_functions.parse_blocks(input)
10189 local output = util.ropetostring(writer.document(document))
```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```
10190 local undosep_start, undosep_end
10191 local potential_secend_start, secend_start
10192 local potential_sep_start, sep_start
10193 while true do
10194 -- find a `writer->undosep`
10195 undosep_start, undosep_end
10196 = output:find(writer.undosep_text, 1, true)
10197 if undosep_start == nil then break end
10198 -- skip any preceding section ends
10199 secend_start = undosep_start
10200 while true do
10201 potential_secend_start = secend_start - #writer.secend_text
```

```

10202 if potential_secend_start < 1
10203 or output:sub(potential_secend_start,
10204 secend_start - 1) ~= writer.secend_text
10205 then
10206 break
10207 end
10208 secend_start = potential_secend_start
10209 end
10210 -- find an immediately preceding
10211 -- block element / paragraph separator
10212 sep_start = secend_start
10213 potential_sep_start = sep_start - #writer.interblocksep_text
10214 if potential_sep_start >= 1
10215 and output:sub(potential_sep_start,
10216 sep_start - 1) == writer.interblocksep_text
10217 then
10218 sep_start = potential_sep_start
10219 else
10220 potential_sep_start = sep_start - #writer.paragraphsep_text
10221 if potential_sep_start >= 1
10222 and output:sub(potential_sep_start,
10223 sep_start - 1) == writer.paragraphsep_text
10224 then
10225 sep_start = potential_sep_start
10226 end
10227 end
10228 -- remove `writer->undosep` and immediately preceding
10229 -- block element / paragraph separator
10230 output = output:sub(1, sep_start - 1)
10231 .. output:sub(secend_start, undosep_start - 1)
10232 .. output:sub(undosep_end + 1)
10233 end
10234 return output
10235 end
10236 end
10237 return self
10238 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.

```

10239 M.extensions = {}

```

### 3.1.7.1 Bracketed Spans

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

```
10240 M.extensions.bracketed_spans = function()
10241 return {
10242 name = "built-in bracketed_spans syntax extension",
10243 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.

```
10244 function self.span(s, attr)
10245 if self.flatten_inlines then return s end
10246 return {"\\markdownRendererBracketedSpanAttributeContextBegin",
10247 self.attributes(attr),
10248 s,
10249 "\\markdownRendererBracketedSpanAttributeContextEnd{}}"}
10250 end
10251 end, extend_reader = function(self)
10252 local parsers = self.parsers
10253 local writer = self.writer
10254
10255 local span_label = parsers.lbracket
10256 * (Cs((parsers.alphanumeric^1
10257 + parsers.inticks
10258 + parsers.autolink
10259 + V("InlineHtml")
10260 + (parsers.backslash * parsers.backslash)
10261 + (parsers.backslash
10262 * (parsers.lbracket + parsers.rbracket)
10263 + V("Space") + V("Endline")
10264 + (parsers.any
10265 - (parsers.newline
10266 + parsers.lbracket
10267 + parsers.rbracket
10268 + parsers.blankline^2))))^1)
10269 / self.parser_functions.parse_inlines)
10270 * parsers.rbracket
10271
10272 local Span = span_label
10273 * Ct(parsers.attributes)
10274 / writer.span
10275
10276 self.insert_pattern("Inline before LinkAndEmph",
10277 Span, "Span")
10278 end
10279 }
10280 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.

```
10281 M.extensions.citations = function(citation_nbsps)
10282 return {
10283 name = "built-in citations syntax extension",
10284 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.

```
10285 function self.citations(text_cites, cites)
10286 local buffer = {}
10287 if self.flatten_inlines then
10288 for _,cite in ipairs(cites) do
10289 if cite.prenote then
10290 table.insert(buffer, {cite.prenote, " "})
10291 end
10292 table.insert(buffer, cite.name)
10293 if cite.postnote then
10294 table.insert(buffer, {" ", cite.postnote})
10295 end
10296 end
10297 else
10298 table.insert(buffer,
10299 {"\\markdownRenderer",
10300 text_cites and "TextCite" or "Cite",
10301 "{", #cites, "}"})
10302 for _,cite in ipairs(cites) do
10303 table.insert(buffer,
10304 {cite.suppress_author and "-" or "+", "{",
10305 cite.prenote or "", "}{",
```

```

10306 cite.postnote or "", "{", cite.name, "}")
10307 end
10308 end
10309 return buffer
10310 end
10311 end, extend_reader = function(self)
10312 local parsers = self.parsers
10313 local writer = self.writer
10314
10315 local citation_chars
10316 = parsers.alphanumeric
10317 + S("#$%&-+<>~/_")
10318
10319 local citation_name
10320 = Cs(parsers.dash~-1) * parsers.at
10321 * Cs(citation_chars
10322 * (((citation_chars
10323 + parsers.internal_punctuation
10324 - parsers.comma - parsers.semicolon)
10325 * -#((parsers.internal_punctuation
10326 - parsers.comma
10327 - parsers.semicolon)^0
10328 * -(citation_chars
10329 + parsers.internal_punctuation
10330 - parsers.comma
10331 - parsers.semicolon)))^0
10332 * citation_chars)^-1)
10333
10334 local citation_body_prenote
10335 = Cs((parsers.alphanumeric^1
10336 + parsers.bracketed
10337 + parsers.inticks
10338 + parsers.autolink
10339 + V("InlineHtml")
10340 + V("Space") + V("EndlineNoSub")
10341 + (parsers.anyescaped
10342 - (parsers.newline
10343 + parsers.rbracket
10344 + parsers.blankline^2))
10345 - (parsers.spnl
10346 * parsers.dash~-1
10347 * parsers.at))^1)
10348
10349 local citation_body_postnote
10350 = Cs((parsers.alphanumeric^1
10351 + parsers.bracketed
10352 + parsers.inticks

```

```

10353 + parsers.autolink
10354 + V("InlineHtml")
10355 + V("Space") + V("EndlineNoSub")
10356 + (parsers.anyescaped
10357 - (parsers.newline
10358 + parsers.rbracket
10359 + parsers.semicolon
10360 + parsers.blankline^2))
10361 - (parsers.spnl * parsers.rbracket))^1)
10362
10363 local citation_body_chunk
10364 = (citation_body_prenote
10365 * parsers.spnlc_sep
10366 + Cc("")
10367 * parsers.spnlc
10368)
10369 * citation_name
10370 * (parsers.internal_punctuation
10371 - parsers.semicolon)^-1
10372 * (parsers.spnlc / function(_) return end
10373 * citation_body_postnote
10374 + Cc("")
10375 * parsers.spnlc
10376)
10377
10378 local citation_body
10379 = citation_body_chunk
10380 * (parsers.semicolon
10381 * parsers.spnlc
10382 * citation_body_chunk
10383)^0
10384
10385 local citation_headless_body_postnote
10386 = Cs((parsers.alphanumeric^1
10387 + parsers.bracketed
10388 + parsers.inticks
10389 + parsers.autolink
10390 + V("InlineHtml")
10391 + V("Space") + V("Endline")
10392 + (parsers.anyescaped
10393 - (parsers.newline
10394 + parsers.rbracket
10395 + parsers.at
10396 + parsers.semicolon + parsers.blankline^2))
10397 - (parsers.spnl * parsers.rbracket))^0)
10398
10399 local citation_headless_body

```

```

10400 = citation_headless_body_postnote
10401 * (parsers.semicolon
10402 * parsers.spnlc
10403 * citation_body_chunk
10404)^0
10405
10406 local citations
10407 = function(text_cites, raw_cites)
10408 local function normalize(str)
10409 if str == "" then
10410 str = nil
10411 else
10412 str = (citation_nbsps and
10413 self.parser_functions.parse_inlines_nbsp or
10414 self.parser_functions.parse_inlines)(str)
10415 end
10416 return str
10417 end
10418
10419 local cites = {}
10420 for i = 1,#raw_cites,4 do
10421 cites[#cites+1] = {
10422 prenote = normalize(raw_cites[i]),
10423 suppress_author = raw_cites[i+1] == "-",
10424 name = writer.identifier(raw_cites[i+2]),
10425 postnote = normalize(raw_cites[i+3]),
10426 }
10427 end
10428 return writer.citations(text_cites, cites)
10429 end
10430
10431 local TextCitations
10432 = Ct((parsers.spnlc
10433 * Cc("")
10434 * citation_name
10435 * ((parsers.spnlc
10436 * parsers.lbracket
10437 * citation_headless_body
10438 * parsers.rbracket) + Cc("")))^1)
10439 / function(raw_cites)
10440 return citations(true, raw_cites)
10441 end
10442
10443 local ParenthesizedCitations
10444 = Ct((parsers.spnlc
10445 * parsers.lbracket
10446 * citation_body

```

```

10447 * parsers.rbracket)^1)
10448 / function(raw_cites)
10449 return citations(false, raw_cites)
10450 end
10451
10452 local Citations = TextCitations + ParenthesizedCitations
10453
10454 self.insert_pattern("Inline before LinkAndEmph",
10455 Citations, "Citations")
10456
10457 self.add_special_character("@")
10458 self.add_special_character("-")
10459 end
10460 }
10461 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.

```

10462 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.

```

10463 local languages_json = (function()
10464 local base, prev, curr
10465 for _, pathname in ipairs{kpse.lookup(language_map,
10466 {all=true})} do
10467 local file = io.open(pathname, "r")
10468 if not file then goto continue end
10469 local input = assert(file:read("*a"))
10470 assert(file:close())
10471 local json = input:gsub('("[^\\n]-"):', '[%1]=')
10472 curr = load("_ENV = {}; return "..json")()
10473 if type(curr) == "table" then
10474 if base == nil then
10475 base = curr
10476 else
10477 setmetatable(prev, { __index = curr })
10478 prev = curr
10479 end
10480 end
10481 ::continue::
10482 end

```

```

10483 return base or {}
10484 end()()
10485
10486 return {
10487 name = "built-in content_blocks syntax extension",
10488 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.

```

10489 function self.contentblock(src,suf,type,tit)
10490 if not self.is_writing then return "" end
10491 src = src.." "..suf
10492 suf = suf:lower()
10493 if type == "onlineimage" then
10494 return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
10495 "{" ,self.string(src),"} ",
10496 "{" ,self.uri(src),"} ",
10497 "{" ,self.string(tit or "") ,"} "}
10498 elseif languages_json[suf] then
10499 return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
10500 "{" ,self.string(languages_json[suf]) ,"} ",
10501 "{" ,self.string(src),"} ",
10502 "{" ,self.uri(src),"} ",
10503 "{" ,self.string(tit or "") ,"} "}
10504 else
10505 return {"\\markdownRendererContentBlock{" ,suf,"} ",
10506 "{" ,self.string(src),"} ",
10507 "{" ,self.uri(src),"} ",
10508 "{" ,self.string(tit or "") ,"} "}
10509 end
10510 end
10511 end, extend_reader = function(self)
10512 local parsers = self.parsers
10513 local writer = self.writer
10514
10515 local contentblock_tail
10516 = parsers.optionalttitle
10517 * (parsers.newline + parsers.eof)
10518
10519 -- case insensitive online image suffix:
10520 local onlineimagesuffix
10521 = (function(...)
10522 local parser = nil
10523 for _, suffix in ipairs({...}) do
10524 local pattern=nil

```

```

10525 for i=1,#suffix do
10526 local char=suffix:sub(i,i)
10527 char = S(char:lower()..char:upper())
10528 if pattern == nil then
10529 pattern = char
10530 else
10531 pattern = pattern * char
10532 end
10533 end
10534 if parser == nil then
10535 parser = pattern
10536 else
10537 parser = parser + pattern
10538 end
10539 end
10540 return parser
10541 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
10542
10543 -- online image url for iA Writer content blocks with
10544 -- mandatory suffix, allowing nested brackets:
10545 local onlineimageurl
10546 = (parsers.less
10547 * Cs((parsers.anyescaped
10548 - parsers.more
10549 - parsers.spacing
10550 - #(parsers.period
10551 * onlineimagesuffix
10552 * parsers.more
10553 * contentblock_tail))^0)
10554 * parsers.period
10555 * Cs(onlineimagesuffix)
10556 * parsers.more
10557 + (Cs((parsers.inparens
10558 + (parsers.anyescaped
10559 - parsers.spacing
10560 - parsers.rparent
10561 - #(parsers.period
10562 * onlineimagesuffix
10563 * contentblock_tail))))^0)
10564 * parsers.period
10565 * Cs(onlineimagesuffix))
10566) * Cc("onlineimage")
10567
10568 -- filename for iA Writer content blocks with mandatory suffix:
10569 local localfilepath
10570 = parsers.slash
10571 * Cs((parsers.anyescaped

```

```

10572 - parsers.tab
10573 - parsers.newline
10574 - #(parsers.period
10575 * parsers.alphanumeric^1
10576 * contentblock_tail))^1)
10577 * parsers.period
10578 * Cs(parsers.alphanumeric^1)
10579 * Cc("localfile")
10580
10581 local ContentBlock
10582 = parsers.check_trail_no_rem
10583 * (localfilepath + onlineimageurl)
10584 * contentblock_tail
10585 / writer.contentblock
10586
10587 self.insert_pattern("Block before Blockquote",
10588 ContentBlock, "ContentBlock")
10589 end
10590 }
10591 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.

```

10592 M.extensions.definition_lists = function(tight_lists)
10593 return {
10594 name = "built-in definition_lists syntax extension",
10595 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.

```

10596 local function dlitem(term, defs)
10597 local retVal = {"\\markdownRendererDlItem{",term,""}
10598 for _, def in ipairs(defs) do
10599 retVal[#retVal+1]
10600 = {"\\markdownRendererDlDefinitionBegin ",def,
10601 "\\markdownRendererDlDefinitionEnd "}
10602 end
10603 retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
10604 return retVal
10605 end
10606
10607 function self.definitionlist(items,tight)

```



```

10608 if not self.is_writing then return "" end
10609 local buffer = {}
10610 for _,item in ipairs(items) do
10611 buffer[#buffer + 1] = dlitem(item.term, item.definitions)
10612 end
10613 if tight and tight_lists then
10614 return {"\\markdownRendererDlBeginTight\n", buffer,
10615 "\\n\\markdownRendererDlEndTight"}
10616 else
10617 return {"\\markdownRendererDlBegin\n", buffer,
10618 "\\n\\markdownRendererDlEnd"}
10619 end
10620 end
10621 end, extend_reader = function(self)
10622 local parsers = self.parsers
10623 local writer = self.writer
10624
10625 local defstartchar = S("~:")
10626
10627 local defstart
10628 = parsers.check_trail_length(0) * defstartchar
10629 * #parsers.spacing
10630 * (parsers.tab + parsers.space^-3)
10631 + parsers.check_trail_length(1)
10632 * defstartchar * #parsers.spacing
10633 * (parsers.tab + parsers.space^-2)
10634 + parsers.check_trail_length(2)
10635 * defstartchar * #parsers.spacing
10636 * (parsers.tab + parsers.space^-1)
10637 + parsers.check_trail_length(3)
10638 * defstartchar * #parsers.spacing
10639
10640 local indented_line
10641 = (parsers.check_minimal_indent / "")
10642 * parsers.check_code_trail * parsers.line
10643
10644 local blank
10645 = parsers.check_minimal_blank_indent_and_any_trail
10646 * parsers.optionalspace * parsers.newline
10647
10648 local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
10649
10650 local indented_blocks = function(bl)
10651 return Cs(bl
10652 * (blank^1 * (parsers.check_minimal_indent / "")
10653 * parsers.check_code_trail * -parsers.blankline * bl)^0
10654 * (blank^1 + parsers.eof))

```

```

10655 end
10656
10657 local function definition_list_item(term, defs, _)
10658 return { term = self.parser_functions.parse_inlines(term),
10659 definitions = defs }
10660 end
10661
10662 local DefinitionListItemLoose
10663 = C(parsers.line) * blank^0
10664 * Ct((parsers.check_minimal_indent * (defstart
10665 * indented_blocks(dlchunk)
10666 / self.parser_functions.parse_blocks_nested))^1)
10667 * Cc(false) / definition_list_item
10668
10669 local DefinitionListItemTight
10670 = C(parsers.line)
10671 * Ct((parsers.check_minimal_indent * (defstart * dlchunk
10672 / self.parser_functions.parse_blocks_nested))^1)
10673 * Cc(true) / definition_list_item
10674
10675 local DefinitionList
10676 = (Ct(DefinitionListItemLoose^1) * Cc(false)
10677 + Ct(DefinitionListItemTight^1)
10678 * (blank^0
10679 * -DefinitionListItemLoose * Cc(true))
10680) / writer.definitionlist
10681
10682 self.insert_pattern("Block after Heading",
10683 DefinitionList, "DefinitionList")
10684 end
10685 }
10686 end

```

### 3.1.7.5 Fancy Lists

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

```

10687 M.extensions.fancy_lists = function()
10688 return {
10689 name = "built-in fancy_lists syntax extension",
10690 extend_writer = function(self)
10691 local options = self.options
10692

```

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.

```

10693 function self.fancylst(items,tight,startnum,numstyle,numdelim)
10694 if not self.is_writing then return "" end
10695 local buffer = {}
10696 local num = startnum
10697 for _,item in ipairs(items) do
10698 if item ~= "" then
10699 buffer[#buffer + 1] = self.fancyitem(item,num)
10700 end
10701 if num ~= nil and item ~= "" then
10702 num = num + 1
10703 end
10704 end
10705 local contents = util.intersperse(buffer,"\n")
10706 if tight and options.tightLists then
10707 return {"\\markdownRendererFancyOlBeginTight{",
10708 numstyle,"}{",numdelim,"}",contents,
10709 "\\n\\markdownRendererFancyOlEndTight "}
10710 else
10711 return {"\\markdownRendererFancyOlBegin{",
10712 numstyle,"}{",numdelim,"}",contents,
10713 "\\n\\markdownRendererFancyOlEnd "}
10714 end
10715 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.

```

10716 function self.fancyitem(s,num)
10717 if num ~= nil then
10718 return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,

```

```

10719 "\\markdownRendererFancyOlItemEnd "}
10720 else
10721 return {"\\markdownRendererFancyOlItem ",s,
10722 "\\markdownRendererFancyOlItemEnd "}
10723 end
10724 end
10725 end, extend_reader = function(self)
10726 local parsers = self.parsers
10727 local options = self.options
10728 local writer = self.writer
10729
10730 local function combine_markers_and_delims(markers, delims)
10731 local markers_table = {}
10732 for _,marker in ipairs(markers) do
10733 local start_marker
10734 local continuation_marker
10735 if type(marker) == "table" then
10736 start_marker = marker[1]
10737 continuation_marker = marker[2]
10738 else
10739 start_marker = marker
10740 continuation_marker = marker
10741 end
10742 for _,delim in ipairs(delims) do
10743 table.insert(markers_table,
10744 {start_marker, continuation_marker, delim})
10745 end
10746 end
10747 return markers_table
10748 end
10749
10750 local function join_table_with_func(func, markers_table)
10751 local pattern = func(table.unpack(markers_table[1]))
10752 for i = 2, #markers_table do
10753 pattern = pattern + func(table.unpack(markers_table[i]))
10754 end
10755 return pattern
10756 end
10757
10758 local lowercase_letter_marker = R("az")
10759 local uppercase_letter_marker = R("AZ")
10760
10761 local roman_marker = function(chars)
10762 local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
10763 local l, x, v, i
10764 = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
10765 return m^-3

```

```

10766 * (c*m + c*d + d^-1 * c^-3)
10767 * (x*c + x*l + l^-1 * x^-3)
10768 * (i*x + i*v + v^-1 * i^-3)
10769 end
10770
10771 local lowercase_roman_marker
10772 = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10773 local uppercase_roman_marker
10774 = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10775
10776 local lowercase_opening_roman_marker = P("i")
10777 local uppercase_opening_roman_marker = P("I")
10778
10779 local digit_marker = parsers.dig * parsers.dig^-8
10780
10781 local markers = {
10782 {lowercase_opening_roman_marker, lowercase_roman_marker},
10783 {uppercase_opening_roman_marker, uppercase_roman_marker},
10784 lowercase_letter_marker,
10785 uppercase_letter_marker,
10786 lowercase_roman_marker,
10787 uppercase_roman_marker,
10788 digit_marker
10789 }
10790
10791 local delims = {
10792 parsers.period,
10793 parsers.rparent
10794 }
10795
10796 local markers_table = combine_markers_and_delims(markers, delims)
10797
10798 local function enumerator(start_marker, _,
10799 delimiter_type, interrupting)
10800 local delimiter_range
10801 local allowed_end
10802 if interrupting then
10803 delimiter_range = P("1")
10804 allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10805 else
10806 delimiter_range = start_marker
10807 allowed_end = C(parsers.spacechar^1)
10808 + #(parsers.newline + parsers.eof)
10809 end
10810
10811 return parsers.check_trail
10812 * Ct(C(delimiter_range) * C(delimiter_type))

```

```

10813 * allowed_end
10814 end
10815
10816 local starter = join_table_with_func(enumerator, markers_table)
10817
10818 local TightListItem = function(starter)
10819 return parsers.add_indent(starter, "li")
10820 * parsers.indented_content_tight
10821 end
10822
10823 local LooseListItem = function(starter)
10824 return parsers.add_indent(starter, "li")
10825 * parsers.indented_content_loose
10826 * remove_indent("li")
10827 end
10828
10829 local function roman2number(roman)
10830 local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100,
10831 ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
10832 local numeral = 0
10833
10834 local i = 1
10835 local len = string.len(roman)
10836 while i < len do
10837 local z1, z2 = romans[string.sub(roman, i, i)],
10838 romans[string.sub(roman, i+1, i+1)]
10839 if z1 < z2 then
10840 numeral = numeral + (z2 - z1)
10841 i = i + 2
10842 else
10843 numeral = numeral + z1
10844 i = i + 1
10845 end
10846 end
10847 if i <= len then
10848 numeral = numeral + romans[string.sub(roman,i,i)]
10849 end
10850 return numeral
10851 end
10852
10853 local function sniffstyle(numstr, delimend)
10854 local numdelim
10855 if delimend == ")" then
10856 numdelim = "OneParen"
10857 elseif delimend == "." then
10858 numdelim = "Period"
10859 else

```

```

10860 numdelim = "Default"
10861 end
10862
10863 local num
10864 num = numstr:match("^([I])$")
10865 if num then
10866 return roman2number(num), "UpperRoman", numdelim
10867 end
10868 num = numstr:match("^([i])$")
10869 if num then
10870 return roman2number(string.upper(num)), "LowerRoman", numdelim
10871 end
10872 num = numstr:match("^([A-Z])$")
10873 if num then
10874 return string.byte(num) - string.byte("A") + 1,
10875 "UpperAlpha", numdelim
10876 end
10877 num = numstr:match("^([a-z])$")
10878 if num then
10879 return string.byte(num) - string.byte("a") + 1,
10880 "LowerAlpha", numdelim
10881 end
10882 num = numstr:match("^([IVXLCDM]+)")
10883 if num then
10884 return roman2number(num), "UpperRoman", numdelim
10885 end
10886 num = numstr:match("^([ivxlcdm]+)")
10887 if num then
10888 return roman2number(string.upper(num)), "LowerRoman", numdelim
10889 end
10890 return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10891 end
10892
10893 local function fancylist(items,tight,start)
10894 local startnum, numstyle, numdelim
10895 = sniffstyle(start[2][1], start[2][2])
10896 return writer.fancylist(items,tight,
10897 options.startNumber and startnum or 1,
10898 numstyle or "Decimal",
10899 numdelim or "Default")
10900 end
10901
10902 local FancyListOfType
10903 = function(start_marker, continuation_marker, delimiter_type)
10904 local enumerator_start
10905 = enumerator(start_marker, continuation_marker,
10906 delimiter_type)

```

```

10907 local enumerator_cont
10908 = enumerator(continuation_marker, continuation_marker,
10909 delimiter_type)
10910 return Cg(enumerator_start, "listtype")
10911 * (Ct(TightListItem(Cb("listtype"))
10912 * ((parsers.check_minimal_indent / "")
10913 * TightListItem(enumerator_cont))^0)
10914 * Cc(true)
10915 * -#((parsers.conditionally_indented_blankline^0 / "")
10916 * parsers.check_minimal_indent * enumerator_cont)
10917 + Ct(LooseListItem(Cb("listtype"))
10918 * ((parsers.conditionally_indented_blankline^0 / "")
10919 * (parsers.check_minimal_indent / "")
10920 * LooseListItem(enumerator_cont))^0)
10921 * Cc(false)
10922) * Ct(Cb("listtype")) / fancylist
10923 end
10924
10925 local FancyList
10926 = join_table_with_func(FancyListOfType, markers_table)
10927
10928 local Endline = parsers.newline
10929 * (parsers.check_minimal_indent
10930 * -parsers.EndlineExceptions
10931 + parsers.check_optional_indent
10932 * -parsers.EndlineExceptions
10933 * -starter)
10934 * parsers.spacechar^0
10935 / writer.soft_line_break
10936
10937 self.update_rule("OrderedList", FancyList)
10938 self.update_rule("Endline", Endline)
10939 end
10940 }
10941 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.



```

10942 M.extensions.fenced_code = function(blank_before_code_fence,
10943 allow_attributes,
10944 allow_raw_blocks)
10945 return {
10946 name = "built-in fenced_code syntax extension",
10947 extend_writer = function(self)
10948 local options = self.options
10949

```

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.

```

10950 function self.fencedCode(s, i, attr)
10951 if not self.is_writing then return "" end
10952 s = s:gsub("\n$", "")
10953 local buf = {}
10954 if attr ~= nil then
10955 table.insert(buf,
10956 {"\\markdownRendererFencedCodeAttributeContextBegin",
10957 self.attributes(attr)})
10958 end
10959 local name = util.cache_verbatim(options.cacheDir, s)
10960 table.insert(buf,
10961 {"\\markdownRendererInputFencedCode{",
10962 name,"}{",self.string(i),"}{",self.infostring(i),"}")})
10963 if attr ~= nil then
10964 table.insert(buf,
10965 "\\markdownRendererFencedCodeAttributeContextEnd{")
10966 end
10967 return buf
10968 end
10969

```

Define **writer->rawBlock** as a function that will transform an input raw block **s** with the raw attribute **attr** to the output format.

```

10970 if allow_raw_blocks then
10971 function self.rawBlock(s, attr)
10972 if not self.is_writing then return "" end
10973 s = s:gsub("\n$", "")
10974 local name = util.cache_verbatim(options.cacheDir, s)
10975 return {"\\markdownRendererInputRawBlock{",
10976 name,"}{", self.string(attr),"}")
10977 end
10978 end
10979 end, extend_reader = function(self)
10980 local parsers = self.parsers
10981 local writer = self.writer
10982
10983 local function captures_geq_length(_,i,a,b)

```

```

10984 return #a >= #b and i
10985 end
10986
10987 local function strip_enclosing_whitespaces(str)
10988 return str:gsub("^%s*(.)%s*$", "%1")
10989 end
10990
10991 local tilde_infostring = Cs(Cs((V("HtmlEntity")
10992 + parsers.anyescaped
10993 - parsers.newline)^0)
10994 / strip_enclosing_whitespaces)
10995
10996 local backtick_infostring
10997 = Cs(Cs((V("HtmlEntity")
10998 + (-#(parsers.backslash * parsers.backtick)
10999 * parsers.anyescaped
11000 - parsers.newline
11001 - parsers.backtick)^0)
11002 / strip_enclosing_whitespaces)
11003
11004 local fenceindent
11005
11006 local function has_trail(indent_table)
11007 return indent_table ~= nil and
11008 indent_table.trail ~= nil and
11009 next(indent_table.trail) ~= nil
11010 end
11011
11012 local function has_indents(indent_table)
11013 return indent_table ~= nil and
11014 indent_table.indents ~= nil and
11015 next(indent_table.indents) ~= nil
11016 end
11017
11018 local function get_last_indent_name(indent_table)
11019 if has_indents(indent_table) then
11020 return indent_table.indents[#indent_table.indents].name
11021 end
11022 end
11023
11024 local count_fenced_start_indent =
11025 function(_, _, indent_table, trail)
11026 local last_indent_name = get_last_indent_name(indent_table)
11027 fenceindent = 0
11028 if last_indent_name ~= "li" then
11029 fenceindent = #trail
11030 end

```

```

11031 return true
11032 end
11033
11034 local fencehead = function(char, infostring)
11035 return Cmt(Cb("indent_info")
11036 * parsers.check_trail, count_fenced_start_indent)
11037 * Cg(char^3, "fencelength")
11038 * parsers.optionalspace
11039 * infostring
11040 * (parsers.newline + parsers.eof)
11041 end
11042
11043 local fencetail = function(char)
11044 return parsers.check_trail_no_rem
11045 * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
11046 * parsers.optionalspace * (parsers.newline + parsers.eof)
11047 + parsers.eof
11048 end
11049
11050 local process_fenced_line =
11051 function(s, i, -- luacheck: ignore s i
11052 indent_table, line_content, is_blank)
11053 local remainder = ""
11054 if has_trail(indent_table) then
11055 remainder = indent_table.trail.internal_remainder
11056 end
11057
11058 if is_blank
11059 and get_last_indent_name(indent_table) == "li" then
11060 remainder = ""
11061 end
11062
11063 local str = remainder .. line_content
11064 local index = 1
11065 local remaining = fenceindent
11066
11067 while true do
11068 local c = str:sub(index, index)
11069 if c == " " and remaining > 0 then
11070 remaining = remaining - 1
11071 index = index + 1
11072 elseif c == "\t" and remaining > 3 then
11073 remaining = remaining - 4
11074 index = index + 1
11075 else
11076 break
11077 end

```

```

11078 end
11079
11080 return true, str:sub(index)
11081 end
11082
11083 local fencedline = function(char)
11084 return Cmt(Cb("indent_info")
11085 * C(parsers.line - fencetail(char))
11086 * Cc(false), process_fenced_line)
11087 end
11088
11089 local blankfencedline
11090 = Cmt(Cb("indent_info")
11091 * C(parsers.blankline)
11092 * Cc(true), process_fenced_line)
11093
11094 local TildeFencedCode
11095 = fencehead(parsers.tilde, tilde_infostring)
11096 * Cs(((parsers.check_minimal_blank_indent / "")
11097 * blankfencedline
11098 + (parsers.check_minimal_indent / "")
11099 * fencedline(parsers.tilde))^0)
11100 * ((parsers.check_minimal_indent / "")
11101 * fencetail(parsers.tilde) + parsers.succeed)
11102
11103 local BacktickFencedCode
11104 = fencehead(parsers.backtick, backtick_infostring)
11105 * Cs(((parsers.check_minimal_blank_indent / "")
11106 * blankfencedline
11107 + (parsers.check_minimal_indent / "")
11108 * fencedline(parsers.backtick))^0)
11109 * ((parsers.check_minimal_indent / "")
11110 * fencetail(parsers.backtick) + parsers.succeed)
11111
11112 local infostring_with_attributes
11113 = Ct(C((parsers.linechar
11114 - (parsers.optionalspace
11115 * parsers.attributes))^0)
11116 * parsers.optionalspace
11117 * Ct(parsers.attributes))
11118
11119 local FencedCode
11120 = ((TildeFencedCode + BacktickFencedCode)
11121 / function(infostring, code)
11122 local expanded_code = self.expandtabs(code)
11123
11124 if allow_raw_blocks then

```

```

11125 local raw_attr = lpeg.match(parsers.raw_attribute,
11126 infostring)
11127 if raw_attr then
11128 return writer.rawBlock(expanded_code, raw_attr)
11129 end
11130 end
11131
11132 local attr = nil
11133 if allow_attributes then
11134 local match = lpeg.match(infostring_with_attributes,
11135 infostring)
11136 if match then
11137 infostring, attr = table.unpack(match)
11138 end
11139 end
11140 return writer.fencedCode(expanded_code, infostring, attr)
11141 end)
11142
11143 self.insert_pattern("Block after Verbatim",
11144 FencedCode, "FencedCode")
11145
11146 local fencestart
11147 if blank_before_code_fence then
11148 fencestart = parsers.fail
11149 else
11150 fencestart = fencehead(parsers.backtick, backtick_infostring)
11151 + fencehead(parsers.tilde, tilde_infostring)
11152 end
11153
11154 self.update_rule("EndlineExceptions", function(previous_pattern)
11155 if previous_pattern == nil then
11156 previous_pattern = parsers.EndlineExceptions
11157 end
11158 return previous_pattern + fencestart
11159 end)
11160
11161 self.add_special_character("~")
11162 self.add_special_character("~")
11163 end
11164 }
11165 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.

```
11166 M.extensions.fenced_divs = function(blank_before_div_fence)
11167 return {
11168 name = "built-in fenced_divs syntax extension",
11169 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.

```
11170 function self.div_begin(attributes)
11171 local start_output
11172 = {"\\markdownRendererFencedDivAttributeContextBegin\\n",
11173 self.attributes(attributes)}
11174 local end_output
11175 = {"\\markdownRendererFencedDivAttributeContextEnd{\\n"}
11176 return self.push_attributes(
11177 "div", attributes, start_output, end_output)
11178 end
```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```
11179 function self.div_end()
11180 return self.pop_attributes("div")
11181 end
11182 end, extend_reader = function(self)
11183 local parsers = self.parsers
11184 local writer = self.writer
```

Define basic patterns for matching the opening and the closing tag of a div.

```
11185 local fenced_div_infostring
11186 = C((parsers.linechar
11187 - (parsers.spacechar^1
11188 * parsers.colon^1))^1)
11189
11190 local fenced_div_begin = parsers.nonindentSPACE
11191 * parsers.colon^3
11192 * parsers.optionalspace
11193 * fenced_div_infostring
11194 * (parsers.spacechar^1
11195 * parsers.colon^1)^0
11196 * parsers.optionalspace
11197 * (parsers.newline + parsers.eof)
11198
11199 local fenced_div_end = parsers.nonindentSPACE
11200 * parsers.colon^3
11201 * parsers.optionalspace
11202 * (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.

```

11203 self.initialize_named_group("fenced_div_level", "0")
11204 self.initialize_named_group("fenced_div_num_opening_indents")
11205
11206 local function increment_div_level()
11207 local push_indent_table =
11208 function(s, i, indent_table, -- luacheck: ignore s i
11209 fenced_div_num_opening_indents, fenced_div_level)
11210 fenced_div_level = tonumber(fenced_div_level) + 1
11211 local num_opening_indents = 0
11212 if indent_table.indents ~= nil then
11213 num_opening_indents = #indent_table.indents
11214 end
11215 fenced_div_num_opening_indents[fenced_div_level]
11216 = num_opening_indents
11217 return true, fenced_div_num_opening_indents
11218 end
11219
11220 local increment_level =
11221 function(s, i, fenced_div_level) -- luacheck: ignore s i
11222 fenced_div_level = tonumber(fenced_div_level) + 1
11223 return true, tostring(fenced_div_level)
11224 end
11225
11226 return Cg(Cmt(Cb("indent_info")
11227 * Cb("fenced_div_num_opening_indents")
11228 * Cb("fenced_div_level"), push_indent_table)
11229 , "fenced_div_num_opening_indents")
11230 * Cg(Cmt(Cb("fenced_div_level"), increment_level)
11231 , "fenced_div_level")
11232 end
11233
11234 local function decrement_div_level()
11235 local pop_indent_table =
11236 function(s, i, -- luacheck: ignore s i
11237 fenced_div_indent_table, fenced_div_level)
11238 fenced_div_level = tonumber(fenced_div_level)
11239 fenced_div_indent_table[fenced_div_level] = nil
11240 return true, tostring(fenced_div_level - 1)
11241 end
11242
```

```

11243 return Cg(Cmt(Cb("fenced_div_num_opening_indents")
11244 * Cb("fenced_div_level"), pop_indent_table)
11245 , "fenced_div_level")
11246 end
11247
11248
11249 local non_fenced_div_block
11250 = parsers.check_minimal_indent * V("Block")
11251 - parsers.check_minimal_indent_and_trail * fenced_div_end
11252
11253 local non_fenced_div_paragraph
11254 = parsers.check_minimal_indent * V("Paragraph")
11255 - parsers.check_minimal_indent_and_trail * fenced_div_end
11256
11257 local blank = parsers.minimally_indented_blank
11258
11259 local block_separated = parsers.block_sep_group(blank)
11260 * non_fenced_div_block
11261
11262 local loop_body_pair
11263 = parsers.create_loop_body_pair(block_separated,
11264 non_fenced_div_paragraph,
11265 parsers.block_sep_group(blank),
11266 parsers.par_sep_group(blank))
11267
11268 local content_loop = (non_fenced_div_block
11269 * loop_body_pair.block^0
11270 + non_fenced_div_paragraph
11271 * block_separated
11272 * loop_body_pair.block^0
11273 + non_fenced_div_paragraph
11274 * loop_body_pair.par^0)
11275 * blank^0
11276
11277 local FencedDiv = fenced_div_begin
11278 / function (infostring)
11279 local attr
11280 = lpeg.match(Ct(parsers.attributes),
11281 infostring)
11282 if attr == nil then
11283 attr = {"." .. infostring}
11284 end
11285 return attr
11286 end
11287 / writer.div_begin
11288 * increment_div_level()
11289 * parsers.skipblanklines

```



```

11290 * Ct(content_loop)
11291 * parsers.minimally_indented_blank^0
11292 * parsers.check_minimal_indent_and_trail
11293 * fenced_div_end
11294 * decrement_div_level()
11295 * (Cc("") / writer.div_end)
11296
11297 self.insert_pattern("Block after Verbatim",
11298 FencedDiv, "FencedDiv")
11299
11300 self.add_special_character(":".")
11301

```

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.

```

11302 local function is_inside_div()
11303 local check_div_level =
11304 function(s, i, fenced_div_level) -- luacheck: ignore s i
11305 fenced_div_level = tonumber(fenced_div_level)
11306 return fenced_div_level > 0
11307 end
11308
11309 return Cmt(Cb("fenced_div_level"), check_div_level)
11310 end
11311
11312 local function check_indent()
11313 local compare_indent =
11314 function(s, i, indent_table, -- luacheck: ignore s i
11315 fenced_div_num_opening_indents, fenced_div_level)
11316 fenced_div_level = tonumber(fenced_div_level)
11317 local num_current_indents
11318 = (indent_table.current_line_indents ~= nil and
11319 #indent_table.current_line_indents) or 0
11320 local num_opening_indents
11321 = fenced_div_num_opening_indents[fenced_div_level]
11322 return num_current_indents == num_opening_indents
11323 end
11324
11325 return Cmt(Cb("indent_info")
11326 * Cb("fenced_div_num_opening_indents")
11327 * Cb("fenced_div_level"), compare_indent)
11328 end
11329
11330 local fencestart = is_inside_div()
11331 * fenced_div_end
11332 * check_indent()

```

```

11333
11334 if not blank_before_div_fence then
11335 self.update_rule("EndlineExceptions", function(previous_pattern)
11336 if previous_pattern == nil then
11337 previous_pattern = parsers.EndlineExceptions
11338 end
11339 return previous_pattern + fencestart
11340 end)
11341 end
11342 end
11343 }
11344 end

```

### 3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

11345 M.extensions.header_attributes = function()
11346 return {
11347 name = "built-in header_attributes syntax extension",
11348 extend_writer = function()
11349 end, extend_reader = function(self)
11350 local parsers = self.parsers
11351 local writer = self.writer
11352
11353 local function strip_atx_end(s)
11354 return s:gsub("%s+##%s*$", "")
11355 end
11356
11357 local AtxHeading = Cg(parsers.heading_start, "level")
11358 * parsers.optionalspace
11359 * (C(((parsers.linechar
11360 - (parsers.attributes
11361 * parsers.optionalspace
11362 * parsers.newline))
11363 * (parsers.linechar
11364 - parsers.lbrace)^0)^1)
11365 / strip_atx_end
11366 / parsers.parse_heading_text)
11367 * Cg(Ct(parsers.newline
11368 + (parsers.attributes
11369 * parsers.optionalspace
11370 * parsers.newline)), "attributes")
11371 * Cb("level")
11372 * Cb("attributes")
11373 / writer.heading
11374

```

```

11375 local function strip_trailing_spaces(s)
11376 return s:gsub("%s*$", "")
11377 end
11378
11379 local heading_line = (parsers.linechar
11380 - (parsers.attributes
11381 * parsers.optionalspace
11382 * parsers.newline))^1
11383 - parsers.thematic_break_lines
11384
11385 local heading_text
11386 = heading_line
11387 * ((V("Endline") / "\n")
11388 * (heading_line - parsers.heading_level))^0
11389 * parsers.newline^-1
11390
11391 local SettextHeading
11392 = parsers.freeze_trail * parsers.check_trail_no_rem
11393 * #(heading_text
11394 * (parsers.attributes
11395 * parsers.optionalspace
11396 * parsers.newline)^-1
11397 * parsers.check_minimal_indent
11398 * parsers.check_trail
11399 * parsers.heading_level)
11400 * Cs(heading_text) / strip_trailing_spaces
11401 / parsers.parse_heading_text
11402 * Cg(Ct((parsers.attributes
11403 * parsers.optionalspace
11404 * parsers.newline)^-1), "attributes")
11405 * parsers.check_minimal_indent_and_trail * parsers.heading_level
11406 * Cb("attributes")
11407 * parsers.newline
11408 * parsers.unfreeze_trail
11409 / writer.heading
11410
11411 local Heading = AtxHeading + SettextHeading
11412 self.update_rule("Heading", Heading)
11413 end
11414 }
11415 end

```

### 3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

11416 M.extensions.inline_code_attributes = function()

```

```

11417 return {
11418 name = "built-in inline_code_attributes syntax extension",
11419 extend_writer = function()
11420 end, extend_reader = function(self)
11421 local writer = self.writer
11422
11423 local CodeWithAttributes = parsers.inticks
11424 * Ct(parsers.attributes)
11425 / writer.code
11426
11427 self.insert_pattern("Inline before Code",
11428 CodeWithAttributes,
11429 "CodeWithAttributes")
11430 end
11431 }
11432 end

```

### 3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

11433 M.extensions.line_blocks = function()
11434 return {
11435 name = "built-in line_blocks syntax extension",
11436 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.

```

11437 function self.lineblock(lines)
11438 if not self.is_writing then return "" end
11439 local buffer = {}
11440 for i = 1, #lines - 1 do
11441 buffer[#buffer + 1] = { lines[i], self.hard_line_break }
11442 end
11443 buffer[#buffer + 1] = lines[#lines]
11444
11445 return {"\\markdownRendererLineBlockBegin\\n"
11446 ,buffer,
11447 "\\n\\markdownRendererLineBlockEnd "}
11448 end
11449 end, extend_reader = function(self)
11450 local parsers = self.parsers
11451 local writer = self.writer
11452
11453 local LineBlock
11454 = Ct((Cs((parsers.pipe * parsers.space) / ""
11455 * ((parsers.space)/entities.char_entity("nbsp"))^0
11456 * parsers.linechar^0 * (parsers.newline/""))

```

```

11457 * (-parsers.pipe
11458 * (parsers.space^1/" ")
11459 * parsers.linechar^1
11460 * (parsers.newline/"")
11461)^0
11462 * (parsers.blankline/"")^0)
11463 / self.parser_functions.parse_inlines)^1)
11464 / writer.lineblock
11465
11466 self.insert_pattern("Block after Blockquote",
11467 LineBlock, "LineBlock")
11468 end
11469 }
11470 end

```

### 3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

11471 M.extensions.mark = function()
11472 return {
11473 name = "built-in mark syntax extension",
11474 extend_writer = function(self)

```

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

```

11475 function self.mark(s)
11476 if self.flatten_inlines then return s end
11477 return {"\\markdownRendererMark{" , s, "}"}
11478 end
11479 end, extend_reader = function(self)
11480 local parsers = self.parsers
11481 local writer = self.writer
11482
11483 local doubleequals = P("==")
11484
11485 local Mark
11486 = parsers.between(V("Inline"), doubleequals, doubleequals)
11487 / function (inlines) return writer.mark(inlines) end
11488
11489 self.add_special_character("=")
11490 self.insert_pattern("Inline before LinkAndEmph",
11491 Mark, "Mark")
11492 end
11493 }
11494 end

```

### 3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
11495 M.extensions.link_attributes = function()
11496 return {
11497 name = "built-in link_attributes syntax extension",
11498 extend_writer = function()
11499 end, extend_reader = function(self)
11500 local parsers = self.parsers
11501 local options = self.options
11502
```

The following patterns define link reference definitions with attributes.

```
11503 local define_reference_parser
11504 = (parsers.check_trail / "")
11505 * parsers.link_label
11506 * parsers.colon
11507 * parsers.spnlc * parsers.url
11508 * (parsers.spnlc_sep * parsers.title
11509 * (parsers.spnlc * Ct(parsers.attributes))
11510 * parsers.only_blank
11511 + parsers.spnlc_sep * parsers.title * parsers.only_blank
11512 + Cc("") * (parsers.spnlc * Ct(parsers.attributes))
11513 * parsers.only_blank
11514 + Cc("") * parsers.only_blank)
11515
11516 local ReferenceWithAttributes = define_reference_parser
11517 / self.register_link
11518
11519 self.update_rule("Reference", ReferenceWithAttributes)
11520
```

The following patterns define direct and indirect links with attributes.

```
11521
11522 local LinkWithAttributesAndEmph
11523 = Ct(parsers.link_and_emph_table * Cg(Cc(true),
11524 "match_link_attributes"))
11525 / self.defer_link_and_emphasis_processing
11526
11527 self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
11528
```

The following patterns define autolinks with attributes.

```
11529 local AutoLinkUrlWithAttributes
11530 = parsers.auto_link_url
11531 * Ct(parsers.attributes)
11532 / self.auto_link_url
11533
11534 self.insert_pattern("Inline before AutoLinkUrl",
```

```

11535 AutoLinkUrlWithAttributes,
11536 "AutoLinkUrlWithAttributes")
11537
11538 local AutoLinkEmailWithAttributes
11539 = parsers.auto_link_email
11540 * Ct(parsers.attributes)
11541 / self.auto_link_email
11542
11543 self.insert_pattern("Inline before AutoLinkEmail",
11544 AutoLinkEmailWithAttributes,
11545 "AutoLinkEmailWithAttributes")
11546
11547 if options.relativeReferences then
11548
11549 local AutoLinkRelativeReferenceWithAttributes
11550 = parsers.auto_link_relative_reference
11551 * Ct(parsers.attributes)
11552 / self.auto_link_url
11553
11554 self.insert_pattern(
11555 "Inline before AutoLinkRelativeReference",
11556 AutoLinkRelativeReferenceWithAttributes,
11557 "AutoLinkRelativeReferenceWithAttributes")
11558
11559 end
11560
11561 end
11562 }
11563 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.

```

11564 M.extensions.notes = function(notes, inline_notes)
11565 assert(notes or inline_notes)
11566 return {
11567 name = "built-in notes syntax extension",
11568 extend_writer = function(self)

```

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

```

11569 function self.note(s)
11570 if self.flatten_inlines then return "" end
11571 return {"\\markdownRendererNote{",s,""}

```

```

11572 end
11573 end, extend_reader = function(self)
11574 local parsers = self.parsers
11575 local writer = self.writer
11576
11577 local rawnotes = parsers.rawnotes
11578
11579 if inline_notes then
11580 local InlineNote
11581 = parsers.circumflex
11582 * (parsers.link_label
11583 / self.parser_functions.parse_inlines_no_inline_note)
11584 / writer.note
11585
11586 self.insert_pattern("Inline after LinkAndEmph",
11587 InlineNote, "InlineNote")
11588 end
11589 if notes then
11590 local function strip_first_char(s)
11591 return s:sub(2)
11592 end
11593
11594 local RawNoteRef
11595 = #(parsers.lbracket * parsers.circumflex)
11596 * parsers.link_label / strip_first_char
11597
11598 -- like indirect_link
11599 local function lookup_note(ref)
11600 return writer.defer_call(function()
11601 local found = rawnotes[self.normalize_tag(ref)]
11602 if found then
11603 return writer.note(
11604 self.parser_functions.parse_blocks_nested(found))
11605 else
11606 return {"[",
11607 self.parser_functions.parse_inlines("^" .. ref), "]" }
11608 end
11609 end)
11610 end
11611
11612 local function register_note(ref, rawnote)
11613 local normalized_tag = self.normalize_tag(ref)
11614 if rawnotes[normalized_tag] == nil then
11615 rawnotes[normalized_tag] = rawnote
11616 end
11617 return ""
11618 end

```



```

11619
11620 local NoteRef = RawNoteRef / lookup_note
11621
11622 local optionally_indented_line
11623 = parsers.check_optional_indent_and_any_trail * parsers.line
11624
11625 local blank
11626 = parsers.check_optional_blank_indent_and_any_trail
11627 * parsers.optionalspace * parsers.newline
11628
11629 local chunk
11630 = Cs(parsers.line
11631 * (optionally_indented_line - blank)^0)
11632
11633 local indented_blocks = function(bl)
11634 return Cs(bl
11635 * (blank^1 * (parsers.check_optional_indent / "")
11636 * parsers.check_code_trail
11637 * -parsers.blankline * bl)^0)
11638 end
11639
11640 local NoteBlock
11641 = parsers.check_trail_no_rem
11642 * RawNoteRef * parsers.colon
11643 * parsers.spnlc * indented_blocks(chunk)
11644 / register_note
11645
11646 local Reference = NoteBlock + parsers.Reference
11647
11648 self.update_rule("Reference", Reference)
11649 self.insert_pattern("Inline before LinkAndEmph",
11650 NoteRef, "NoteRef")
11651 end
11652
11653 self.add_special_character("^")
11654 end
11655 }
11656 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.

```
11657 M.extensions.pipe_tables = function(table_captions, table_attributes)
11658
11659 local function make_pipe_table_rectangular(rows)
11660 local num_columns = #rows[2]
11661 local rectangular_rows = {}
11662 for i = 1, #rows do
11663 local row = rows[i]
11664 local rectangular_row = {}
11665 for j = 1, num_columns do
11666 rectangular_row[j] = row[j] or ""
11667 end
11668 table.insert(rectangular_rows, rectangular_row)
11669 end
11670 return rectangular_rows
11671 end
11672
11673 local function pipe_table_row(allow_empty_first_column
11674 , nonempty_column
11675 , column_separator
11676 , column)
11677 local row_beginning
11678 if allow_empty_first_column then
11679 row_beginning = -- empty first column
11680 #(parsers.spacechar^4
11681 * column_separator)
11682 * parsers.optionalspace
11683 * column
11684 * parsers.optionalspace
11685 -- non-empty first column
11686 + parsers.nonindentSPACE
11687 * nonempty_column^-1
11688 * parsers.optionalspace
11689 else
11690 row_beginning = parsers.nonindentSPACE
11691 * nonempty_column^-1
11692 * parsers.optionalspace
11693 end
11694
11695 return Ct(row_beginning
11696 * (-- single column with no leading pipes
11697 #(column_separator
11698 * parsers.optionalspace
11699 * parsers.newline)
11700 * column_separator
11701 * parsers.optionalspace
```

```

11702 -- single column with leading pipes or
11703 -- more than a single column
11704 + (column_separator
11705 * parsers.optionalspace
11706 * column
11707 * parsers.optionalspace)^1
11708 * (column_separator
11709 * parsers.optionalspace)^-1))
11710 end
11711
11712 return {
11713 name = "built-in pipe_tables syntax extension",
11714 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.

```

11715 function self.table(rows, caption, attributes)
11716 if not self.is_writing then return "" end
11717 local buffer = {}
11718 if attributes ~= nil then
11719 table.insert(buffer,
11720 "\\markdownRendererTableAttributeContextBegin\\n")
11721 table.insert(buffer, self.attributes(attributes))
11722 end
11723 table.insert(buffer,
11724 {"\\markdownRendererTable{",
11725 caption or "", "}{" , #rows - 1, "}{" ,
11726 #rows[1], "}")
11727 local temp = rows[2] -- put alignments on the first row
11728 rows[2] = rows[1]
11729 rows[1] = temp
11730 for i, row in ipairs(rows) do
11731 table.insert(buffer, "{")
11732 for _, column in ipairs(row) do
11733 if i > 1 then -- do not use braces for alignments
11734 table.insert(buffer, "{")
11735 end
11736 table.insert(buffer, column)
11737 if i > 1 then
11738 table.insert(buffer, "}")
11739 end
11740 end
11741 table.insert(buffer, "}")
11742 end
11743 if attributes ~= nil then
11744 table.insert(buffer,

```

```

11745 "\\markdownRendererTableAttributeContextEnd{}")
11746 end
11747 return buffer
11748 end
11749 end, extend_reader = function(self)
11750 local parsers = self.parsers
11751 local writer = self.writer
11752
11753 local table_hline_separator = parsers.pipe + parsers.plus
11754
11755 local table_hline_column = (parsers.dash
11756 - #(parsers.dash
11757 * (parsers.spacechar
11758 + table_hline_separator
11759 + parsers.newline)))^1
11760 * (parsers.colon * Cc("r")
11761 + parsers.dash * Cc("d"))
11762 + parsers.colon
11763 * (parsers.dash
11764 - #(parsers.dash
11765 * (parsers.spacechar
11766 + table_hline_separator
11767 + parsers.newline)))^1
11768 * (parsers.colon * Cc("c")
11769 + parsers.dash * Cc("l"))
11770
11771 local table_hline = pipe_table_row(false
11772 , table_hline_column
11773 , table_hline_separator
11774 , table_hline_column)
11775
11776 local table_caption_beginning
11777 = (parsers.check_minimal_blank_indent_and_any_trail_no_rem
11778 * parsers.optionalspace * parsers.newline)^0
11779 * parsers.check_minimal_indent_and_trail
11780 * (P("Table")^-1 * parsers.colon)
11781 * parsers.optionalspace
11782
11783 local function strip_trailing_spaces(s)
11784 return s:gsub("%s*$", "")
11785 end
11786
11787 local table_row
11788 = pipe_table_row(true
11789 , (C((parsers.linechar - parsers.pipe)^1)
11790 / strip_trailing_spaces
11791 / self.parser_functions.parse_inlines)

```

```

11792 , parsers.pipe
11793 , (C((parsers.linechar - parsers.pipe)^0)
11794 / strip_trailing_spaces
11795 / self.parser_functions.parse_inlines))
11796
11797 local table_caption
11798 if table_captions then
11799 table_caption = #table_caption_beginning
11800 * table_caption_beginning
11801 if table_attributes then
11802 table_caption = table_caption
11803 * (C((((parsers.linechar
11804 - (parsers.attributes
11805 * parsers.optionalspace
11806 * parsers.newline
11807 * -(parsers.optionalspace
11808 * parsers.linechar)))
11809 + (parsers.newline
11810 * #(parsers.optionalspace
11811 * parsers.linechar)
11812 * C(parsers.optionalspace)
11813 / writer.space))
11814 * (parsers.linechar
11815 - parsers.lbrace)^0)^1)
11816 / self.parser_functions.parse_inlines)
11817 * (parsers.newline
11818 + (Ct(parsers.attributes)
11819 * parsers.optionalspace
11820 * parsers.newline))
11821 else
11822 table_caption = table_caption
11823 * C((parsers.linechar
11824 + (parsers.newline
11825 * #(parsers.optionalspace
11826 * parsers.linechar)
11827 * C(parsers.optionalspace)
11828 / writer.space))^1)
11829 / self.parser_functions.parse_inlines
11830 * parsers.newline
11831 end
11832 else
11833 table_caption = parsers.fail
11834 end
11835
11836 local PipeTable
11837 = Ct(table_row * parsers.newline
11838 * (parsers.check_minimal_indent_and_trail / {}))

```

```

11839 * table_hline * parsers.newline
11840 * ((parsers.check_minimal_indent / {})
11841 * table_row * parsers.newline)^0)
11842 / make_pipe_table_rectangular
11843 * table_caption^-1
11844 / writer.table
11845
11846 self.insert_pattern("Block after Blockquote",
11847 PipeTable, "PipeTable")
11848 end
11849 }
11850 end

```

### 3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

11851 M.extensions.raw_inline = function()
11852 return {
11853 name = "built-in raw_inline syntax extension",
11854 extend_writer = function(self)
11855 local options = self.options
11856

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

11857 function self.rawInline(s, attr)
11858 if not self.is_writing then return "" end
11859 if self.flatten_inlines then return s end
11860 local name = util.cache_verbatim(options.cacheDir, s)
11861 return {"\\markdownRendererInputRawInline{",
11862 name,"}{", self.string(attr),"}"}
11863 end
11864 end, extend_reader = function(self)
11865 local writer = self.writer
11866
11867 local RawInline = parsers.inticks
11868 * parsers.raw_attribute
11869 / writer.rawInline
11870
11871 self.insert_pattern("Inline before Code",
11872 RawInline, "RawInline")
11873 end
11874 }
11875 end

```

### 3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```
11876 M.extensions.strike_through = function()
11877 return {
11878 name = "built-in strike_through syntax extension",
11879 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.

```
11880 function self.strike_through(s)
11881 if self.flatten_inlines then return s end
11882 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
11883 end
11884 end, extend_reader = function(self)
11885 local parsers = self.parsers
11886 local writer = self.writer
11887
11888 local StrikeThrough = (
11889 parsers.between(parsers.Inline, parsers.doubletildes,
11890 parsers.doubletildes)
11891) / writer.strike_through
11892
11893 self.insert_pattern("Inline after LinkAndEmph",
11894 StrikeThrough, "StrikeThrough")
11895
11896 self.add_special_character("~")
11897 end
11898 }
11899 end
```

### 3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```
11900 M.extensions.subscripts = function()
11901 return {
11902 name = "built-in subscripts syntax extension",
11903 extend_writer = function(self)
```

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

```
11904 function self.subscript(s)
11905 if self.flatten_inlines then return s end
11906 return {"\\markdownRendererSubscript{" ,s,"}"}
11907 end
11908 end, extend_reader = function(self)
11909 local parsers = self.parsers
```

```

11910 local writer = self.writer
11911
11912 local Subscript = (
11913 parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
11914) / writer.subscript
11915
11916 self.insert_pattern("Inline after LinkAndEmph",
11917 Subscript, "Subscript")
11918
11919 self.add_special_character("~")
11920 end
11921 }
11922 end

```

### 3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

11923 M.extensions.superscripts = function()
11924 return {
11925 name = "built-in superscripts syntax extension",
11926 extend_writer = function(self)

```

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

```

11927 function self.superscript(s)
11928 if self.flatten_inlines then return s end
11929 return {"\\markdownRendererSuperscript{" ,s,""} }
11930 end
11931 end, extend_reader = function(self)
11932 local parsers = self.parsers
11933 local writer = self.writer
11934
11935 local Superscript = (
11936 parsers.between(parsers.Str, parsers.circumflex,
11937 parsers.circumflex)
11938) / writer.superscript
11939
11940 self.insert_pattern("Inline after LinkAndEmph",
11941 Superscript, "Superscript")
11942
11943 self.add_special_character("^")
11944 end
11945 }
11946 end

```

### 3.1.7.19 T<sub>E</sub>X Math



The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
11947 M.extensions.tex_math = function(tex_math_dollars,
11948 tex_math_single_backslash,
11949 tex_math_double_backslash)
11950 return {
11951 name = "built-in tex_math syntax extension",
11952 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.

```
11953 function self.display_math(s)
11954 if self.flatten_inlines then return s end
11955 return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
11956 end
```

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

```
11957 function self.inline_math(s)
11958 if self.flatten_inlines then return s end
11959 return {"\\markdownRendererInlineMath{",self.math(s),"}"}
11960 end
11961 end, extend_reader = function(self)
11962 local parsers = self.parsers
11963 local writer = self.writer
11964
11965 local function between(p, starter, ender)
11966 return (starter * Cs(p * (p - ender)^0) * ender)
11967 end
11968
11969 local function strip_preceding_spaces(str)
11970 return str:gsub("^%s*(.-)$", "%1")
11971 end
11972
11973 local allowed_before_closing
11974 = B(parsers.backslash * parsers.any
11975 + parsers.any * (parsers.any - parsers.backslash))
11976
11977 local allowed_before_closing_no_space
11978 = B(parsers.backslash * parsers.any
11979 + parsers.any * (parsers.nonspacechar - parsers.backslash))
11980
```

The following patterns implement the Pandoc dollar math syntax extension.

```
11981 local dollar_math_content
11982 = (parsers.newline * (parsers.check_optional_indent / "")
11983 + parsers.backslash^-1
11984 * parsers.linechar)
```

```

11985 - parsers.blankline^2
11986 - parsers.dollar
11987
11988 local inline_math_opening_dollars = parsers.dollar
11989 * #(parsers.nonspacechar)
11990
11991 local inline_math_closing_dollars
11992 = allowed_before_closing_no_space
11993 * parsers.dollar
11994 * -#(parsers.digit)
11995
11996 local inline_math_dollars = between(Cs(dollar_math_content),
11997 inline_math_opening_dollars,
11998 inline_math_closing_dollars)
11999
12000 local display_math_opening_dollars = parsers.dollar
12001 * parsers.dollar
12002
12003 local display_math_closing_dollars = parsers.dollar
12004 * parsers.dollar
12005
12006 local display_math_dollars = between(Cs(dollar_math_content),
12007 display_math_opening_dollars,
12008 display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

12009 local backslash_math_content
12010 = (parsers.newline * (parsers.check_optional_indent / ""))
12011 + parsers.linechar)
12012 - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

12013 local inline_math_opening_double = parsers.backslash
12014 * parsers.backslash
12015 * parsers.lparent
12016
12017 local inline_math_closing_double = allowed_before_closing
12018 * parsers.spacechar^0
12019 * parsers.backslash
12020 * parsers.backslash
12021 * parsers.rparent
12022
12023 local inline_math_double = between(Cs(backslash_math_content),
12024 inline_math_opening_double,
12025 inline_math_closing_double)
12026 / strip_preceding_whitespaces

```

```

12027
12028 local display_math_opening_double = parsers.backslash
12029 * parsers.backslash
12030 * parsers.lbracket
12031
12032 local display_math_closing_double = allowed_before_closing
12033 * parsers.spacechar^0
12034 * parsers.backslash
12035 * parsers.backslash
12036 * parsers.rbracket
12037
12038 local display_math_double = between(Cs(backslash_math_content),
12039 display_math_opening_double,
12040 display_math_closing_double)
12041 / strip_preceding_whitespaces

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

12042 local inline_math_opening_single = parsers.backslash
12043 * parsers.lparent
12044
12045 local inline_math_closing_single = allowed_before_closing
12046 * parsers.spacechar^0
12047 * parsers.backslash
12048 * parsers.rparent
12049
12050 local inline_math_single = between(Cs(backslash_math_content),
12051 inline_math_opening_single,
12052 inline_math_closing_single)
12053 / strip_preceding_whitespaces
12054
12055 local display_math_opening_single = parsers.backslash
12056 * parsers.lbracket
12057
12058 local display_math_closing_single = allowed_before_closing
12059 * parsers.spacechar^0
12060 * parsers.backslash
12061 * parsers.rbracket
12062
12063 local display_math_single = between(Cs(backslash_math_content),
12064 display_math_opening_single,
12065 display_math_closing_single)
12066 / strip_preceding_whitespaces
12067
12068 local display_math = parsers.fail
12069
12070 local inline_math = parsers.fail
12071
12072 if tex_math_dollars then

```

```

12073 display_math = display_math + display_math_dollars
12074 inline_math = inline_math + inline_math_dollars
12075 end
12076
12077 if tex_math_double_backslash then
12078 display_math = display_math + display_math_double
12079 inline_math = inline_math + inline_math_double
12080 end
12081
12082 if tex_math_single_backslash then
12083 display_math = display_math + display_math_single
12084 inline_math = inline_math + inline_math_single
12085 end
12086
12087 local TexMath = display_math / writer.display_math
12088 + inline_math / writer.inline_math
12089
12090 self.insert_pattern("Inline after LinkAndEmph",
12091 TexMath, "TexMath")
12092
12093 if tex_math_dollars then
12094 self.add_special_character("$")
12095 end
12096
12097 if tex_math_single_backslash or tex_math_double_backslash then
12098 self.add_special_character("\\")
12099 self.add_special_character("[")
12100 self.add_special_character("]")
12101 self.add_special_character("(")
12102 self.add_special_character("(")
12103 end
12104 end
12105 }
12106 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.

```

12107 M.extensions.jekyll_data = function(expect_jekyll_data,
12108 ensure_jekyll_data)
12109 return {
12110 name = "built-in jekyll_data syntax extension",

```

```
12111 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.

```
12112 function self.jekyllData(d, t, p)
12113 if not self.is_writing then return "" end
12114
12115 local buf = {}
12116
12117 local keys = {}
12118 for k, _ in pairs(d) do
12119 table.insert(keys, k)
12120 end
```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```
12121 table.sort(keys, function(first, second)
12122 if type(first) ~= type(second) then
12123 return type(first) < type(second)
12124 else
12125 return first < second
12126 end
12127 end)
12128
12129 if not p then
12130 table.insert(buf, "\\markdownRendererJekyllDataBegin")
12131 end
12132
12133 local is_sequence = false
12134 if #d > 0 and #d == #keys then
12135 for i=1, #d do
12136 if d[i] == nil then
12137 goto not_a_sequence
12138 end
12139 end
12140 is_sequence = true
12141 end
12142 ::not_a_sequence::
12143
12144 if is_sequence then
12145 table.insert(buf,
12146 "\\markdownRendererJekyllDataSequenceBegin{")
12147 table.insert(buf, self.identifier(p or "null"))
```

```

12148 table.insert(buf, "{")
12149 table.insert(buf, #keys)
12150 table.insert(buf, "}")
12151 else
12152 table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
12153 table.insert(buf, self.identifier(p or "null"))
12154 table.insert(buf, "{")
12155 table.insert(buf, #keys)
12156 table.insert(buf, "}")
12157 end
12158
12159 for _, k in ipairs(keys) do
12160 local v = d[k]
12161 local typ = type(v)
12162 k = tostring(k or "null")
12163 if typ == "table" and next(v) ~= nil then
12164 table.insert(
12165 buf,
12166 self.jekyllData(v, t, k)
12167)
12168 else
12169 k = self.identifier(k)
12170 v = tostring(v)
12171 if typ == "boolean" then
12172 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
12173 table.insert(buf, k)
12174 table.insert(buf, "{")
12175 table.insert(buf, v)
12176 table.insert(buf, "}")
12177 elseif typ == "number" then
12178 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
12179 table.insert(buf, k)
12180 table.insert(buf, "{")
12181 table.insert(buf, v)
12182 table.insert(buf, "}")
12183 elseif typ == "string" then
12184 table.insert(buf,
12185 "\\markdownRendererJekyllDataProgrammaticString{")
12186 table.insert(buf, k)
12187 table.insert(buf, "{")
12188 table.insert(buf, self.identifier(v))
12189 table.insert(buf, "}")
12190 table.insert(buf,
12191 "\\markdownRendererJekyllDataTypographicString{")
12192 table.insert(buf, k)
12193 table.insert(buf, "{")
12194 table.insert(buf, t(v))

```

```

12195 table.insert(buf, "}")
12196 elseif typ == "table" then
12197 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
12198 table.insert(buf, k)
12199 table.insert(buf, "}")
12200 else
12201 local error = self.error(format(
12202 "Unexpected type %s for value of "
12203 .. "YAML key %s.", typ, k))
12204 table.insert(buf, error)
12205 end
12206 end
12207 end
12208
12209 if is_sequence then
12210 table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
12211 else
12212 table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
12213 end
12214
12215 if not p then
12216 table.insert(buf, "\\markdownRendererJekyllDataEnd")
12217 end
12218
12219 return buf
12220 end
12221 end, extend_reader = function(self)
12222 local parsers = self.parsers
12223 local writer = self.writer
12224
12225 local JekyllData
12226 = Cmt(C((parsers.line - P("---") - P("..."))^0)
12227 , function(s, i, text) -- luacheck: ignore s i
12228 local data
12229 local ran_ok, _ = pcall(function()
12230 -- TODO: Use `require("tinyyaml")` in TeX Live 2023
12231 local tinyyaml = require("markdown-tinyyaml")
12232 data = tinyyaml.parse(text, {timestamps=false})
12233 end)
12234 if ran_ok and data ~= nil then
12235 return true, writer.jekyllData(data, function(s)
12236 return self.parser_functions.parse_blocks_nested(s)
12237 end, nil)
12238 else
12239 return false
12240 end
12241 end

```

```

12242)
12243
12244 local UnexpectedJekyllData
12245 = P("----")
12246 * parsers.blankline / 0
12247 -- if followed by blank, it's thematic break
12248 * #(-parsers.blankline)
12249 * JekyllData
12250 * (P("----") + P("..."))
12251
12252 local ExpectedJekyllData
12253 = (P("----")
12254 * parsers.blankline / 0
12255 -- if followed by blank, it's thematic break
12256 * #(-parsers.blankline)
12257)^-1
12258 * JekyllData
12259 * (P("----") + P("..."))^-1
12260
12261 if ensure_jekyll_data then
12262 ExpectedJekyllData = ExpectedJekyllData
12263 * parsers.eof
12264 else
12265 ExpectedJekyllData = (ExpectedJekyllData
12266 * (V("Blank")^0 / writer.interblocksep)
12267)^-1
12268 end
12269
12270 self.insert_pattern("Block before Blockquote",
12271 UnexpectedJekyllData, "UnexpectedJekyllData")
12272 if expect_jekyll_data then
12273 self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
12274 end
12275 end
12276 }
12277 end

```

### 3.1.8 Conversion from Markdown to Plain T<sub>E</sub>X

The `new` function of file `markdown.lua` loads file `markdown-parser.lua` and calls its own function `new` unless option `eagerCache` or `finalizeCache` has been enabled and a cached conversion output exists, in which case it is returned without loading file `markdown-parser.lua`.

```

12278 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

12279 options = options or {}

```



```

12280 setmetatable(options, { __index = function (_, key)
12281 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.

```

12282 local parser_convert = nil
12283 return function(input)
12284 local function convert(input)
12285 if parser_convert == nil then

```

Lazy-load `markdown-parser.lua` and check that it originates from the same version of the Markdown package.

```

12286 local parser = require("markdown-parser")
12287 if metadata.version ~= parser.metadata.version then
12288 warn("markdown.lua " .. metadata.version .. " used with " ..
12289 "markdown-parser.lua " .. parser.metadata.version .. ".")
12290 end
12291 parser_convert = parser.new(options)
12292 end
12293 return parser_convert(input)
12294 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output.

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3).

```

12295 local output
12296 if options.eagerCache or options.finalizeCache then
12297 local salt = util.salt(options)
12298 local name = util.cache(options.cacheDir, input, salt, convert,
12299 ".md.tex")
12300 output = [[\input{}} .. name .. [{}\relax]]

```

Otherwise, return the result of the conversion directly.

```

12301 else
12302 output = convert(input)
12303 end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

12304 if options.finalizeCache then
12305 local file, mode
12306 if options.frozenCacheCounter > 0 then
12307 mode = "a"
12308 else

```

```

12309 mode = "w"
12310 end
12311 file = assert(io.open(options.frozenCacheFileName, mode),
12312 [[Could not open file]] .. options.frozenCacheFileName
12313 .. [[for writing]])
12314 assert(file:write(
12315 [[\expandafter\global\expandafter\def\csname]]
12316 .. [[markdownFrozenCache]] .. options.frozenCacheCounter
12317 .. [[\endcsname{}}] .. output .. [{}]] .. "\n"))
12318 assert(file:close())
12319 end
12320 return output
12321 end
12322 end

```

The `new` function from file `markdown-parser.lua` returns a conversion function that takes a markdown string and turns it into a plain  $\text{\TeX}$  output. See Section 2.1.1.

```

12323 function M.new(options)

```

Make the `options` table inherit from the `defaultOptions` table.

```

12324 options = options or {}
12325 setmetatable(options, { __index = function (_, key)
12326 return defaultOptions[key] end })

```

If the singleton cache contains a conversion function for the same `options`, reuse it.

```

12327 if options.singletonCache and singletonCache.convert then
12328 for k, v in pairs(defaultOptions) do
12329 if type(v) == "table" then
12330 for i = 1, math.max(#singletonCache.options[k], #options[k]) do
12331 if singletonCache.options[k][i] ~= options[k][i] then
12332 goto miss
12333 end
12334 end

```

The `cacheDir` option is disregarded.

```

12335 elseif k ~= "cacheDir"
12336 and singletonCache.options[k] ~= options[k] then
12337 goto miss
12338 end
12339 end
12340 return singletonCache.convert
12341 end
12342 ::miss::

```

Apply built-in syntax extensions based on `options`.

```

12343 local extensions = {}
12344

```

```

12345 if options.bracketedSpans then
12346 local bracketed_spans_extension = M.extensions.bracketed_spans()
12347 table.insert(extensions, bracketed_spans_extension)
12348 end
12349
12350 if options.contentBlocks then
12351 local content_blocks_extension = M.extensions.content_blocks(
12352 options.contentBlocksLanguageMap)
12353 table.insert(extensions, content_blocks_extension)
12354 end
12355
12356 if options.definitionLists then
12357 local definition_lists_extension = M.extensions.definition_lists(
12358 options.tightLists)
12359 table.insert(extensions, definition_lists_extension)
12360 end
12361
12362 if options.fencedCode then
12363 local fenced_code_extension = M.extensions.fenced_code(
12364 options.blankBeforeCodeFence,
12365 options.fencedCodeAttributes,
12366 options.rawAttribute)
12367 table.insert(extensions, fenced_code_extension)
12368 end
12369
12370 if options.fencedDivs then
12371 local fenced_div_extension = M.extensions.fenced_divs(
12372 options.blankBeforeDivFence)
12373 table.insert(extensions, fenced_div_extension)
12374 end
12375
12376 if options.headerAttributes then
12377 local header_attributes_extension = M.extensions.header_attributes()
12378 table.insert(extensions, header_attributes_extension)
12379 end
12380
12381 if options.inlineCodeAttributes then
12382 local inline_code_attributes_extension =
12383 M.extensions.inline_code_attributes()
12384 table.insert(extensions, inline_code_attributes_extension)
12385 end
12386
12387 if options.jekyllData then
12388 local jekyll_data_extension = M.extensions.jekyll_data(
12389 options.expectJekyllData, options.ensureJekyllData)
12390 table.insert(extensions, jekyll_data_extension)
12391 end

```

```

12392
12393 if options.linkAttributes then
12394 local link_attributes_extension =
12395 M.extensions.link_attributes()
12396 table.insert(extensions, link_attributes_extension)
12397 end
12398
12399 if options.lineBlocks then
12400 local line_block_extension = M.extensions.line_blocks()
12401 table.insert(extensions, line_block_extension)
12402 end
12403
12404 if options.mark then
12405 local mark_extension = M.extensions.mark()
12406 table.insert(extensions, mark_extension)
12407 end
12408
12409 if options.pipeTables then
12410 local pipe_tables_extension = M.extensions.pipe_tables(
12411 options.tableCaptions, options.tableAttributes)
12412 table.insert(extensions, pipe_tables_extension)
12413 end
12414
12415 if options.rawAttribute then
12416 local raw_inline_extension = M.extensions.raw_inline()
12417 table.insert(extensions, raw_inline_extension)
12418 end
12419
12420 if options.strikeThrough then
12421 local strike_through_extension = M.extensions.strike_through()
12422 table.insert(extensions, strike_through_extension)
12423 end
12424
12425 if options.subscripts then
12426 local subscript_extension = M.extensions.subscripts()
12427 table.insert(extensions, subscript_extension)
12428 end
12429
12430 if options.superscripts then
12431 local superscript_extension = M.extensions.superscripts()
12432 table.insert(extensions, superscript_extension)
12433 end
12434
12435 if options.texMathDollars or
12436 options.texMathSingleBackslash or
12437 options.texMathDoubleBackslash then
12438 local tex_math_extension = M.extensions.tex_math(

```

```

12439 options.texMathDollars,
12440 options.texMathSingleBackslash,
12441 options.texMathDoubleBackslash)
12442 table.insert(extensions, tex_math_extension)
12443 end
12444
12445 if options.notes or options.inlineNotes then
12446 local notes_extension = M.extensions.notes(
12447 options.notes, options.inlineNotes)
12448 table.insert(extensions, notes_extension)
12449 end
12450
12451 if options.citations then
12452 local citations_extension
12453 = M.extensions.citations(options.citationNbsps)
12454 table.insert(extensions, citations_extension)
12455 end
12456
12457 if options.fancyLists then
12458 local fancy_lists_extension = M.extensions.fancy_lists()
12459 table.insert(extensions, fancy_lists_extension)
12460 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

12461 for _, user_extension_filename in ipairs(options.extensions) do
12462 local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

12463 local pathname = assert(kpse.find_file(filename),
12464 [[Could not locate user-defined syntax extension "]]
12465 .. filename)
12466 local input_file = assert(io.open(pathname, "r"),
12467 [[Could not open user-defined syntax extension "]]
12468 .. pathname .. [[" for reading]])
12469 local input = assert(input_file:read("*a"))
12470 assert(input_file:close())
12471 local user_extension, err = load([[
12472 local sandbox = {}
12473 setmetatable(sandbox, {__index = _G})
12474 _ENV = sandbox
12475]] .. input)()
12476 assert(user_extension,
12477 [[Failed to compile user-defined syntax extension "]]
12478 .. pathname .. [[:]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

12479 assert(user_extension.api_version ~= nil,
12480 [[User-defined syntax extension "]] .. pathname

```

```

12481 .. [" does not specify mandatory field "api_version"'])
12482 assert(type(user_extension.api_version) == "number",
12483 [[User-defined syntax extension "]] .. pathname
12484 .. [" specifies field "api_version" of type "]]
12485 .. type(user_extension.api_version)
12486 .. [" but "number" was expected'])
12487 assert(user_extension.api_version > 0
12488 and user_extension.api_version
12489 <= metadata.user_extension_api_version,
12490 [[User-defined syntax extension "]] .. pathname
12491 .. [" uses syntax extension API version "]]
12492 .. user_extension.api_version .. [[but markdown.lua]]
12493 .. metadata.version .. [[uses API version]]
12494 .. metadata.user_extension_api_version
12495 .. [[, which is incompatible'])
12496
12497 assert(user_extension.grammar_version ~= nil,
12498 [[User-defined syntax extension "]] .. pathname
12499 .. [" does not specify mandatory field "grammar_version"'])
12500 assert(type(user_extension.grammar_version) == "number",
12501 [[User-defined syntax extension "]] .. pathname
12502 .. [" specifies field "grammar_version" of type "]]
12503 .. type(user_extension.grammar_version)
12504 .. [" but "number" was expected'])
12505 assert(user_extension.grammar_version == metadata.grammar_version,
12506 [[User-defined syntax extension "]] .. pathname
12507 .. [" uses grammar version "]]
12508 .. user_extension.grammar_version
12509 .. [[but markdown.lua]] .. metadata.version
12510 .. [[uses grammar version]] .. metadata.grammar_version
12511 .. [[, which is incompatible'])
12512
12513 assert(user_extension.finalize_grammar ~= nil,
12514 [[User-defined syntax extension "]] .. pathname
12515 .. [" does not specify mandatory "finalize_grammar" field'])
12516 assert(type(user_extension.finalize_grammar) == "function",
12517 [[User-defined syntax extension "]] .. pathname
12518 .. [" specifies field "finalize_grammar" of type "]]
12519 .. type(user_extension.finalize_grammar)
12520 .. [" 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.)

```

12521 local extension = {
12522 name = [[user-defined "]] .. pathname .. [[" syntax extension]],
12523 extend_reader = user_extension.finalize_grammar,
12524 extend_writer = function() end,

```

```

12525 }
12526 return extension
12527 end)(user_extension_filename)
12528 table.insert(extensions, user_extension)
12529 end

```

Produce a conversion function from markdown to plain T<sub>E</sub>X.

```

12530 local writer = M.writer.new(options)
12531 local reader = M.reader.new(writer, options)
12532 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`.

```

12533 collectgarbage("collect")

```

Update the singleton cache.

```

12534 if options.singletonCache then
12535 local singletonCacheOptions = {}
12536 for k, v in pairs(options) do
12537 singletonCacheOptions[k] = v
12538 end
12539 setmetatable(singletonCacheOptions,
12540 { __index = function (_, key)
12541 return defaultOptions[key] end })
12542 singletonCache.options = singletonCacheOptions
12543 singletonCache.convert = convert
12544 end

```

Return the conversion function from markdown to plain T<sub>E</sub>X.

```

12545 return convert
12546 end
12547 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.

```

12548
12549 local input
12550 if input_filename then
12551 local input_file = assert(io.open(input_filename, "r"),
12552 [[Could not open file]] .. input_filename .. [[for reading]])
12553 input = assert(input_file:read("*a"))
12554 assert(input_file:close())
12555 else
12556 input = assert(io.read("*a"))
12557 end
12558

```

First, ensure that the `options.cacheDir` directory exists.

```
12559 local lfs = require("lfs")
12560 if options.cacheDir and not lfs.isdir(options.cacheDir) then
12561 assert(lfs.mkdir(options["cacheDir"]))
12562 end
```

If Kpathsea has not been loaded before or if LuaTeX has not yet been initialized, configure Kpathsea on top of loading it.

```
12563 local kpse
12564 (function()
12565 local should_initialize = package.loaded.kpse == nil
12566 or tex.initialize ~= nil
12567 kpse = require("kpse")
12568 if should_initialize then
12569 kpse.set_program_name("luatex")
12570 end
12571 end)()
12572 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.

```
12573 if metadata.version ~= md.metadata.version then
12574 warn("markdown-cli.lua " .. metadata.version .. " used with " ..
12575 "markdown.lua " .. md.metadata.version .. ".")
12576 end
12577 local convert = md.new(options)
12578 local output = convert(input)
12579
12580 if output_filename then
12581 local output_file = assert(io.open(output_filename, "w"),
12582 [[Could not open file]] .. output_filename .. [[for writing]])
12583 assert(output_file:write(output))
12584 assert(output_file:close())
12585 else
12586 assert(io.write(output))
12587 end
```

Remove the `options.cacheDir` directory if it is empty.

```
12588 if options.cacheDir then
12589 lfs.rmdir(options.cacheDir)
12590 end
```

## 3.2 Plain TeX Implementation

The plain TeX implementation provides macros for the interfacing between TeX 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 TeX exposed by the plain TeX interface (see Section 2.2).



### 3.2.1 Logging Facilities

```
12591 \ExplSyntaxOn
12592 \cs_if_free:NT
12593 \markdownInfo
12594 {
12595 \cs_new:Npn
12596 \markdownInfo #1
12597 {
12598 \msg_info:nne
12599 { markdown }
12600 { generic-message }
12601 { #1 }
12602 }
12603 }
12604 \cs_if_free:NT
12605 \markdownWarning
12606 {
12607 \cs_new:Npn
12608 \markdownWarning #1
12609 {
12610 \msg_warning:nne
12611 { markdown }
12612 { generic-message }
12613 { #1 }
12614 }
12615 }
12616 \cs_if_free:NT
12617 \markdownError
12618 {
12619 \cs_new:Npn
12620 \markdownError #1 #2
12621 {
12622 \msg_error:nnee
12623 { markdown }
12624 { generic-message-with-help-text }
12625 { #1 }
12626 { #2 }
12627 }
12628 }
12629 \msg_new:nnn
12630 { markdown }
12631 { generic-message }
12632 { #1 }
12633 \msg_new:nnnn
12634 { markdown }
12635 { generic-message-with-help-text }
```

```

12636 { #1 }
12637 { #2 }
12638 \cs_generate_variant:Nn
12639 \msg_info:nnn
12640 { nne }
12641 \cs_generate_variant:Nn
12642 \msg_warning:nnn
12643 { nne }
12644 \cs_generate_variant:Nn
12645 \msg_error:nnnn
12646 { nnee }
12647 \ExplSyntaxOff

```

### 3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes provided with the Markdown package. Furthermore, this section also implements the built-in plain T<sub>E</sub>X themes provided with the Markdown package.

```

12648 \ExplSyntaxOn
12649 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
12650 \prop_new:N \g_@@_plain_tex_loaded_themes_versions_prop
12651 \cs_new:Nn
12652 \@@_plain_tex_load_theme:nnn
12653 {
12654 \prop_get:NnNTF
12655 \g_@@_plain_tex_loaded_themes_linenos_prop
12656 { #1 }
12657 \l_tmpa_tl
12658 {
12659 \prop_get:NnN
12660 \g_@@_plain_tex_loaded_themes_versions_prop
12661 { #1 }
12662 \l_tmpb_tl
12663 \str_if_eq:nVTF
12664 { #2 }
12665 \l_tmpb_tl
12666 {
12667 \msg_warning:nnnVn
12668 { markdown }
12669 { repeatedly-loaded-plain-tex-theme }
12670 { #1 }
12671 \l_tmpa_tl
12672 { #2 }
12673 }
12674 {
12675 \msg_error:nnnnVV
12676 { markdown }

```

```

12677 { different-versions-of-plain-tex-theme }
12678 { #1 }
12679 { #2 }
12680 \l_tmpb_tl
12681 \l_tmpa_tl
12682 }
12683 }
12684 {
12685 \msg_info:nnnn
12686 { markdown }
12687 { loading-plain-tex-theme }
12688 { #1 }
12689 { #2 }
12690 \prop_gput:Nnx
12691 \g_@@_plain_tex_loaded_themes_linenos_prop
12692 { #1 }
12693 { \tex_the:D \tex_inputlineno:D }
12694 \prop_gput:Nnn
12695 \g_@@_plain_tex_loaded_themes_versions_prop
12696 { #1 }
12697 { #2 }
12698 \file_input:n
12699 { markdown theme #3 }
12700 }
12701 }
12702 \msg_new:nnn
12703 { markdown }
12704 { loading-plain-tex-theme }
12705 { Loading~version~#2~of~plain~TeX~Markdown~theme~#1 }
12706 \msg_new:nnn
12707 { markdown }
12708 { repeatedly-loaded-plain-tex-theme }
12709 {
12710 Version~#3~of~plain~TeX~Markdown~theme~#1~was~previously~
12711 loaded~on~line~#2,~not~loading~it~again
12712 }
12713 \msg_new:nnn
12714 { markdown }
12715 { different-versions-of-plain-tex-theme }
12716 {
12717 Tried~to~load~version~#2~of~plain~TeX~Markdown~theme~#1~
12718 but~version~#3~has~already~been~loaded~on~line~#4
12719 }
12720 \cs_generate_variant:Nn
12721 \prop_gput:Nnn
12722 { Nnx }
12723 \cs_gset_eq:NN

```

```

12724 \@@_load_theme:nnn
12725 \@@_plain_tex_load_theme:nnn
12726 \cs_generate_variant:Nn
12727 \@@_load_theme:nnn
12728 { VeV }
12729 \cs_generate_variant:Nn
12730 \msg_error:nnnnnn
12731 { nnnnVV }
12732 \cs_generate_variant:Nn
12733 \msg_warning:nnnnn
12734 { 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<sup>A</sup>T<sub>E</sub>X and ConT<sub>E</sub>Xt.

```

12735 \cs_new:Npn
12736 \markdownLoadPlainTeXTheme
12737 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

12738 \tl_set:NV
12739 \l_tmpa_tl
12740 \g_@@_current_theme_tl
12741 \tl_reverse:N
12742 \l_tmpa_tl
12743 \tl_set:Ne
12744 \l_tmpb_tl
12745 {
12746 \tl_tail:V
12747 \l_tmpa_tl
12748 }
12749 \tl_reverse:N
12750 \l_tmpb_tl

```

Next, we munge the theme name.

```

12751 \str_set:NV
12752 \l_tmpa_str
12753 \l_tmpb_tl
12754 \str_replace_all:Nnn
12755 \l_tmpa_str
12756 { / }
12757 { _ }

```

Finally, we load the plain TeX theme.

```

12758 \@@_plain_tex_load_theme:VeV
12759 \l_tmpb_tl
12760 { \markdownThemeVersion }

```

```

12761 \l_tmpa_str
12762 }
12763 \cs_generate_variant:Nn
12764 \tl_set:Nn
12765 { Ne }
12766 \cs_generate_variant:Nn
12767 \@@_plain_tex_load_theme:nnn
12768 { VeV }
12769 \ExplSyntaxOff

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

12770 \markdownSetup {
12771 rendererPrototypes = {
12772 tilde = {\~},
12773 },
12774 }

```

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.

```

12775 \def\markdownRendererInterblockSeparatorPrototype{\par}%
12776 \def\markdownRendererParagraphSeparatorPrototype{%
12777 \markdownRendererInterblockSeparator}%
12778 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
12779 \def\markdownRendererSoftLineBreakPrototype{ }%
12780 \let\markdownRendererEllipsisPrototype\dots
12781 \def\markdownRendererNbspPrototype{\~}%
12782 \def\markdownRendererLeftBracePrototype{\char`\{}%
12783 \def\markdownRendererRightBracePrototype{\char`\}%
12784 \def\markdownRendererDollarSignPrototype{\char`\$}%
12785 \def\markdownRendererPercentSignPrototype{\char`\}%
12786 \def\markdownRendererAmpersandPrototype{\&%
12787 \def\markdownRendererUnderscorePrototype{\char`_%
12788 \def\markdownRendererHashPrototype{\char`\#}%
12789 \def\markdownRendererCircumflexPrototype{\char`\^}%
12790 \def\markdownRendererBackslashPrototype{\char`\}%
12791 \def\markdownRendererTildePrototype{\char`\~}%
12792 \def\markdownRendererPipePrototype{|}%
12793 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
12794 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
12795 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
12796 \markdownInput{#3}}%
12797 \def\markdownRendererContentBlockOnlineImagePrototype{%
12798 \markdownRendererImage}%

```

```

12799 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
12800 \markdownRendererInputFencedCode{#3}{#2}{#2}}%
12801 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
12802 \def\markdownRendererUlBeginPrototype{}%
12803 \def\markdownRendererUlBeginTightPrototype{}%
12804 \def\markdownRendererUlItemPrototype{}%
12805 \def\markdownRendererUlItemEndPrototype{}%
12806 \def\markdownRendererUlEndPrototype{}%
12807 \def\markdownRendererUlEndTightPrototype{}%
12808 \def\markdownRendererOlBeginPrototype{}%
12809 \def\markdownRendererOlBeginTightPrototype{}%
12810 \def\markdownRendererFancyOlBeginPrototype#1#2{%
12811 \markdownRendererOlBegin}%
12812 \def\markdownRendererFancyOlBeginTightPrototype#1#2{%
12813 \markdownRendererOlBeginTight}%
12814 \def\markdownRendererOlItemPrototype{}%
12815 \def\markdownRendererOlItemWithNumberPrototype#1{}%
12816 \def\markdownRendererOlItemEndPrototype{}%
12817 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
12818 \def\markdownRendererFancyOlItemWithNumberPrototype{%
12819 \markdownRendererOlItemWithNumber}%
12820 \def\markdownRendererFancyOlItemEndPrototype{}%
12821 \def\markdownRendererOlEndPrototype{}%
12822 \def\markdownRendererOlEndTightPrototype{}%
12823 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
12824 \def\markdownRendererFancyOlEndTightPrototype{%
12825 \markdownRendererOlEndTight}%
12826 \def\markdownRendererDlBeginPrototype{}%
12827 \def\markdownRendererDlBeginTightPrototype{}%
12828 \def\markdownRendererDlItemPrototype#1{#1}%
12829 \def\markdownRendererDlItemEndPrototype{}%
12830 \def\markdownRendererDlDefinitionBeginPrototype{}%
12831 \def\markdownRendererDlDefinitionEndPrototype{\par}%
12832 \def\markdownRendererDlEndPrototype{}%
12833 \def\markdownRendererDlEndTightPrototype{}%
12834 \def\markdownRendererEmphasisPrototype#1{{\it#1}}%
12835 \def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
12836 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
12837 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
12838 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
12839 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
12840 \def\markdownRendererInputVerbatimPrototype#1{%
12841 \par{\tt\input#1\relax{}}\par}%
12842 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
12843 \markdownRendererInputVerbatim{#1}}%
12844 \def\markdownRendererHeadingOnePrototype#1{#1}%
12845 \def\markdownRendererHeadingTwoPrototype#1{#1}%

```

```

12846 \def\markdownRendererHeadingThreePrototype#1{#1}%
12847 \def\markdownRendererHeadingFourPrototype#1{#1}%
12848 \def\markdownRendererHeadingFivePrototype#1{#1}%
12849 \def\markdownRendererHeadingSixPrototype#1{#1}%
12850 \def\markdownRendererThematicBreakPrototype{}%
12851 \def\markdownRendererNotePrototype#1{#1}%
12852 \def\markdownRendererCitePrototype#1{}%
12853 \def\markdownRendererTextCitePrototype#1{}%
12854 \def\markdownRendererTickedBoxPrototype{[X]}%
12855 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
12856 \def\markdownRendererUntickedBoxPrototype{[]}%
12857 \def\markdownRendererStrikeThroughPrototype#1{#1}%
12858 \def\markdownRendererSuperscriptPrototype#1{#1}%
12859 \def\markdownRendererSubscriptPrototype#1{#1}%
12860 \def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
12861 \def\markdownRendererInlineMathPrototype#1{$#1$}%
12862 \ExplSyntaxOn
12863 \cs_gset:Npn
12864 \markdownRendererHeaderAttributeContextBeginPrototype
12865 {
12866 \group_begin:
12867 \color_group_begin:
12868 }
12869 \cs_gset:Npn
12870 \markdownRendererHeaderAttributeContextEndPrototype
12871 {
12872 \color_group_end:
12873 \group_end:
12874 }
12875 \cs_gset_eq:NN
12876 \markdownRendererBracketedSpanAttributeContextBeginPrototype
12877 \markdownRendererHeaderAttributeContextBeginPrototype
12878 \cs_gset_eq:NN
12879 \markdownRendererBracketedSpanAttributeContextEndPrototype
12880 \markdownRendererHeaderAttributeContextEndPrototype
12881 \cs_gset_eq:NN
12882 \markdownRendererFencedDivAttributeContextBeginPrototype
12883 \markdownRendererHeaderAttributeContextBeginPrototype
12884 \cs_gset_eq:NN
12885 \markdownRendererFencedDivAttributeContextEndPrototype
12886 \markdownRendererHeaderAttributeContextEndPrototype
12887 \cs_gset_eq:NN
12888 \markdownRendererFencedCodeAttributeContextBeginPrototype
12889 \markdownRendererHeaderAttributeContextBeginPrototype
12890 \cs_gset_eq:NN
12891 \markdownRendererFencedCodeAttributeContextEndPrototype
12892 \markdownRendererHeaderAttributeContextEndPrototype

```

```

12893 \cs_gset:Npn
12894 \markdownRendererReplacementCharacterPrototype
12895 { \codepoint_str_generate:n { fffd } }
12896 \ExplSyntaxOff
12897 \def\markdownRendererSectionBeginPrototype{%
12898 \def\markdownRendererSectionEndPrototype{%
12899 \ExplSyntaxOn
12900 \cs_gset:Npn
12901 \markdownRendererWarningPrototype
12902 #1#2#3#4
12903 {
12904 \tl_set:Nn
12905 \l_tmpa_tl
12906 { #2 }
12907 \tl_if_empty:nF
12908 { #4 }
12909 {
12910 \tl_put_right:Nn
12911 \l_tmpa_tl
12912 { \iow_newline: #4 }
12913 }
12914 \exp_args:NV
12915 \markdownWarning
12916 \l_tmpa_tl
12917 }
12918 \ExplSyntaxOff
12919 \def\markdownRendererErrorPrototype#1#2#3#4{%
12920 \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.

```

12921 \ExplSyntaxOn
12922 \cs_new:Nn
12923 \@@_plain_tex_default_input_raw_inline:nn
12924 {
12925 \str_case:nn
12926 { #2 }
12927 {
12928 { md } { \markdownInput{#1} }
12929 { tex } { \markdownEscape{#1} \unskip }
12930 }
12931 }
12932 \cs_new:Nn
12933 \@@_plain_tex_default_input_raw_block:nn

```



```

12934 {
12935 \str_case:nn
12936 { #2 }
12937 {
12938 { md } { \markdownInput{#1} }
12939 { tex } { \markdownEscape{#1} }
12940 }
12941 }
12942 \cs_gset:Npn
12943 \markdownRendererInputRawInlinePrototype#1#2
12944 {
12945 \@@_plain_tex_default_input_raw_inline:nn
12946 { #1 }
12947 { #2 }
12948 }
12949 \cs_gset:Npn
12950 \markdownRendererInputRawBlockPrototype#1#2
12951 {
12952 \@@_plain_tex_default_input_raw_block:nn
12953 { #1 }
12954 { #2 }
12955 }
12956 \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  $p$ .

```

12957 \ExplSyntaxOn
12958 \seq_new:N \g_@@_jekyll_data_datatypes_seq
12959 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
12960 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
12961 \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.

```

12962 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
12963 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
12964 {
12965 \seq_if_empty:NF
12966 \g_@@_jekyll_data_datatypes_seq
12967 {
12968 \seq_get_right:NN
12969 \g_@@_jekyll_data_datatypes_seq
12970 \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.

```

12971 \str_if_eq:NNTF
12972 \l_tmpa_tl
12973 \c_@@_jekyll_data_sequence_tl
12974 {
12975 \seq_put_right:Nn
12976 \g_@@_jekyll_data_wildcard_absolute_address_seq
12977 { * }
12978 }
12979 {
12980 \seq_put_right:Nn
12981 \g_@@_jekyll_data_wildcard_absolute_address_seq
12982 { #1 }
12983 }
12984 }
12985 }

```

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.

```
12986 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
12987 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
12988 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
12989 {
12990 \seq_pop_left:NN #1 \l_tmpa_tl
12991 \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
12992 \seq_put_left:NV #1 \l_tmpa_tl
12993 }
12994 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
12995 {
12996 \markdown_jekyll_data_concatenate_address:NN
12997 \g_@@_jekyll_data_wildcard_absolute_address_seq
12998 \g_@@_jekyll_data_wildcard_absolute_address_tl
12999 \seq_get_right:NN
13000 \g_@@_jekyll_data_wildcard_absolute_address_seq
13001 \g_@@_jekyll_data_wildcard_relative_address_tl
13002 }
```

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.

```
13003 \cs_new:Nn \markdown_jekyll_data_push:nN
13004 {
13005 \markdown_jekyll_data_push_address_segment:n
13006 { #1 }
13007 \seq_put_right:NV
13008 \g_@@_jekyll_data_datatypes_seq
13009 #2
13010 \markdown_jekyll_data_update_address_tls:
13011 }
13012 \cs_new:Nn \markdown_jekyll_data_pop:
13013 {
13014 \seq_pop_right:NN
13015 \g_@@_jekyll_data_wildcard_absolute_address_seq
13016 \l_tmpa_tl
```

```

13017 \seq_pop_right:NN
13018 \g_@@_jekyll_data_datatypes_seq
13019 \l_tmpa_tl
13020 \markdown_jekyll_data_update_address_tls:
13021 }

```

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.

```

13022 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
13023 {
13024 \keys_set_known:nn
13025 { markdown/jekyllData }
13026 { { #1 } = { #2 } }
13027 }
13028 \cs_generate_variant:Nn
13029 \markdown_jekyll_data_set_keyval:nn
13030 { Vn }
13031 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
13032 {
13033 \markdown_jekyll_data_push:nN
13034 { #1 }
13035 \c_@@_jekyll_data_scalar_tl
13036 \markdown_jekyll_data_set_keyval:Vn
13037 \g_@@_jekyll_data_wildcard_absolute_address_tl
13038 { #2 }
13039 \markdown_jekyll_data_set_keyval:Vn
13040 \g_@@_jekyll_data_wildcard_relative_address_tl
13041 { #2 }
13042 \markdown_jekyll_data_pop:
13043 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

13044 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
13045 \markdown_jekyll_data_push:nN
13046 { #1 }
13047 \c_@@_jekyll_data_sequence_tl
13048 }
13049 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
13050 \markdown_jekyll_data_push:nN
13051 { #1 }
13052 \c_@@_jekyll_data_mapping_tl
13053 }
13054 \def\markdownRendererJekyllDataSequenceEndPrototype{
13055 \markdown_jekyll_data_pop:
13056 }
13057 \def\markdownRendererJekyllDataMappingEndPrototype{

```

```

13058 \markdown_jekyll_data_pop:
13059 }
13060 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
13061 \markdown_jekyll_data_set_keyvals:nn
13062 { #1 }
13063 { #2 }
13064 }
13065 \def\markdownRendererJekyllDataEmptyPrototype#1{}
13066 \def\markdownRendererJekyllDataNumberPrototype#1#2{
13067 \markdown_jekyll_data_set_keyvals:nn
13068 { #1 }
13069 { #2 }
13070 }

```

We will process all string scalar values assuming that they may contain markdown markup and are intended for typesetting.

```

13071 \def\markdownRendererJekyllDataProgrammaticStringPrototype#1#2{}
13072 \def\markdownRendererJekyllDataTypographicStringPrototype#1#2{
13073 \markdown_jekyll_data_set_keyvals:nn
13074 { #1 }
13075 { #2 }
13076 }
13077 \ExplSyntaxOff

```

If plain  $\text{\TeX}$  is the top layer, we load the [witiko/markdown/defaults](#) plain  $\text{\TeX}$  theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

13078 \ExplSyntaxOn
13079 \str_if_eq:VVT
13080 \c_@@_top_layer_tl
13081 \c_@@_option_layer_plain_tex_tl
13082 {
13083 \ExplSyntaxOff
13084 \@@_if_option:nF
13085 { noDefaults }
13086 {
13087 \@@_if_option:nTF
13088 { experimental }
13089 {
13090 \@@_setup:n
13091 { theme = witiko/markdown/defaults@experimental }
13092 }
13093 {
13094 \@@_setup:n
13095 { theme = witiko/markdown/defaults }
13096 }
13097 }
13098 \ExplSyntaxOn

```

```

13099 }
13100 \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).

```

13101 \ExplSyntaxOn
13102 \tl_new:N \g_@@_formatted_lua_options_tl
13103 \cs_new:Nn \@@_format_lua_options:
13104 {
13105 \tl_gclear:N
13106 \g_@@_formatted_lua_options_tl
13107 \seq_map_function:NN
13108 \g_@@_lua_options_seq
13109 \@@_format_lua_option:n
13110 }
13111 \cs_new:Nn \@@_format_lua_option:n
13112 {
13113 \@@_typecheck_option:n
13114 { #1 }
13115 \@@_get_option_type:nN
13116 { #1 }
13117 \l_tmpa_tl
13118 \bool_case_true:nF
13119 {
13120 {
13121 \str_if_eq_p:VV
13122 \l_tmpa_tl
13123 \c_@@_option_type_boolean_tl ||
13124 \str_if_eq_p:VV
13125 \l_tmpa_tl
13126 \c_@@_option_type_number_tl ||
13127 \str_if_eq_p:VV
13128 \l_tmpa_tl
13129 \c_@@_option_type_counter_tl
13130 }
13131 {
13132 \@@_get_option_value:nN
13133 { #1 }
13134 \l_tmpa_tl
13135 \tl_gput_right:Nx
13136 \g_@@_formatted_lua_options_tl
13137 { #1~== \l_tmpa_tl ,~ }
13138 }

```

```

13139 {
13140 \str_if_eq_p:VV
13141 \l_tmpa_tl
13142 \c_@@_option_type_clist_tl
13143 }
13144 {
13145 \@@_get_option_value:nN
13146 { #1 }
13147 \l_tmpa_tl
13148 \tl_gput_right:Nx
13149 \g_@@_formatted_lua_options_tl
13150 { #1~::~\c_left_brace_str }
13151 \clist_map_inline:Vn
13152 \l_tmpa_tl
13153 {
13154 \@@_lua_escape:xN
13155 { ##1 }
13156 \l_tmpb_tl
13157 \tl_gput_right:Nn
13158 \g_@@_formatted_lua_options_tl
13159 { " }
13160 \tl_gput_right:NV
13161 \g_@@_formatted_lua_options_tl
13162 \l_tmpb_tl
13163 \tl_gput_right:Nn
13164 \g_@@_formatted_lua_options_tl
13165 { " ,~ }
13166 }
13167 \tl_gput_right:Nx
13168 \g_@@_formatted_lua_options_tl
13169 { \c_right_brace_str ,~ }
13170 }
13171 }
13172 {
13173 \@@_get_option_value:nN
13174 { #1 }
13175 \l_tmpa_tl
13176 \@@_lua_escape:xN
13177 { \l_tmpa_tl }
13178 \l_tmpb_tl
13179 \tl_gput_right:Nn
13180 \g_@@_formatted_lua_options_tl
13181 { #1~::~ " }
13182 \tl_gput_right:NV
13183 \g_@@_formatted_lua_options_tl
13184 \l_tmpb_tl
13185 \tl_gput_right:Nn

```

```

13186 \g_@@_formatted_lua_options_tl
13187 { " ,~ }
13188 }
13189 }
13190 \cs_generate_variant:Nn
13191 \clist_map_inline:nn
13192 { Vn }
13193 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
13194 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
13195 \sys_if_engine luatex:TF
13196 {
13197 \cs_new:Nn
13198 \@@_lua_escape:nN
13199 {
13200 \tl_set:Nx
13201 #2
13202 {
13203 \lua_escape:n
13204 { #1 }
13205 }
13206 }
13207 }
13208 {
13209 \regex_const:Nn
13210 \c_@@_lua_escape_regex
13211 { [\\"] }
13212 \cs_new:Nn
13213 \@@_lua_escape:nN
13214 {
13215 \tl_set:Nn
13216 #2
13217 { #1 }
13218 \regex_replace_all:NnN
13219 \c_@@_lua_escape_regex
13220 { \u { c_backslash_str } \0 }
13221 #2
13222 }
13223 }
13224 \cs_generate_variant:Nn
13225 \@@_lua_escape:nN
13226 { 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.

```

13227 \tl_new:N
13228 \markdownInputFilename

```



```

13229 \cs_new:Npn
13230 \markdownPrepareInputFilename
13231 #1
13232 {
13233 \@@_lua_escape:xN
13234 { #1 }
13235 \markdownInputFilename
13236 \tl_gset:Nx
13237 \markdownInputFilename
13238 { " \markdownInputFilename " }
13239 }

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T<sub>E</sub>X. It exposes the `convert` function for the use by any further Lua code.

```

13240 \cs_new:Npn
13241 \markdownPrepare
13242 {

```

First, ensure that the `cacheDir` directory exists.

```

13243 local~lfs = require("lfs")
13244 local~options = \markdownLuaOptions
13245 if~not~lfs.isdir(options.cacheDir) then~
13246 assert(lfs.mkdir(options.cacheDir))
13247 end~

```

Next, load the `markdown` module and create a converter function using the plain T<sub>E</sub>X options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

13248 local~md = require("markdown")
13249 local~convert = md.new(options)
13250 }

```

The `\markdownConvert` macro contains the Lua code that is executed during the conversion from markdown to plain T<sub>E</sub>X. It opens the input file, converts it, and prints the conversion result.

```

13251 \cs_new:Npn
13252 \markdownConvert
13253 {
13254 local~filename = \markdownInputFilename
13255 local~file = assert(io.open(filename, "r"),
13256 [[Could~not~open~file~]] .. filename .. [[~for~reading]])
13257 local~input = assert(file:read("*a"))
13258 assert(file:close())
13259 print(convert(input))
13260 }
13261 \ExplSyntaxOff

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain T<sub>E</sub>X.

```

13262 \def\markdownCleanup{%
Remove the options.cacheDir directory if it is empty.
13263 if options.cacheDir then
13264 lfs.rmdir(options.cacheDir)
13265 end
13266 }%

```

### 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.

```

13267 \csname newread\endcsname\markdownInputFileStream
13268 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

13269 \begingroup
13270 \catcode\^^I=12%
13271 \gdef\markdownReadAndConvertTab{^^I}%
13272 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the  $\text{\LaTeX 2}_{\epsilon}$  `\filecontents` macro to plain  $\text{\TeX}$ .

```

13273 \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.

```

13274 \catcode\^^M=13%
13275 \catcode\^^I=13%
13276 \catcode|=0%
13277 \catcode\=12%
13278 |catcode@=14%
13279 |catcode|=12@
13280 |gdef|markdownReadAndConvert#1#2{@
13281 |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

13282 |markdownIfOption{frozenCache}{-}{@
13283 |immediate|openout|markdownOutputFileStream@
13284 |markdownOptionInputTempFileName|relax@
13285 |markdownInfo{@
13286 Buffering block-level markdown input into the temporary @
13287 input file "|markdownOptionInputTempFileName" and scanning @

```

```

13288 for the closing token sequence "#1"}@
13289 }@

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

13290 |def|do##1{|catcode`##1=12}|dospecials@
13291 |catcode`\ =12@
13292 |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.

```

13293 |def|markdownReadAndConvertStripPercentSign##1{@
13294 |markdownIfOption{stripPercentSigns}{@
13295 |if##1%@
13296 |expandafter|expandafter|expandafter@
13297 |markdownReadAndConvertProcessLine@
13298 |else@
13299 |expandafter|expandafter|expandafter@
13300 |markdownReadAndConvertProcessLine@
13301 |expandafter|expandafter|expandafter##1@
13302 |fi@
13303 }{@
13304 |expandafter@
13305 |markdownReadAndConvertProcessLine@
13306 |expandafter##1@
13307 }@
13308 }@

```

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.

```

13309 |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.

```

13310 |ifx|relax##3|relax@
13311 |markdownIfOption{frozenCache}{@
13312 |immediate|write|markdownOutputFileStream{##1}@
13313 }@
13314 |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.

```

13315 |def^^M{@
13316 |markdownInfo{The ending token sequence was found}@
13317 |markdownIfOption{frozenCache}{-}{@
13318 |immediate|closeout|markdownOutputFileStream@
13319 }@
13320 |endgroup@
13321 |markdownInput{@
13322 |markdownOptionOutputDir@
13323 /|markdownOptionInputTempFileName@
13324 }@
13325 #2}@
13326 |fi@

```

Repeat with the next line.

```

13327 ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

13328 |catcode`\^^I=13@
13329 |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.

```

13330 |catcode`\^^M=13@
13331 |def^^M##1^^M{@
13332 |def^^M###1^^M{@
13333 |markdownReadAndConvertStripPercentSign###1#1#1|relax}@
13334 ^^M}@
13335 ^^M}@

```

Reset the character categories back to the former state.

```

13336 |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.

```

13337 \ExplSyntaxOn
13338 \cs_new:Npn
13339 \markdownLuaExecute
13340 #1
13341 {
13342 \int_compare:nNt
13343 { \g_luabridge_method_int }
13344 =
13345 { \c_luabridge_method_shell_int }
13346 {

```

```

13347 \sys_if_shell_unrestricted:F
13348 {
13349 \sys_if_shell:TF
13350 {
13351 \msg_error:nn
13352 { markdown }
13353 { restricted-shell-access }
13354 }
13355 {
13356 \msg_error:nn
13357 { markdown }
13358 { disabled-shell-access }
13359 }
13360 }
13361 }
13362 \str_gset:NV
13363 \g_luabridge_output_dirname_str
13364 \markdownOptionOutputDir
13365 \luabridge_now:e
13366 { #1 }
13367 }
13368 \cs_generate_variant:Nn
13369 \msg_new:nnnn
13370 { nnnV }
13371 \tl_set:Nn
13372 \l_tmpa_tl
13373 {
13374 You~may~need~to~run~TeX~with~the~---shell-escape~or~the~
13375 --enable-write18~flag,~or~write~shell_escape=t~in~the~
13376 texmf.cnf~file.
13377 }
13378 \msg_new:nnnV
13379 { markdown }
13380 { restricted-shell-access }
13381 { Shell~escape~is~restricted }
13382 \l_tmpa_tl
13383 \msg_new:nnnV
13384 { markdown }
13385 { disabled-shell-access }
13386 { Shell~escape~is~disabled }
13387 \l_tmpa_tl
13388 \ExplSyntaxOff

```

### 3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

13389 \ExplSyntaxOn

```

```

13390 \tl_new:N
13391 \g_@@_after_markinline_tl
13392 \tl_gset:Nn
13393 \g_@@_after_markinline_tl
13394 { \unskip }
13395 \cs_new:Npn
13396 \markinline
13397 {

```

Locally change the category of the special plain T<sub>E</sub>X characters to *other* in order to prevent unwanted interpretation of the input markdown text as T<sub>E</sub>X code.

```

13398 \group_begin:
13399 \cctab_select:N
13400 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

13401 \@@_if_option:nF
13402 { frozenCache }
13403 {
13404 \immediate
13405 \openout
13406 \markdownOutputFileStream
13407 \markdownOptionInputTempFileName
13408 \relax
13409 \msg_info:nne
13410 { markdown }
13411 { buffering-markinline }
13412 { \markdownOptionInputTempFileName }
13413 }

```

Peek ahead and extract the inline markdown text.

```

13414 \peek_regex_replace_once:nnF
13415 { { (.*) } }
13416 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

13417 \c { @@_if_option:nF }
13418 \cB { frozenCache \cE }
13419 \cB {
13420 \c { immediate }
13421 \c { write }
13422 \c { markdownOutputFileStream }
13423 \cB { \1 \cE }
13424 \c { immediate }
13425 \c { closeout }
13426 \c { markdownOutputFileStream }
13427 \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

13428 \c { group_end: }
13429 \c { group_begin: }
13430 \c { @@_setup:n }
13431 \cB { contentLevel = inline \cE }
13432 \c { markdownInput }
13433 \cB {
13434 \c { markdownOptionOutputDir } /
13435 \c { markdownOptionInputTempFileName }
13436 \cE }
13437 \c { group_end: }
13438 \c { tl_use:N }
13439 \c { g_@@_after_markinline_tl }
13440 }
13441 {
13442 \msg_error:nn
13443 { markdown }
13444 { markinline-peek-failure }
13445 \group_end:
13446 \tl_use:N
13447 \g_@@_after_markinline_tl
13448 }
13449 }
13450 \msg_new:nnn
13451 { markdown }
13452 { buffering-markinline }
13453 { Buffering~inline~markdown~input~into~
13454 the~temporary~input~file~"#1". }
13455 \msg_new:nnnn
13456 { markdown }
13457 { markinline-peek-failure }
13458 { Use~of~\iow_char:N \ markinline~doesn't~match~its~definition }
13459 { The~macro~should~be~followed~by~inline~
13460 markdown~text~in~curly~braces }
13461 \ExplSyntaxOff

```

### 3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T<sub>E</sub>X.

```

13462 \ExplSyntaxOn
13463 \cs_new:Npn
13464 \markdownInput
13465 #1
13466 {

```

```

13467 \@@_if_option:nTF
13468 { frozenCache }
13469 {
13470 \markdownInputRaw
13471 { #1 }
13472 }
13473 {

```

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  $\text{\LaTeX}$ , this also includes the directories specified in `\input@path`.

```

13474 \tl_set:Nx
13475 \l_tmpa_tl
13476 { #1 }
13477 \file_get_full_name:VNTF
13478 \l_tmpa_tl
13479 \l_tmpb_tl
13480 {
13481 \exp_args:NV
13482 \markdownInputRaw
13483 \l_tmpb_tl
13484 }
13485 {
13486 \msg_error:nnV
13487 { markdown }
13488 { markdown-file-does-not-exist }
13489 \l_tmpa_tl
13490 }
13491 }
13492 }
13493 \msg_new:nnn
13494 { markdown }
13495 { markdown-file-does-not-exist }
13496 {
13497 Markdown~file~#1~does-not~exist
13498 }
13499 \ExplSyntaxOff
13500 \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.

```

13501 \catcode`\|=0%
13502 \catcode`\|=12%
13503 \catcode`\&=6%
13504 \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.

```
13505 |begingroup
13506 |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.

```
13507 |catcode`\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
13508 |markdownIfOption{frozenCache}{%
13509 |ifnum|markdownOptionFrozenCacheCounter=0|relax
13510 |markdownInfo{Reading frozen cache from
13511 "||markdownOptionFrozenCacheFileName"}%
13512 |input|markdownOptionFrozenCacheFileName|relax
13513 |fi
13514 |markdownInfo{Including markdown document number
13515 "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
13516 |csname markdownFrozenCache%
13517 |the|markdownOptionFrozenCacheCounter|endcsname
13518 |global|advance|markdownOptionFrozenCacheCounter by 1|relax
13519 }{%
13520 |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L<sup>A</sup>T<sub>E</sub>X Mk to track changes to the markdown document.

```
13521 |openin|markdownInputFileStream&1
13522 |closein|markdownInputFileStream
13523 |markdownPrepareLuaOptions
13524 |markdownPrepareInputFilename{&1}%
13525 |markdownLuaExecute{%
13526 |markdownPrepare
13527 |markdownConvert
13528 |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
13529 |markdownIfOption{finalizeCache}{%
13530 |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
13531 }%
13532 |endgroup
13533 }%
13534 |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  $\text{\TeX}$  to execute a  $\text{\TeX}$  document in the middle of a markdown document fragment.

```

13535 \gdef\markdownEscape#1{%
13536 \catcode`\%=14\relax
13537 \catcode`\#=6\relax
13538 \input #1\relax
13539 \catcode`\%=12\relax
13540 \catcode`\#=12\relax
13541 }%
```

### 3.3 $\text{\LaTeX}$ Implementation

The  $\text{\LaTeX}$  implementation makes use of the fact that, apart from some subtle differences,  $\text{\LaTeX}$  implements the majority of the plain  $\text{\TeX}$  format [15, Section 9]. As a consequence, we can directly reuse the existing plain  $\text{\TeX}$  implementation.

```

13542 \def\markdownVersionSpace{ }%
13543 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
13544 \markdownVersion\markdownVersionSpace markdown renderer]%
```

#### 3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain  $\text{\TeX}$  implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the  $\text{\LaTeX}$  interface (see Section 2.3.3).

```

13545 \ExplSyntaxOn
13546 \cs_gset_eq:NN
13547 \markinlinePlainTeX
13548 \markinline
13549 \cs_gset:Npn
13550 \markinline
13551 {
13552 \peek_regex_replace_once:nn
13553 { (\[(.*) \]) ? }
13554 {
```

Apply the options locally.

```

13555 \c { group_begin: }
13556 \c { @@_setup:n }
13557 \cB { \2 \cE }
13558 \c { tl_put_right:Nn }
13559 \c { g_@@_after_markinline_tl }
13560 \cB { \c { group_end: } \cE }
```

```

13561 \c { markinlinePlainTeX }
13562 }
13563 }
13564 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain TeX implementation of the `\yamlInput` macro. The `\markdownInput` and `\yamlInput` macros are then redefined to accept an optional argument with options recognized by the L<sup>A</sup>T<sub>E</sub>X interface (see Section 2.3.3).

```

13565 \let\markdownInputPlainTeX\markdownInput
13566 \renewcommand\markdownInput[2] [] {%
13567 \begingroup
13568 \markdownSetup{#1}%
13569 \markdownInputPlainTeX{#2}%
13570 \endgroup}%
13571 \renewcommand\yamlInput[2] [] {%
13572 \begingroup
13573 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData, #1}%
13574 \markdownInputPlainTeX{#2}%
13575 \endgroup}%

```

The `markdown`, `markdown*`, and `yaml` L<sup>A</sup>T<sub>E</sub>X environments are implemented using the `\markdownReadAndConvert` macro.

```

13576 \ExplSyntaxOn
13577 \renewenvironment
13578 { markdown }
13579 {

```

In our implementation of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment, we want to distinguish between the following two cases:

<code>\begin{markdown} [smartEllipses]</code>	<code>\begin{markdown}</code>
<code>% This is an optional argument ^</code>	<code>[smartEllipses]</code>
<code>% ...</code>	<code>% ^ This is link</code>
<code>\end{markdown}</code>	<code>\end{markdown}</code>

Therefore, we cannot use the built-in L<sup>A</sup>T<sub>E</sub>X support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` L<sup>A</sup>T<sub>E</sub>X environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return

character will be produced by T<sub>E</sub>X via the `\endlinechar` plain T<sub>E</sub>X macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```
13580 \group_begin:
13581 \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).

```
13582 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening `[` that begins the optional argument, we accept carriage returns as well.

```
13583 \peek_regex_replace_once:nnF
13584 { \ *\[r*([~]*)\[^\r]* }
13585 {
```

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.

```
13586 \c { group_end: }
13587 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
13588 \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 L<sup>A</sup>T<sub>E</sub>X environment.

We also make provision for using the `\markdown` command as a part of a different L<sup>A</sup>T<sub>E</sub>X environment as follows:

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

```
13589 \c { exp_args:NV }
13590 \c { markdownReadAndConvert@ }
13591 \c { @currenvir }
13592 }
13593 {
13594 \group_end:
13595 \exp_args:NV
13596 \markdownReadAndConvert@
13597 \@currenvir
```

```

13598 }
13599 }
13600 { \markdownEnd }
13601 \renewenvironment
13602 { markdown* }
13603 [1]
13604 {
13605 \@@_if_option:nTF
13606 { experimental }
13607 {
13608 \msg_error:nnn
13609 { markdown }
13610 { latex-markdown-star-deprecated }
13611 { #1 }
13612 }
13613 {
13614 \msg_warning:nnn
13615 { markdown }
13616 { latex-markdown-star-deprecated }
13617 { #1 }
13618 }
13619 \@@_setup:n
13620 { #1 }
13621 \markdownReadAndConvert@
13622 { markdown* }
13623 }
13624 { \markdownEnd }
13625 \renewenvironment
13626 { yaml }
13627 {
13628 \group_begin:
13629 \yamlSetup{jekyllData, expectJekyllData, ensureJekyllData}%
13630 \markdown
13631 }
13632 { \yamlEnd }
13633 \msg_new:nnn
13634 { markdown }
13635 { latex-markdown-star-deprecated }
13636 {
13637 The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
13638 be~removed~in~the~next~major~version~of~the~Markdown~package.
13639 }
13640 \cs_generate_variant:Nn
13641 \@@_setup:n
13642 { V }
13643 \ExplSyntaxOff
13644 \begingroup

```

Locally swap the category code of the backslash symbol with the pipe symbol, and of the left (`{`) and right brace (`}`) with the less-than (`<`) and greater-than (`>`) signs. This is required in order that all the special symbols that appear in the first argument of the `markdownReadAndConvert` macro have the category code *other*.

```

13645 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
13646 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
13647 \gdef|markdownReadAndConvert@#1<%
13648 |markdownReadAndConvert<\end{#1}>%
13649 <|end<#1>>>%
13650 |endgroup

```

### 3.3.2 Themes

This section overrides the plain  $\text{\TeX}$  implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in  $\text{\LaTeX}$  themes provided with the Markdown package.

```

13651 \ExplSyntaxOn
13652 \prop_new:N \g_@@_latex_loaded_themes_linenos_prop
13653 \prop_new:N \g_@@_latex_loaded_themes_versions_prop
13654 \cs_gset:Nn
13655 \@@_load_theme:nnn
13656 {

```

If the Markdown package has already been loaded, determine whether a file named `markdowntheme<munged theme name>.sty` exists and whether we are still in the preamble.

```

13657 \ifmarkdownLaTeXLoaded
13658 \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load  $\text{\LaTeX}$  themes after the preamble. Otherwise, try loading a plain  $\text{\TeX}$  theme instead.

```

13659 \file_if_exist:nTF
13660 { markdown theme #3.sty }
13661 {
13662 \msg_error:nnn
13663 { markdown }
13664 { latex-theme-after-preamble }
13665 { #1 }
13666 }
13667 {
13668 \@@_plain_tex_load_theme:nnn
13669 { #1 }
13670 { #2 }
13671 { #3 }
13672 }
13673 \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L<sup>A</sup>T<sub>E</sub>X theme if it exists or load a plain T<sub>E</sub>X theme otherwise.

```

13674 \file_if_exist:nTF
13675 { markdown theme #3.sty }
13676 {
13677 \prop_get:NnNTF
13678 \g_@@_latex_loaded_themes_linenos_prop
13679 { #1 }
13680 \l_tmpa_tl
13681 {
13682 \prop_get:NnN
13683 \g_@@_latex_loaded_themes_versions_prop
13684 { #1 }
13685 \l_tmpb_tl
13686 \str_if_eq:nVTF
13687 { #2 }
13688 \l_tmpb_tl
13689 {
13690 \msg_warning:nnnVn
13691 { markdown }
13692 { repeatedly-loaded-latex-theme }
13693 { #1 }
13694 \l_tmpa_tl
13695 { #2 }
13696 }
13697 {
13698 \msg_error:nnnnVV
13699 { markdown }
13700 { different-versions-of-latex-theme }
13701 { #1 }
13702 { #2 }
13703 \l_tmpb_tl
13704 \l_tmpa_tl
13705 }
13706 }
13707 {
13708 \msg_info:nnnn
13709 { markdown }
13710 { loading-latex-theme }
13711 { #1 }
13712 { #2 }
13713 \prop_gput:Nnx
13714 \g_@@_latex_loaded_themes_linenos_prop
13715 { #1 }
13716 { \tex_the:D \tex_inputlineno:D }
13717 \prop_gput:Nnn
13718 \g_@@_latex_loaded_themes_versions_prop

```

```

13719 { #1 }
13720 { #2 }
13721 \RequirePackage
13722 { markdown theme #3 }
13723 }
13724 }
13725 {
13726 \@@_plain_tex_load_theme:nnn
13727 { #1 }
13728 { #2 }
13729 { #3 }
13730 }
13731 \fi
13732 \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

13733 \msg_info:nnnn
13734 { markdown }
13735 { theme-loading-postponed }
13736 { #1 }
13737 { #2 }
13738 \AtEndOfPackage
13739 {
13740 \@@_set_theme:n
13741 { #1 @ #2 }
13742 }
13743 \fi
13744 }
13745 \msg_new:nnn
13746 { markdown }
13747 { theme-loading-postponed }
13748 {
13749 Postponing~loading~version~#2~of~Markdown~theme~#1~until~
13750 Markdown~package~has~finished~loading
13751 }
13752 \msg_new:nnn
13753 { markdown }
13754 { loading-latex-theme }
13755 { Loading~version~#2~of~LaTeX~Markdown~theme~#1 }
13756 \msg_new:nnn
13757 { markdown }
13758 { repeatedly-loaded-latex-theme }
13759 {
13760 Version~#3~of~LaTeX~Markdown~theme~#1~was~previously~
13761 loaded~on~line~#2,~not~loading~it~again
13762 }

```



```

13763 \msg_new:nnn
13764 { markdown }
13765 { different-versions-of-latex-theme }
13766 {
13767 Tried-to-load-version-#2-of-LaTeX-Markdown-theme-#1~
13768 but-version-#3-has-already-been-loaded-on-line-#4
13769 }
13770 \cs_generate_variant:Nn
13771 \msg_new:nnnn
13772 { nnVV }
13773 \tl_set:Nn
13774 \l_tmpa_tl
13775 { Cannot-load-LaTeX-Markdown-theme-#1-after~ }
13776 \tl_put_right:NV
13777 \l_tmpa_tl
13778 \c_backslash_str
13779 \tl_put_right:Nn
13780 \l_tmpa_tl
13781 { begin{document} }
13782 \tl_set:Nn
13783 \l_tmpb_tl
13784 { Load-Markdown-theme-#1-before~ }
13785 \tl_put_right:NV
13786 \l_tmpb_tl
13787 \c_backslash_str
13788 \tl_put_right:Nn
13789 \l_tmpb_tl
13790 { begin{document} }
13791 \msg_new:nnVV
13792 { markdown }
13793 { latex-theme-after-preamble }
13794 \l_tmpa_tl
13795 \l_tmpb_tl
13796 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
13797 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
13798 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```

13799 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
13800 \markdownRendererInputFencedCodePrototype

```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T<sub>E</sub>X option is disabled and the code block has not been previously typeset:

```

13801 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
13802 \def\next##1 ##2\relax{%
13803 \ifthenelse{\equal{##1}{dot}}{%
13804 \markdownIfOption{frozenCache}{}{%
13805 \immediate\write18{%
13806 if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
13807 then
13808 dot -Tpdf -o #1.pdf #1;
13809 cp #1 #1.pdf.source;
13810 fi}}%

```

We include the typeset image using the image token renderer:

```

13811 \markdownRendererImage{Graphviz image}{#1.pdf}{#1.pdf}{##2}%

```

If the infostring does not start with `dot ...`, we use the previous definition of the fenced code token renderer prototype:

```

13812 }{%
13813 \markdown@witiko@dot@oldRendererInputFencedCodePrototype
13814 {#1}{#2}{#3}%
13815 }%
13816 }%
13817 \next#2 \relax}%

```

The [witiko/graphicx/http](#) theme stores the previous definition of the image token renderer prototype:

```

13818 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
13819 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```

13820 \RequirePackage{catchfile,grffile}

```

We define the `\markdown@witiko@graphicx@http@counter` counter to enumerate the images for caching and the `\markdown@witiko@graphicx@http@filename` command, which will store the pathname of the file containing the pathname of the downloaded image file.

```

13821 \newcount\markdown@witiko@graphicx@http@counter
13822 \markdown@witiko@graphicx@http@counter=0
13823 \newcommand\markdown@witiko@graphicx@http@filename{%
13824 \markdownOptionCacheDir/witiko_graphicx_http%
13825 .\the\markdown@witiko@graphicx@http@counter}%

```

We define the `\markdown@witiko@graphicx@http@download` command, which will receive two arguments that correspond to the URL of the online image and to the pathname, where the online image should be downloaded. The command will produce a shell command that tries to download the online image to the pathname.

```

13826 \newcommand\markdown@witiko@graphicx@http@download[2]{%
13827 wget -O #2 #1 || curl --location -o #2 #1 || rm -f #2}

```

We locally swap the category code of the percentage sign with the line feed control character, so that we can use percentage signs in the shell code:

```
13828 \begingroup
13829 \catcode`\%=12
13830 \catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
13831 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
13832 \begingroup
13833 \edef\filename{\markdown@witiko@graphicx@http\filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the `frozenCache` plain TeX option is disabled:

```
13834 \markdownIfOption{frozenCache}{}{^^A
13835 \immediate\write18{^^A
13836 mkdir -p "\markdownOptionCacheDir";
13837 if printf '%s' "#3" | grep -q -E '^\https?:';
13838 then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
13839 OUTPUT_PREFIX="\markdownOptionCacheDir";
13840 OUTPUT_BODY="$ (printf '%s' '#3' | md5sum | cut -d' ' -f1)";
13841 OUTPUT_SUFFIX="$ (printf '%s' '#3' | sed 's/.*[.]/')";
13842 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
13843 if ! [-e "$OUTPUT"];
13844 then
13845 \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
13846 printf '%s' "$OUTPUT" > "\filename";
13847 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:

```
13848 else
13849 printf '%s' '#3' > "\filename";
13850 fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
13851 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
13852 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
13853 {#1}{#2}{\filename}{#4}^^A
13854 \endgroup
13855 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
13856 \endgroup
```

The `witiko/markdown/defaults` L<sup>A</sup>T<sub>E</sub>X theme provides default definitions for token renderer prototypes. First, the L<sup>A</sup>T<sub>E</sub>X theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```
13857 \markdownLoadPlainTeXTheme
```

Next, the L<sup>A</sup>T<sub>E</sub>X theme overrides some of the plain T<sub>E</sub>X definitions. See Section 3.3.4 for the actual definitions.

### 3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```
13858 \DeclareOption*{%
13859 \expandafter\markdownSetup\expandafter{\CurrentOption}}%
13860 \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.

```
13861 \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`.

```
13862 \ExplSyntaxOn
13863 \bool_new:N
13864 \g_@@_tight_or_fancy_lists_bool
13865 \bool_gset_false:N
13866 \g_@@_tight_or_fancy_lists_bool
13867 \@@_if_option:nTF
13868 { tightLists }
13869 {
13870 \bool_gset_true:N
13871 \g_@@_tight_or_fancy_lists_bool
13872 }
13873 {
13874 \@@_if_option:nT
13875 { fancyLists }
13876 {
13877 \bool_gset_true:N
13878 \g_@@_tight_or_fancy_lists_bool
```

```

13879 }
13880 }
13881 \bool_new:N
13882 \g_@@_beamer_paralist_or_enumitem_bool
13883 \bool_gset_true:N
13884 \g_@@_beamer_paralist_or_enumitem_bool
13885 \@ifclassloaded
13886 { beamer }
13887 { }
13888 {
13889 \@ifpackageloaded
13890 { paralist }
13891 { }
13892 {
13893 \@ifpackageloaded
13894 { enumitem }
13895 { }
13896 {
13897 \bool_gset_false:N
13898 \g_@@_beamer_paralist_or_enumitem_bool
13899 }
13900 }
13901 }
13902 \bool_if:nT
13903 {
13904 \g_@@_tight_or_fancy_lists_bool &&
13905 ! \g_@@_beamer_paralist_or_enumitem_bool
13906 }
13907 {
13908 \bool_if:nTF
13909 {
13910 \bool_lazy_or_p:nn
13911 {
13912 \str_if_eq_p:en
13913 { \markdownThemeVersion }
13914 { experimental }
13915 }
13916 {
13917 \bool_lazy_and_p:nn
13918 {
13919 \prop_if_exist_p:N
13920 \g__pdfmanagement_documentproperties_prop
13921 }
13922 {
13923 \bool_lazy_any_p:n
13924 {
13925
```

```

13926 \prop_if_in_p:Nn
13927 \g__pdfmanagement_documentproperties_prop
13928 { document / testphase / phase-I }
13929 }
13930 {
13931 \prop_if_in_p:Nn
13932 \g__pdfmanagement_documentproperties_prop
13933 { document / testphase / phase-II }
13934 }
13935 {
13936 \prop_if_in_p:Nn
13937 \g__pdfmanagement_documentproperties_prop
13938 { document / testphase / phase-III }
13939 }
13940 {
13941 \prop_if_in_p:Nn
13942 \g__pdfmanagement_documentproperties_prop
13943 { document / testphase / phase-IV }
13944 }
13945 {
13946 \prop_if_in_p:Nn
13947 \g__pdfmanagement_documentproperties_prop
13948 { document / testphase / phase-V }
13949 }
13950 {
13951 \prop_if_in_p:Nn
13952 \g__pdfmanagement_documentproperties_prop
13953 { document / testphase / phase-VI }
13954 }
13955 }
13956 }
13957 }
13958 }
13959 {
13960 \RequirePackage
13961 { enumitem }
13962 }
13963 {
13964 \RequirePackage
13965 { paralist }
13966 }
13967 }
13968 \ExplSyntaxOff

```

If we loaded the enumitem package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

13969 \ExplSyntaxOn

```

```

13970 \cs_new:Nn
13971 \@@_latex_fancy_list_item_label_number:nn
13972 {
13973 \str_case:nn
13974 { #1 }
13975 {
13976 { Decimal } { #2 }
13977 { LowerRoman } { \int_to_roman:n { #2 } }
13978 { UpperRoman } { \int_to_Roman:n { #2 } }
13979 { LowerAlpha } { \int_to_alph:n { #2 } }
13980 { UpperAlpha } { \int_to_Alph:n { #2 } }
13981 }
13982 }
13983 \cs_new:Nn
13984 \@@_latex_fancy_list_item_label_delimiter:n
13985 {
13986 \str_case:nn
13987 { #1 }
13988 {
13989 { Default } { . }
13990 { OneParen } {) }
13991 { Period } { . }
13992 }
13993 }
13994 \cs_new:Nn
13995 \@@_latex_fancy_list_item_label:nnn
13996 {
13997 \@@_latex_fancy_list_item_label_number:nn
13998 { #1 }
13999 { #3 }
14000 \@@_latex_fancy_list_item_label_delimiter:n
14001 { #2 }
14002 }
14003 \cs_generate_variant:Nn
14004 \@@_latex_fancy_list_item_label:nnn
14005 { VVn }
14006 \tl_new:N
14007 \l_@@_latex_fancy_list_item_label_number_style_tl
14008 \tl_new:N
14009 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14010 \ifpackageloaded{enumitem}{
14011 \markdownSetup{rendererPrototypes={
First, let's define the tight list item renderer prototypes.
14012 ulBeginTight = {
14013 \begin
14014 { itemize }
14015 [noitemsep]

```

```

14016 },
14017 ulEndTight = {
14018 \end
14019 { itemize }
14020 },
14021 olBeginTight = {
14022 \begin
14023 { enumerate }
14024 [noitemsep]
14025 },
14026 olEndTight = {
14027 \end
14028 { enumerate }
14029 },
14030 dlBeginTight = {
14031 \begin
14032 { description }
14033 [noitemsep]
14034 },
14035 dlEndTight = {
14036 \end
14037 { description }
14038 },

```

Second, let's define the fancy list item renderer prototypes.

```

14039 fancyOlBegin = {
14040 \group_begin:
14041 \tl_set:Nn
14042 \l_@@_latex_fancy_list_item_label_number_style_tl
14043 { #1 }
14044 \tl_set:Nn
14045 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14046 { #2 }
14047 \begin
14048 { enumerate }
14049 },
14050 fancyOlBeginTight = {
14051 \group_begin:
14052 \tl_set:Nn
14053 \l_@@_latex_fancy_list_item_label_number_style_tl
14054 { #1 }
14055 \tl_set:Nn
14056 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14057 { #2 }
14058 \begin
14059 { enumerate }
14060 [noitemsep]
14061 },

```



```

14062 fancyOlEnd(|Tight) = {
14063 \end { enumerate }
14064 \group_end:
14065 },
14066 fancyOlItemWithNumber = {
14067 \item
14068 [
14069 \@@_latex_fancy_list_item_label:VVn
14070 \l_@@_latex_fancy_list_item_label_number_style_tl
14071 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14072 { #1 }
14073]
14074 },
14075 }}

```

Otherwise, if we loaded the paralist package, define the tight and fancy list renderer prototypes to make use of the capabilities of the package.

```

14076 }{\@ifpackageloaded{paralist}{
14077 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

14078 ulBeginTight = {%
14079 \group_begin:
14080 \pltopsep=\topsep
14081 \plpartopsep=\partopsep
14082 \begin{compactitem}
14083 },
14084 ulEndTight = {
14085 \end{compactitem}
14086 \group_end:
14087 },
14088 fancyOlBegin = {
14089 \group_begin:
14090 \tl_set:Nn
14091 \l_@@_latex_fancy_list_item_label_number_style_tl
14092 { #1 }
14093 \tl_set:Nn
14094 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14095 { #2 }
14096 \begin{enumerate}
14097 },
14098 fancyOlEnd = {
14099 \end{enumerate}
14100 \group_end:
14101 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

14102 olBeginTight = {%
14103 \group_begin:
14104 \plpartopsep=\partopsep
14105 \pltopsep=\topsep
14106 \begin{compactenum}
14107 },
14108 olEndTight = {
14109 \end{compactenum}
14110 \group_end:
14111 },
14112 fancyOlBeginTight = {
14113 \group_begin:
14114 \tl_set:Nn
14115 \l_@@_latex_fancy_list_item_label_number_style_tl
14116 { #1 }
14117 \tl_set:Nn
14118 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14119 { #2 }
14120 \plpartopsep=\partopsep
14121 \pltopsep=\topsep
14122 \begin{compactenum}
14123 },
14124 fancyOlEndTight = {
14125 \end{compactenum}
14126 \group_end:
14127 },
14128 fancyOlItemWithNumber = {
14129 \item
14130 [
14131 \@@_latex_fancy_list_item_label:VVn
14132 \l_@@_latex_fancy_list_item_label_number_style_tl
14133 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
14134 { #1 }
14135]
14136 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

14137 dlBeginTight = {
14138 \group_begin:
14139 \plpartopsep=\partopsep
14140 \pltopsep=\topsep
14141 \begin{compactdesc}
14142 },
14143 dlEndTight = {

```

```

14144 \end{compactdesc}
14145 \group_end:
14146 }
14147 }}
14148 }{

```

Otherwise, if we loaded neither the enumitem package nor the paralist package, define the tight and fancy list renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

14149 \markdownSetup
14150 {
14151 rendererPrototypes = {
14152 ulBeginTight = \markdownRendererUlBegin,
14153 ulEndTight = \markdownRendererUlEnd,
14154 fancyOlBegin = \markdownRendererOlBegin,
14155 fancyOlEnd = \markdownRendererOlEnd,
14156 olBeginTight = \markdownRendererOlBegin,
14157 olEndTight = \markdownRendererOlEnd,
14158 fancyOlBeginTight = \markdownRendererOlBegin,
14159 fancyOlEndTight = \markdownRendererOlEnd,
14160 dlBeginTight = \markdownRendererDlBegin,
14161 dlEndTight = \markdownRendererDlEnd,
14162 },
14163 }
14164 }}
14165 \ExplSyntaxOff
14166 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

14167 \@ifpackageloaded{unicode-math}{
14168 \markdownSetup{rendererPrototypes={
14169 untickedBox = {\mdlgwhtsquare},
14170 }}
14171 }{
14172 \RequirePackage{amssymb}
14173 \markdownSetup{rendererPrototypes={
14174 untickedBox = {\square},
14175 }}
14176 }
14177 \RequirePackage{csvsimple}
14178 \RequirePackage{fancyvrb}
14179 \RequirePackage{graphicx}
14180 \markdownSetup{rendererPrototypes={
14181 hardLineBreak = {\},
14182 leftBrace = {\textbraceleft},
14183 rightBrace = {\textbraceright},
14184 dollarSign = {\textdollar},

```

```

14185 underscore = {\textunderscore},
14186 circumflex = {\textasciicircum},
14187 backslash = {\textbackslash},
14188 tilde = {\textasciitilde},
14189 pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T<sub>E</sub>X during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,<sup>34</sup> we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

14190 codeSpan = {%
14191 \ifmmode
14192 \text{#1}%
14193 \else
14194 \texttt{#1}%
14195 \fi
14196 }%
14197 \ExplSyntaxOn
14198 \markdownSetup{
14199 rendererPrototypes = {
14200 contentBlock = {
14201 \str_case:nnF
14202 { #1 }
14203 {
14204 { csv }
14205 {
14206 \begin{table}
14207 \begin{center}
14208 \csvautotabular{#3}
14209 \end{center}
14210 \tl_if_empty:nF
14211 { #4 }
14212 { \caption{#4} }
14213 \end{table}
14214 }
14215 { tex } { \markdownEscape{#3} }
14216 }
14217 { \markdownInput{#3} }
14218 },
14219 },
14220 }

```

---

<sup>34</sup>This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

14221 \ExplSyntaxOff
14222 \markdownSetup{rendererPrototypes={
14223 ulBegin = {\begin{itemize}},
14224 ulEnd = {\end{itemize}},
14225 olBegin = {\begin{enumerate}},
14226 olItem = {\item{}},
14227 olItemWithNumber = {\item[#1.]},
14228 olEnd = {\end{enumerate}},
14229 dlBegin = {\begin{description}},
14230 dlItem = {\item[#1]},
14231 dlEnd = {\end{description}},
14232 emphasis = {\emph{#1}},
14233 tickedBox = {\\boxtimes},
14234 halfTickedBox = {\\boxdot}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

14235 \ExplSyntaxOn
14236 \seq_new:N
14237 \l_@@_header_identifiers_seq
14238 \markdownSetup
14239 {
14240 rendererPrototypes = {
14241 headerAttributeContextBegin = {
14242 \markdownSetup
14243 {
14244 rendererPrototypes = {
14245 attributeIdentifier = {
14246 \seq_put_right:Nn
14247 \l_@@_header_identifiers_seq
14248 { ##1 }
14249 },
14250 },
14251 },
14252 },
14253 headerAttributeContextEnd = {
14254 \seq_map_inline:Nn
14255 \l_@@_header_identifiers_seq
14256 { \label { ##1 } }
14257 \seq_clear:N
14258 \l_@@_header_identifiers_seq
14259 },
14260 },
14261 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

14262 \bool_new:N
14263 \l_@@_header_unnumbered_bool

```

```

14264 \markdownSetup
14265 {
14266 rendererPrototypes = {
14267 headerAttributeContextBegin += {
14268 \markdownSetup
14269 {
14270 rendererPrototypes = {
14271 attributeClassName = {
14272 \bool_if:nT
14273 {
14274 \str_if_eq_p:nn
14275 { ##1 }
14276 { unnumbered } &&
14277 ! \l_@@_header_unnumbered_bool
14278 }
14279 {
14280 \group_begin:
14281 \bool_set_true:N
14282 \l_@@_header_unnumbered_bool
14283 \c@secnumdepth = 0
14284 \markdownSetup
14285 {
14286 rendererPrototypes = {
14287 sectionBegin = {
14288 \group_begin:
14289 },
14290 sectionEnd = {
14291 \group_end:
14292 },
14293 },
14294 }
14295 },
14296 },
14297 },
14298 },
14299 },
14300 },
14301 }
14302 \ExplSyntaxOff
14303 \markdownSetup{rendererPrototypes={
14304 superscript = {#1},
14305 subscript = {\textsubscript{#1}},
14306 blockQuoteBegin = {\begin{quotation}},
14307 blockQuoteEnd = {\end{quotation}},
14308 inputVerbatim = {\VerbatimInput{#1}},
14309 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
14310 note = {\footnote{#1}}}}

```

### 3.3.4.2 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
14311 \RequirePackage{ltxcmds}
14312 \ExplSyntaxOn
14313 \cs_gset:Npn
14314 \markdownRendererInputFencedCodePrototype#1#2#3
14315 {
14316 \tl_if_empty:nTF
14317 { #2 }
14318 { \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.

```
14319 {
14320 \regex_extract_once:nnN
14321 { \w* }
14322 { #2 }
14323 \l_tmpa_seq
14324 \seq_pop_left:NN
14325 \l_tmpa_seq
14326 \l_tmpa_tl
```

When the minted package is loaded, use it for syntax highlighting.

```
14327 \ltx@ifpackageloaded
14328 { minted }
14329 {
14330 \catcode`\%=14\relax
14331 \catcode`\#=6\relax
14332 \exp_args:NV
14333 \inputminted
14334 \l_tmpa_tl
14335 { #1 }
14336 \catcode`\%=12\relax
14337 \catcode`\#=12\relax
14338 }
14339 {
```

When the listings package is loaded, use it for syntax highlighting.

```
14340 \ltx@ifpackageloaded
14341 { listings }
14342 { \lstinputlisting[language=\l_tmpa_tl]{#1} }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
14343 { \markdownRendererInputFencedCode{#1}{}{} }
14344 }
14345 }
14346 }
```

```

14347 \ExplSyntaxOff
 Support the nesting of strong emphasis.
14348 \ExplSyntaxOn
14349 \def\markdownLATEXStrongEmphasis#1{%
14350 \str_if_in:NnTF
14351 \f@series
14352 { b }
14353 { \textnormal{#1} }
14354 { \textbf{#1} }
14355 }
14356 \ExplSyntaxOff
14357 \markdownSetup{rendererPrototypes={strongEmphasis={%
14358 \protect\markdownLATEXStrongEmphasis{#1}}}}
 Support LATEX document classes that do not provide chapters.
14359 \@ifundefined{chapter}{%
14360 \markdownSetup{rendererPrototypes = {
14361 headingOne = {\section{#1}},
14362 headingTwo = {\subsection{#1}},
14363 headingThree = {\subsubsection{#1}},
14364 headingFour = {\paragraph{#1}},
14365 headingFive = {\subparagraph{#1}}}}
14366 }{%
14367 \markdownSetup{rendererPrototypes = {
14368 headingOne = {\chapter{#1}},
14369 headingTwo = {\section{#1}},
14370 headingThree = {\subsection{#1}},
14371 headingFour = {\subsubsection{#1}},
14372 headingFive = {\paragraph{#1}},
14373 headingSix = {\subparagraph{#1}}}}
14374 }%

```

### 3.3.4.3 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

14375 \markdownSetup{
14376 rendererPrototypes = {
14377 ulItem = {%
14378 \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
14379 },
14380 },
14381 }
14382 \def\markdownLaTeXUListItem{%
14383 \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
14384 \item[\markdownLaTeXCheckbox]%
14385 \expandafter\@gobble

```



```

14386 \else
14387 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
14388 \item[\markdownLaTeXCheckbox]%
14389 \expandafter\expandafter\expandafter\@gobble
14390 \else
14391 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
14392 \item[\markdownLaTeXCheckbox]%
14393 \expandafter\expandafter\expandafter\expandafter
14394 \expandafter\expandafter\expandafter\@gobble
14395 \else
14396 \item{}%
14397 \fi
14398 \fi
14399 \fi
14400 }

```

### 3.3.4.4 HTML elements

If the `html` option is enabled and we are using  $\text{\TeX 4ht}$ <sup>35</sup>, we will pass HTML elements to the output HTML document unchanged.

```

14401 \@ifundefined{HCode}{}{
14402 \markdownSetup{
14403 rendererPrototypes = {
14404 inlineHtmlTag = {%
14405 \ifvmode
14406 \IgnorePar
14407 \EndP
14408 \fi
14409 \HCode{#1}%
14410 },
14411 inputBlockHtmlElement = {%
14412 \ifvmode
14413 \IgnorePar
14414 \fi
14415 \EndP
14416 \special{t4ht*<#1}%
14417 \par
14418 \ShowPar
14419 },
14420 },
14421 }
14422 }

```

### 3.3.4.5 Citations

---

<sup>35</sup>See <https://tug.org/tex4ht/>.

Here is a basic implementation for citations that uses the L<sup>A</sup>T<sub>E</sub>X `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL<sup>A</sup>T<sub>E</sub>X `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

14423 \newcount\markdownLaTeXCitationsCounter
14424
14425 % Basic implementation
14426 \RequirePackage{gobble}
14427 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
14428 \advance\markdownLaTeXCitationsCounter by 1\relax
14429 \ifx\relax#4\relax
14430 \ifx\relax#5\relax
14431 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14432 \relax
14433 \cite{#1#2#6}% No prenotes/postnotes, just accumulate cites
14434 \expandafter\expandafter\expandafter
14435 \expandafter\expandafter\expandafter\expandafter
14436 \@gobblethree
14437 \fi
14438 \else% Before a postnote (#5), dump the accumulator
14439 \ifx\relax#1\relax\else
14440 \cite{#1}%
14441 \fi
14442 \cite[#5]{#6}%
14443 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14444 \relax
14445 \else
14446 \expandafter\expandafter\expandafter
14447 \expandafter\expandafter\expandafter\expandafter
14448 \expandafter\expandafter\expandafter
14449 \expandafter\expandafter\expandafter\expandafter
14450 \markdownLaTeXBasicCitations
14451 \fi
14452 \expandafter\expandafter\expandafter
14453 \expandafter\expandafter\expandafter\expandafter{%
14454 \expandafter\expandafter\expandafter
14455 \expandafter\expandafter\expandafter\expandafter}%
14456 \expandafter\expandafter\expandafter
14457 \expandafter\expandafter\expandafter\expandafter{%
14458 \expandafter\expandafter\expandafter
14459 \expandafter\expandafter\expandafter\expandafter}%
14460 \expandafter\expandafter\expandafter
14461 \@gobblethree
14462 \fi
14463 \else% Before a prenote (#4), dump the accumulator
14464 \ifx\relax#1\relax\else

```

```

14465 \cite{#1}%
14466 \fi
14467 \ifnum\markdownLaTeXCitationsCounter>1\relax
14468 \space % Insert a space before the prenote in later citations
14469 \fi
14470 #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
14471 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14472 \relax
14473 \else
14474 \expandafter\expandafter\expandafter
14475 \expandafter\expandafter\expandafter\expandafter
14476 \markdownLaTeXBasicCitations
14477 \fi
14478 \expandafter\expandafter\expandafter{%
14479 \expandafter\expandafter\expandafter}%
14480 \expandafter\expandafter\expandafter{%
14481 \expandafter\expandafter\expandafter}%
14482 \expandafter
14483 \@gobblethree
14484 \fi\markdownLaTeXBasicCitations{#1#2#6},}
14485 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
14486
14487 % Natbib implementation
14488 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
14489 \advance\markdownLaTeXCitationsCounter by 1\relax
14490 \ifx\relax#3\relax
14491 \ifx\relax#4\relax
14492 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14493 \relax
14494 \citep{#1,#5}% No prenotes/postnotes, just accumulate cites
14495 \expandafter\expandafter\expandafter
14496 \expandafter\expandafter\expandafter\expandafter
14497 \@gobbletwo
14498 \fi
14499 \else% Before a postnote (#4), dump the accumulator
14500 \ifx\relax#1\relax\else
14501 \citep{#1}%
14502 \fi
14503 \citep[] [#4]{#5}%
14504 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14505 \relax
14506 \else
14507 \expandafter\expandafter\expandafter
14508 \expandafter\expandafter\expandafter\expandafter
14509 \expandafter\expandafter\expandafter
14510 \expandafter\expandafter\expandafter\expandafter
14511 \markdownLaTeXNatbibCitations

```

```

14512 \fi
14513 \expandafter\expandafter\expandafter
14514 \expandafter\expandafter\expandafter\expandafter{%
14515 \expandafter\expandafter\expandafter
14516 \expandafter\expandafter\expandafter\expandafter}%
14517 \expandafter\expandafter\expandafter
14518 \@gobbletwo
14519 \fi
14520 \else% Before a prenote (#3), dump the accumulator
14521 \ifx\relax#1\relax\relax\else
14522 \citep{#1}%
14523 \fi
14524 \citep[#3][#4]{#5}%
14525 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14526 \relax
14527 \else
14528 \expandafter\expandafter\expandafter
14529 \expandafter\expandafter\expandafter\expandafter
14530 \markdownLaTeXNatbibCitations
14531 \fi
14532 \expandafter\expandafter\expandafter{%
14533 \expandafter\expandafter\expandafter}%
14534 \expandafter
14535 \@gobbletwo
14536 \fi\markdownLaTeXNatbibCitations{#1,#5}}
14537 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
14538 \advance\markdownLaTeXCitationsCounter by 1\relax
14539 \ifx\relax#3\relax
14540 \ifx\relax#4\relax
14541 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14542 \relax
14543 \citet{#1,#5}% No prenotes/postnotes, just accumulate cites
14544 \expandafter\expandafter\expandafter
14545 \expandafter\expandafter\expandafter\expandafter
14546 \@gobbletwo
14547 \fi
14548 \else% After a prenote or a postnote, dump the accumulator
14549 \ifx\relax#1\relax\else
14550 \citet{#1}%
14551 \fi
14552 , \citet[#3][#4]{#5}%
14553 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
14554 \relax
14555 ,
14556 \else
14557 \ifnum
14558 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal

```

```

14559 \relax
14560 ,
14561 \fi
14562 \fi
14563 \expandafter\expandafter\expandafter
14564 \expandafter\expandafter\expandafter\expandafter
14565 \markdownLaTeXNatbibTextCitations
14566 \expandafter\expandafter\expandafter
14567 \expandafter\expandafter\expandafter\expandafter{%
14568 \expandafter\expandafter\expandafter
14569 \expandafter\expandafter\expandafter\expandafter}%
14570 \expandafter\expandafter\expandafter
14571 \@gobbletwo
14572 \fi
14573 \else% After a prenote or a postnote, dump the accumulator
14574 \ifx\relax#1\relax\relax\else
14575 \citet{#1}%
14576 \fi
14577 , \citet[#3][#4]{#5}%
14578 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal
14579 \relax
14580 ,
14581 \else
14582 \ifnum
14583 \markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal
14584 \relax
14585 ,
14586 \fi
14587 \fi
14588 \expandafter\expandafter\expandafter
14589 \markdownLaTeXNatbibTextCitations
14590 \expandafter\expandafter\expandafter{%
14591 \expandafter\expandafter\expandafter}%
14592 \expandafter
14593 \@gobbletwo
14594 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
14595
14596 % BibLaTeX implementation
14597 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
14598 \advance\markdownLaTeXCitationsCounter by 1\relax
14599 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14600 \relax
14601 \autocites#1[#3][#4]{#5}%
14602 \expandafter\@gobbletwo
14603 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
14604 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
14605 \advance\markdownLaTeXCitationsCounter by 1\relax

```

```

14606 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal
14607 \relax
14608 \textcites#1[#3][#4]{#5}%
14609 \expandafter\@gobbletwo
14610 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
14611
14612 \markdownSetup{rendererPrototypes = {
14613 cite = {%
14614 \markdownLaTeXCitationsCounter=1%
14615 \def\markdownLaTeXCitationsTotal{#1}%
14616 \@ifundefined{autocites}{%
14617 \@ifundefined{citep}{%
14618 \expandafter\expandafter\expandafter
14619 \markdownLaTeXBasicCitations
14620 \expandafter\expandafter\expandafter{%
14621 \expandafter\expandafter\expandafter}%
14622 \expandafter\expandafter\expandafter{%
14623 \expandafter\expandafter\expandafter}%
14624 }{%
14625 \expandafter\expandafter\expandafter
14626 \markdownLaTeXNatbibCitations
14627 \expandafter\expandafter\expandafter{%
14628 \expandafter\expandafter\expandafter}%
14629 }%
14630 }{%
14631 \expandafter\expandafter\expandafter
14632 \markdownLaTeXBibLaTeXCitations
14633 \expandafter{\expandafter}%
14634 }},
14635 textCite = {%
14636 \markdownLaTeXCitationsCounter=1%
14637 \def\markdownLaTeXCitationsTotal{#1}%
14638 \@ifundefined{autocites}{%
14639 \@ifundefined{citep}{%
14640 \expandafter\expandafter\expandafter
14641 \markdownLaTeXBasicTextCitations
14642 \expandafter\expandafter\expandafter{%
14643 \expandafter\expandafter\expandafter}%
14644 \expandafter\expandafter\expandafter{%
14645 \expandafter\expandafter\expandafter}%
14646 }{%
14647 \expandafter\expandafter\expandafter
14648 \markdownLaTeXNatbibTextCitations
14649 \expandafter\expandafter\expandafter{%
14650 \expandafter\expandafter\expandafter}%
14651 }%
14652 }{%

```

```

14653 \expandafter\expandafter\expandafter
14654 \markdownLaTeXBibLaTeXTextCitations
14655 \expandafter{\expandafter}%
14656 }}}}

```

### 3.3.4.6 Links

Here is an implementation for hypertext links and relative references.

```

14657 \RequirePackage{url}
14658 \RequirePackage{expl3}
14659 \ExplSyntaxOn
14660 \def\markdownRendererLinkPrototype#1#2#3#4{
14661 \tl_set:Nn \l_tmpa_tl { #1 }
14662 \tl_set:Nn \l_tmpb_tl { #2 }
14663 \bool_set:Nn
14664 \l_tmpa_bool
14665 {
14666 \tl_if_eq_p:NN
14667 \l_tmpa_tl
14668 \l_tmpb_tl
14669 }
14670 \tl_set:Nn \l_tmpa_tl { #4 }
14671 \bool_set:Nn
14672 \l_tmpb_bool
14673 {
14674 \tl_if_empty_p:N
14675 \l_tmpa_tl
14676 }

```

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.

```

14677 \bool_if:nTF
14678 {
14679 \l_tmpa_bool && \l_tmpb_bool
14680 }
14681 {
14682 \markdownLaTeXRendererAutolink { #2 } { #3 }
14683 }{
14684 \markdownLaTeXRendererDirectOrIndirectLink
14685 { #1 } { #2 } { #3 } { #4 }
14686 }
14687 }
14688 \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.

```

14689 \tl_set:Nn

```

```

14690 \l_tmpa_tl
14691 { #2 }
14692 \tl_trim_spaces:N
14693 \l_tmpa_tl
14694 \tl_set:Nx
14695 \l_tmpb_tl
14696 {
14697 \tl_range:Nnn
14698 \l_tmpa_tl
14699 { 1 }
14700 { 1 }
14701 }
14702 \str_if_eq:NNTF
14703 \l_tmpb_tl
14704 \c_hash_str
14705 {
14706 \tl_set:Nx
14707 \l_tmpb_tl
14708 {
14709 \tl_range:Nnn
14710 \l_tmpa_tl
14711 { 2 }
14712 { -1 }
14713 }
14714 \exp_args:NV
14715 \ref
14716 \l_tmpb_tl
14717 }{
14718 \url { #2 }
14719 }
14720 }
14721 \ExplSyntaxOff
14722 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
14723 #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}%

```

### 3.3.4.7 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

14724 \newcount\markdownLaTeXRowCount
14725 \newcount\markdownLaTeXRowTotal
14726 \newcount\markdownLaTeXColumnCounter
14727 \newcount\markdownLaTeXColumnTotal
14728 \newtoks\markdownLaTeXTable
14729 \newtoks\markdownLaTeXTableAlignment
14730 \newtoks\markdownLaTeXTableEnd
14731 \AtBeginDocument{%

```



```

14732 \@ifpackageloaded{booktabs}{%
14733 \def\markdownLaTeXTopRule{\toprule}%
14734 \def\markdownLaTeXMidRule{\midrule}%
14735 \def\markdownLaTeXBottomRule{\bottomrule}%
14736 }{%
14737 \def\markdownLaTeXTopRule{\hline}%
14738 \def\markdownLaTeXMidRule{\hline}%
14739 \def\markdownLaTeXBottomRule{\hline}%
14740 }%
14741 }
14742 \markdownSetup{rendererPrototypes={
14743 table = {%
14744 \markdownLaTeXTable={}%
14745 \markdownLaTeXTableAlignment={}%
14746 \markdownLaTeXTableEnd={%
14747 \markdownLaTeXBottomRule
14748 \end{tabular}}%
14749 \ifx\empty#1\empty\else
14750 \addto@hook\markdownLaTeXTable{%
14751 \begin{table}
14752 \centering}%
14753 \addto@hook\markdownLaTeXTableEnd{%
14754 \caption{#1}
14755 \end{table}}%
14756 \fi
14757 \addto@hook\markdownLaTeXTable{\begin{tabular}}%
14758 \markdownLaTeXRowCount=0%
14759 \markdownLaTeXRowTotal=#2%
14760 \markdownLaTeXColumnTotal=#3%
14761 \markdownLaTeXRenderTableRow
14762 }
14763 }}
14764 \def\markdownLaTeXRenderTableRow#1{%
14765 \markdownLaTeXColumnCounter=0%
14766 \ifnum\markdownLaTeXRowCount=0\relax
14767 \markdownLaTeXReadAlignments#1%
14768 \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
14769 \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
14770 \the\markdownLaTeXTableAlignment}}%
14771 \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
14772 \else
14773 \markdownLaTeXRenderTableCell#1%
14774 \fi
14775 \ifnum\markdownLaTeXRowCount=1\relax
14776 \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
14777 \fi
14778 \advance\markdownLaTeXRowCount by 1\relax

```

```

14779 \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
14780 \the\markdownLaTeXTable
14781 \the\markdownLaTeXTableEnd
14782 \expandafter\@gobble
14783 \fi\markdownLaTeXRenderTableRow}
14784 \def\markdownLaTeXReadAlignments#1{%
14785 \advance\markdownLaTeXColumnCounter by 1\relax
14786 \if#1d%
14787 \addto@hook\markdownLaTeXTableAlignment{1}%
14788 \else
14789 \addto@hook\markdownLaTeXTableAlignment{#1}%
14790 \fi
14791 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
14792 \expandafter\@gobble
14793 \fi\markdownLaTeXReadAlignments}
14794 \def\markdownLaTeXRenderTableCell#1{%
14795 \advance\markdownLaTeXColumnCounter by 1\relax
14796 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
14797 \addto@hook\markdownLaTeXTable{#1&}%
14798 \else
14799 \addto@hook\markdownLaTeXTable{#1\\}%
14800 \expandafter\@gobble
14801 \fi\markdownLaTeXRenderTableCell}

```

### 3.3.4.8 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

14802
14803 \markdownIfOption{lineBlocks}{%
14804 \RequirePackage{verse}
14805 \markdownSetup{rendererPrototypes={
14806 lineBlockBegin = {%
14807 \begingroup
14808 \def\markdownRendererHardLineBreak{\\}%
14809 \begin{verse}%
14810 },
14811 lineBlockEnd = {%
14812 \end{verse}%
14813 \endgroup
14814 },
14815 }}
14816 }{}
14817

```

### 3.3.4.9 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.

```

14818 \ExplSyntaxOn
14819 \keys_define:nn
14820 { markdown/jekyllData }
14821 {
14822 author .code:n = { \author{#1} },
14823 date .code:n = { \date{#1} },
14824 title .code:n = { \title{#1} },
14825 }

```

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.

```

14826 \markdownSetup{
14827 rendererPrototypes = {
14828 jekyllDataEnd = {
14829 \AddToHook{begindocument/end}{\maketitle}
14830 },
14831 },
14832 }

```

### 3.3.4.10 Marked Text

If the `mark` option is enabled, we will load either the `soulutf8` package or the `lua-ul` package and use it to implement marked text.

```

14833 \@@_if_option:nT
14834 { mark }
14835 {
14836 \sys_if_engine luatex:TF
14837 {
14838 \RequirePackage
14839 { luacolor }
14840 \RequirePackage
14841 { lua-ul }
14842 \markdownSetup
14843 {
14844 rendererPrototypes = {
14845 mark = {
14846 \highLight
14847 { #1 }
14848 },
14849 }
14850 }
14851 }

```

```

14852 {
14853 \RequirePackage
14854 { xcolor }
14855 % TODO: Use just package soul after TeX Live 2023.
14856 \IfFormatAtLeastTF
14857 { 2023-02-18 }
14858 {
14859 \RequirePackage
14860 { soul }
14861 }
14862 {
14863 \RequirePackage
14864 { soulutf8 }
14865 }
14866 \markdownSetup
14867 {
14868 rendererPrototypes = {
14869 mark = {
14870 \hl
14871 { #1 }
14872 },
14873 }
14874 }
14875 }
14876 }

```

### 3.3.4.11 Strike-Through

If the `strikeThrough` option is enabled, we will load either the `soulutf8` package or the `lua-ul` package and use it to implement strike-throughs.

```

14877 \@@_if_option:nT
14878 { strikeThrough }
14879 {
14880 \sys_if_engine_luatex:TF
14881 {
14882 \RequirePackage
14883 { lua-ul }
14884 \markdownSetup
14885 {
14886 rendererPrototypes = {
14887 strikeThrough = {
14888 \strikeThrough
14889 { #1 }
14890 },
14891 }
14892 }
14893 }

```

```

14894 {
14895 % TODO: Use just package soul after TeX Live 2023.
14896 \IfFormatAtLeastTF
14897 { 2023-02-18 }
14898 {
14899 \RequirePackage
14900 { soul }
14901 }
14902 {
14903 \RequirePackage
14904 { soulutf8 }
14905 }
14906 \markdownSetup
14907 {
14908 rendererPrototypes = {
14909 strikeThrough = {
14910 \st
14911 { #1 }
14912 },
14913 }
14914 }
14915 }
14916 }

```

### 3.3.4.12 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  $\text{\LaTeX}$  labels for referencing figures.

```

14917 \ExplSyntaxOn
14918 \seq_new:N
14919 \l_@@_image_identifiers_seq
14920 \markdownSetup {
14921 rendererPrototypes = {
14922 image = {
14923 \begin { figure }
14924 \begin { center }
14925 \includegraphics
14926 [alt = { #1 }]
14927 { #3 }
14928 \tl_if_empty:nF
14929 { #4 }
14930 { \caption { #4 } }

```

```

14931 \seq_map_inline:Nn
14932 \l_@@_image_identifiers_seq
14933 { \label { ##1 } }
14934 \end { center }
14935 \end { figure }
14936 },
14937 }
14938 }
14939 \@@_if_option:nT
14940 { linkAttributes }
14941 {
14942 \RequirePackage { graphicx }
14943 \markdownSetup {
14944 rendererPrototypes = {
14945 imageAttributeContextBegin = {
14946 \group_begin:
14947 \markdownSetup {
14948 rendererPrototypes = {
14949 attributeIdentifier = {
14950 \seq_put_right:Nn
14951 \l_@@_image_identifiers_seq
14952 { ##1 }
14953 },
14954 attributeKeyValue = {
14955 \setkeys
14956 { Gin }
14957 { { ##1 } = { ##2 } }
14958 },
14959 },
14960 }
14961 },
14962 imageAttributeContextEnd = {
14963 \group_end:
14964 },
14965 },
14966 }
14967 }
14968 \ExplSyntaxOff

```

### 3.3.4.13 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

14969 \ExplSyntaxOn
14970 \cs_gset:Npn
14971 \markdownRendererInputRawInlinePrototype#1#2
14972 {

```

```

14973 \str_case:nnF
14974 { #2 }
14975 {
14976 { latex }
14977 {
14978 \@@_plain_tex_default_input_raw_inline:nn
14979 { #1 }
14980 { tex }
14981 }
14982 }
14983 {
14984 \@@_plain_tex_default_input_raw_inline:nn
14985 { #1 }
14986 { #2 }
14987 }
14988 }
14989 \cs_gset:Npn
14990 \markdownRendererInputRawBlockPrototype#1#2
14991 {
14992 \str_case:nnF
14993 { #2 }
14994 {
14995 { latex }
14996 {
14997 \@@_plain_tex_default_input_raw_block:nn
14998 { #1 }
14999 { tex }
15000 }
15001 }
15002 {
15003 \@@_plain_tex_default_input_raw_block:nn
15004 { #1 }
15005 { #2 }
15006 }
15007 }
15008 \ExplSyntaxOff
15009 \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.

```

15010 \newcommand\markdownMakeOther{%
15011 \count0=128\relax

```

```

15012 \loop
15013 \catcode\count0=11\relax
15014 \advance\count0 by 1\relax
15015 \ifnum\count0<256\repeat}%

```

### 3.4 ConT<sub>E</sub>Xt Implementation

The ConT<sub>E</sub>Xt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT<sub>E</sub>Xt formats *seem* to implement (the documentation is scarce) the majority of the plain T<sub>E</sub>X format required by the plain T<sub>E</sub>X implementation. As a consequence, we can directly reuse the existing plain T<sub>E</sub>X implementation after supplying the missing plain T<sub>E</sub>X macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L<sup>A</sup>T<sub>E</sub>X package.

```

15016 \def\markdownMakeOther{%
15017 \count0=128\relax
15018 \loop
15019 \catcode\count0=11\relax
15020 \advance\count0 by 1\relax
15021 \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 ConT<sub>E</sub>Xt.

```

15022 \catcode`|=12}%

```

#### 3.4.1 Typesetting Markdown

The `\inputmarkdown` and `\inputyaml` macros are defined to accept an optional argument with options recognized by the ConT<sub>E</sub>Xt interface (see Section 2.4.2).

```

15023 \long\def\inputmarkdown{%
15024 \dosingleempty
15025 \doinputmarkdown}%
15026 \long\def\doinputmarkdown[#1]#2{%
15027 \begingroup
15028 \iffirstargument
15029 \setupmarkdown[#1]%
15030 \fi
15031 \markdownInput{#2}%
15032 \endgroup}%
15033 \long\def\inputyaml{%
15034 \dosingleempty
15035 \doinputyaml}%
15036 \long\def\doinputyaml[#1]#2{%

```



```

15037 \doinputmarkdown
15038 [jekyllData, expectJekyllData, ensureJekyllData, #1]{#2}}%

```

The `\startmarkdown`, `\stopmarkdown`, `\startyaml`, and `\stopyaml` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s  $\text{\TeX}$ , 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) $\text{\TeX}$ , but Con $\text{\TeX}$ t MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in Con $\text{\TeX}$ t MkIV and therefore to insert hard line breaks into markdown text.

```

15039 \startluacode
15040 document.markdown_buffering = false
15041 local function preserve_trailing_spaces(line)
15042 if document.markdown_buffering then
15043 line = line:gsub("[\t][\t]$", "\t\t")
15044 end
15045 return line
15046 end
15047 resolvers.installinputlinehandler(preserve_trailing_spaces)
15048 \stopluacode
15049 \begingroup
15050 \catcode`\|=0%
15051 \catcode`\|=12%
15052 |gdef|startmarkdown{%
15053 |ctxlua{document.markdown_buffering = true}%
15054 |markdownReadAndConvert{\stopmarkdown}%
15055 {|stopmarkdown}}%
15056 |gdef|stopmarkdown{%
15057 |ctxlua{document.markdown_buffering = false}%
15058 |markdownEnd}%
15059 |gdef|startyaml{%
15060 |begingroup
15061 |ctxlua{document.markdown_buffering = true}%
15062 |setupyaml[jekyllData, expectJekyllData, ensureJekyllData]%
15063 |markdownReadAndConvert{\stopyaml}%
15064 {|stopyaml}}%
15065 |gdef|stopyaml{%
15066 |ctxlua{document.markdown_buffering = false}%
15067 |yamlEnd}%
15068 |endgroup

```

### 3.4.2 Themes

This section overrides the plain  $\text{\TeX}$  implementation of the theme-loading mechanism

from Section 3.2.2. Furthermore, this section also implements the built-in ConT<sub>E</sub>Xt themes provided with the Markdown package.

```

15069 \ExplSyntaxOn
15070 \prop_new:N \g_@@_context_loaded_themes_linenos_prop
15071 \prop_new:N \g_@@_context_loaded_themes_versions_prop
15072 \cs_gset:Nn
15073 \@@_load_theme:nnn
15074 {

```

Determine whether a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain T<sub>E</sub>X theme instead.

```

15075 \file_if_exist:nTF
15076 { t - markdown theme #3.tex }
15077 {
15078 \prop_get:NnNTF
15079 \g_@@_context_loaded_themes_linenos_prop
15080 { #1 }
15081 \l_tmpa_tl
15082 {
15083 \prop_get:NnN
15084 \g_@@_context_loaded_themes_versions_prop
15085 { #1 }
15086 \l_tmpb_tl
15087 \str_if_eq:nVTF
15088 { #2 }
15089 \l_tmpb_tl
15090 {
15091 \msg_warning:nnnVn
15092 { markdown }
15093 { repeatedly-loaded-context-theme }
15094 { #1 }
15095 \l_tmpa_tl
15096 { #2 }
15097 }
15098 {
15099 \msg_error:nnnnVV
15100 { markdown }
15101 { different-versions-of-context-theme }
15102 { #1 }
15103 { #2 }
15104 \l_tmpb_tl
15105 \l_tmpa_tl
15106 }
15107 }
15108 {
15109 \msg_info:nnn
15110 { markdown }

```

```

15111 { loading-context-theme }
15112 { #1 }
15113 { #2 }
15114 \prop_gput:Nnx
15115 \g_@@_context_loaded_themes_linenos_prop
15116 { #1 }
15117 { \tex_the:D \tex_inputlineno:D }
15118 \prop_gput:Nnn
15119 \g_@@_context_loaded_themes_versions_prop
15120 { #1 }
15121 { #2 }
15122 \usemodule
15123 [t]
15124 [markdown theme #3]
15125 }
15126 }
15127 {
15128 \@@_plain_tex_load_theme:nnn
15129 { #1 }
15130 { #2 }
15131 { #3 }
15132 }
15133 }
15134 \msg_new:nnn
15135 { markdown }
15136 { loading-context-theme }
15137 { Loading~version~#2~of~ConTeXt~Markdown~theme~#1 }
15138 \msg_new:nnn
15139 { markdown }
15140 { repeatedly-loaded-context-theme }
15141 {
15142 Version~#3~of~ConTeXt~Markdown~theme~#1~was~previously~
15143 loaded~on~line~#2,~not~loading~it~again
15144 }
15145 \msg_new:nnn
15146 { markdown }
15147 { different-versions-of-context-theme }
15148 {
15149 Tried~to~load~version~#2~of~ConTeXt~Markdown~theme~#1~
15150 but~version~#3~has~already~been~loaded~on~line~#4
15151 }
15152 \ExplSyntaxOff

```

The [witiko/markdown/defaults](#) ConT<sub>E</sub>Xt theme provides default definitions for token renderer prototypes. First, the ConT<sub>E</sub>Xt theme loads the plain T<sub>E</sub>X theme with the default definitions for plain T<sub>E</sub>X:

```

15153 \markdownLoadPlainTeXTheme

```

Next, the ConTeXt theme overrides some of the plain TeX definitions. See Section 3.4.3 for the actual definitions.

### 3.4.3 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

15154 \markdownIfOption{plain}{\iffalse}{\iftrue}
15155 \def\markdownRendererHardLineBreakPrototype{\blank}%
15156 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
15157 \def\markdownRendererRightBracePrototype{\textbraceright}%
15158 \def\markdownRendererDollarSignPrototype{\textdollar}%
15159 \def\markdownRendererPercentSignPrototype{\percent}%
15160 \def\markdownRendererUnderscorePrototype{\textunderscore}%
15161 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
15162 \def\markdownRendererBackslashPrototype{\textbackslash}%
15163 \def\markdownRendererTildePrototype{\textasciitilde}%
15164 \def\markdownRendererPipePrototype{\char`|}%
15165 \def\markdownRendererLinkPrototype#1#2#3#4{%
15166 \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
15167 \fi\texttt<\hyphenatedurl{#3}>}}%
15168 \usemodule[database]
15169 \defineseparatedlist
15170 [MarkdownConTeXtCSV]
15171 [separator={,},
15172 before=\bTABLE,after=\eTABLE,
15173 first=\bTR,last=\eTR,
15174 left=\bTD,right=\eTD]
15175 \def\markdownConTeXtCSV{csv}
15176 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
15177 \def\markdownConTeXtCSV@arg{#1}%
15178 \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
15179 \placetable[] [tab:#1]{#4}{%
15180 \processseparatedfile[MarkdownConTeXtCSV][#3]}%
15181 \else
15182 \markdownInput{#3}%
15183 \fi}%
15184 \def\markdownRendererImagePrototype#1#2#3#4{%
15185 \placefigure[] []{#4}{\externalfigure[#3]}}%
15186 \def\markdownRendererUlBeginPrototype{\startitemize}%
15187 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
15188 \def\markdownRendererUlItemPrototype{\item}%
15189 \def\markdownRendererUlEndPrototype{\stopitemize}%
15190 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
15191 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
15192 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%

```

```

15193 \def\markdownRendererOlItemPrototype{\item}%
15194 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
15195 \def\markdownRendererOlEndPrototype{\stopitemize}%
15196 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
15197 \definedescription
15198 [MarkdownConTeXtDlItemPrototype]
15199 [location=hanging,
15200 margin=standard,
15201 headstyle=bold]%
15202 \definestartstop
15203 [MarkdownConTeXtDlPrototype]
15204 [before=\blank,
15205 after=\blank]%
15206 \definestartstop
15207 [MarkdownConTeXtDlTightPrototype]
15208 [before=\blank\startpacked,
15209 after=\stoppacked\blank]%
15210 \def\markdownRendererDlBeginPrototype{%
15211 \startMarkdownConTeXtDlPrototype}%
15212 \def\markdownRendererDlBeginTightPrototype{%
15213 \startMarkdownConTeXtDlTightPrototype}%
15214 \def\markdownRendererDlItemPrototype#1{%
15215 \startMarkdownConTeXtDlItemPrototype{#1}}%
15216 \def\markdownRendererDlItemEndPrototype{%
15217 \stopMarkdownConTeXtDlItemPrototype}%
15218 \def\markdownRendererDlEndPrototype{%
15219 \stopMarkdownConTeXtDlPrototype}%
15220 \def\markdownRendererDlEndTightPrototype{%
15221 \stopMarkdownConTeXtDlTightPrototype}%
15222 \def\markdownRendererEmphasisPrototype#1{\em#1}%
15223 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
15224 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
15225 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
15226 \def\markdownRendererLineBlockBeginPrototype{%
15227 \begingroup
15228 \def\markdownRendererHardLineBreak{
15229 }%
15230 \startlines
15231 }%
15232 \def\markdownRendererLineBlockEndPrototype{%
15233 \stoplines
15234 \endgroup
15235 }%
15236 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

### 3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

15237 \ExplSyntaxOn
15238 \cs_gset:Npn
15239 \markdownRendererInputFencedCodePrototype#1#2#3
15240 {
15241 \tl_if_empty:nTF
15242 { #2 }
15243 { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConT<sub>E</sub>Xt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
 \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
 \stopmarkdown
\stoptext

```

```

15244 {
15245 \regex_extract_once:nnN
15246 { \w* }
15247 { #2 }
15248 \l_tmpa_seq
15249 \seq_pop_left:NN
15250 \l_tmpa_seq
15251 \l_tmpa_tl
15252 \typefile[\l_tmpa_tl] []{#1}
15253 }
15254 }
15255 \ExplSyntaxOff
15256 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
15257 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
15258 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
15259 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
15260 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%

```

```

15261 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
15262 \def\markdownRendererThematicBreakPrototype{%
15263 \blackrule[height=1pt, width=\hsize]}%
15264 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
15265 \def\markdownRendererTickedBoxPrototype{${\boxtimes}$}
15266 \def\markdownRendererHalfTickedBoxPrototype{${\boxdot}$}
15267 \def\markdownRendererUntickedBoxPrototype{${\square}$}
15268 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
15269 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
15270 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
15271 \def\markdownRendererDisplayMathPrototype#1{%
15272 \startformula#1\stopformula}%

```

### 3.4.3.2 Tables

There is a basic implementation of tables.

```

15273 \newcount\markdownConTeXtRowCounter
15274 \newcount\markdownConTeXtRowTotal
15275 \newcount\markdownConTeXtColumnCounter
15276 \newcount\markdownConTeXtColumnTotal
15277 \newtoks\markdownConTeXtTable
15278 \newtoks\markdownConTeXtTableFloat
15279 \def\markdownRendererTablePrototype#1#2#3{%
15280 \markdownConTeXtTable={}%
15281 \ifx\empty#1\empty
15282 \markdownConTeXtTableFloat={%
15283 \the\markdownConTeXtTable}%
15284 \else
15285 \markdownConTeXtTableFloat={%
15286 \placetable{#1}{\the\markdownConTeXtTable}}%
15287 \fi
15288 \begingroup
15289 \setupTABLE[r][each][topframe=off, bottomframe=off,
15290 leftframe=off, rightframe=off]
15291 \setupTABLE[c][each][topframe=off, bottomframe=off,
15292 leftframe=off, rightframe=off]
15293 \setupTABLE[r][1][topframe=on, bottomframe=on]
15294 \setupTABLE[r][#1][bottomframe=on]
15295 \markdownConTeXtRowCounter=0%
15296 \markdownConTeXtRowTotal=#2%
15297 \markdownConTeXtColumnTotal=#3%
15298 \markdownConTeXtRenderTableRow}
15299 \def\markdownConTeXtRenderTableRow#1{%
15300 \markdownConTeXtColumnCounter=0%
15301 \ifnum\markdownConTeXtRowCounter=0\relax
15302 \markdownConTeXtReadAlignments#1%
15303 \markdownConTeXtTable={\bTABLE}%

```

```

15304 \else
15305 \markdownConTeXtTable=\expandafter{%
15306 \the\markdownConTeXtTable\bTR}%
15307 \markdownConTeXtRenderTableCell#1%
15308 \markdownConTeXtTable=\expandafter{%
15309 \the\markdownConTeXtTable\eTR}%
15310 \fi
15311 \advance\markdownConTeXtRowCounter by 1\relax
15312 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
15313 \markdownConTeXtTable=\expandafter{%
15314 \the\markdownConTeXtTable\eTABLE}%
15315 \the\markdownConTeXtTableFloat
15316 \endgroup
15317 \expandafter\gobbleoneargument
15318 \fi\markdownConTeXtRenderTableRow}
15319 \def\markdownConTeXtReadAlignments#1{%
15320 \advance\markdownConTeXtColumnCounter by 1\relax
15321 \if#1d%
15322 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
15323 \fi\if#1l%
15324 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
15325 \fi\if#1c%
15326 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
15327 \fi\if#1r%
15328 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
15329 \fi
15330 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
15331 \else
15332 \expandafter\gobbleoneargument
15333 \fi\markdownConTeXtReadAlignments}
15334 \def\markdownConTeXtRenderTableCell#1{%
15335 \advance\markdownConTeXtColumnCounter by 1\relax
15336 \markdownConTeXtTable=\expandafter{%
15337 \the\markdownConTeXtTable\bTD#1\eTD}%
15338 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax
15339 \else
15340 \expandafter\gobbleoneargument
15341 \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`.

```

15342 \ExplSyntaxOn
15343 \cs_gset:Npn
15344 \markdownRendererInputRawInlinePrototype#1#2
15345 {

```



```

15346 \str_case:nnF
15347 { #2 }
15348 {
15349 { latex }
15350 {
15351 \@@_plain_tex_default_input_raw_inline:nn
15352 { #1 }
15353 { context }
15354 }
15355 }
15356 {
15357 \@@_plain_tex_default_input_raw_inline:nn
15358 { #1 }
15359 { #2 }
15360 }
15361 }
15362 \cs_gset:Npn
15363 \markdownRendererInputRawBlockPrototype#1#2
15364 {
15365 \str_case:nnF
15366 { #2 }
15367 {
15368 { context }
15369 {
15370 \@@_plain_tex_default_input_raw_block:nn
15371 { #1 }
15372 { tex }
15373 }
15374 }
15375 {
15376 \@@_plain_tex_default_input_raw_block:nn
15377 { #1 }
15378 { #2 }
15379 }
15380 }
15381 \cs_gset_eq:NN
15382 \markdownRendererInputRawBlockPrototype
15383 \markdownRendererInputRawInlinePrototype
15384 \fi % Closes \markdownIfOption{plain}{\iffalse}{\iftrue}
15385 \ExplSyntaxOff
15386 \stopmodule
15387 \protect

```

At the end of the ConT<sub>E</sub>Xt module, we load the [witiko/markdown/defaults](#) ConT<sub>E</sub>Xt theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

15388 \ExplSyntaxOn

```

```

15389 \str_if_eq:VVT
15390 \c_@@_top_layer_tl
15391 \c_@@_option_layer_context_tl
15392 {
15393 \ExplSyntaxOff
15394 \@@_if_option:nF
15395 { noDefaults }
15396 {
15397 \@@_if_option:nTF
15398 { experimental }
15399 {
15400 \@@_setup:n
15401 { theme = witiko/markdown/defaults@experimental }
15402 }
15403 {
15404 \@@_setup:n
15405 { theme = witiko/markdown/defaults }
15406 }
15407 }
15408 \ExplSyntaxOn
15409 }
15410 \ExplSyntaxOff
15411 \stopmodule
15412 \protect

```

## References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Frank Mittelbach, Ulrike Fischer, and L<sup>A</sup>T<sub>E</sub>X Project. *The documentmetadata-support code*. June 1, 2024. URL: <https://mirrors.ctan.org/macros/latex/required/latex-lab/documentmetadata-support-code.pdf> (visited on 10/21/2024).
- [3] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [4] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [5] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).

- [6] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under\_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [7] Donald Ervin Knuth. *The T<sub>E</sub>Xbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [8] Frank Mittelbach. *The doc and 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<sup>A</sup>T<sub>E</sub>X’s hook management*. June 26, 2024. URL: <https://mirrors.ctan.org/macros/latex/base/lthooks-code.pdf> (visited on 10/02/2024).
- [13] Geoffrey M. Poore. *The minted Package. Highlighted source code in L<sup>A</sup>T<sub>E</sub>X*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [14] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [15] Johannes Braams et al. *The L<sup>A</sup>T<sub>E</sub>X<sub>2<sub>ε</sub></sub> Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [16] Donald Ervin Knuth. *T<sub>E</sub>X: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [17] Victor Eijkhout. *T<sub>E</sub>X by Topic. A T<sub>E</sub>Xnician’s Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

## Index

<code>autoIdentifiers</code>	<a href="#">21</a> , <a href="#">33</a> , <a href="#">80</a> , <a href="#">93</a>
<code>blankBeforeBlockquote</code>	<a href="#">21</a>
<code>blankBeforeCodeFence</code>	<a href="#">22</a>

blankBeforeDivFence	22
blankBeforeHeading	22
blankBeforeList	23
bracketedSpans	23, 82
breakableBlockquotes	23
cacheDir	4, 17, 20, 59, 146, 161, 362, 385, 403
citationNbsps	24
citations	24, 84, 85
codeSpans	25
contentBlocks	20, 25, 35
contentBlocksLanguageMap	20
contentLevel	26
debugExtensions	10, 20, 26, 306
debugExtensionsFileName	20, 26
defaultOptions	10, 51, 360, 362
definitionLists	26, 88
eagerCache	17, 360
ensureJekyllData	27
entities.char_entity	207
entities.dec_entity	207
entities.hex_entity	207
entities.hex_entity_with_x_char	207
escape_minimal	211
escape_programmatic_text	211
escape_typographic_text	211
expandtabs	266
expectJekyllData	27, 27
experimental	5, 18, 404
extensions	29, 156, 311
extensions.bracketed_spans	312
extensions.citations	313
extensions.content_blocks	317
extensions.definition_lists	320
extensions.fancy_lists	322
extensions.fenced_code	328
extensions.fenced_divs	333
extensions.header_attributes	338
extensions.inline_code_attributes	339
extensions.jekyll_data	356
extensions.line_blocks	340

<code>extensions.link_attributes</code>	342
<code>extensions.mark</code>	341
<code>extensions.notes</code>	343
<code>extensions.pipe_table</code>	345
<code>extensions.raw_inline</code>	350
<code>extensions.strike_through</code>	351
<code>extensions.subscripts</code>	351
<code>extensions.superscripts</code>	352
<code>extensions.tex_math</code>	353
<code>fancyLists</code>	30, 103–107, 404
<code>fencedCode</code>	30, 40, 85, 92, 108, 401
<code>fencedCodeAttributes</code>	31, 80, 92
<code>fencedDiv</code>	93
<code>fencedDivs</code>	32, 42
<code>finalizeCache</code>	17, 21, 32, 32, 59, 145, 360, 361
<code>frozenCache</code>	21, 32, 59, 145, 148, 149, 401, 403
<code>frozenCacheCounter</code>	32, 361, 393
<code>frozenCacheFileName</code>	21, 32, 59, 361
<code>gfmAutoIdentifiers</code>	21, 33, 80, 93
<code>hashEnumerators</code>	33
<code>headerAttributes</code>	34, 42, 80, 93
<code>html</code>	34, 96, 417
<code>hybrid</code>	34, 35, 40, 46, 48, 62, 73, 109, 146, 212, 267, 393
<code>inlineCodeAttributes</code>	36, 80, 86
<code>inlineNotes</code>	36
<code>\input</code>	55, 56
<code>\inputmarkdown</code>	151, 152–154, 432
<code>inputTempFileName</code>	60, 62, 386–388, 390
<code>\inputyaml</code>	151, 153, 432
<code>iterlines</code>	266
<code>jeekyllData</code>	3, 27, 28, 37, 117–120, 122
<code>\l_file_search_path_seq</code>	392
<code>languages_json</code>	317, 317
<code>lineBlocks</code>	38, 98
<code>linkAttributes</code>	37, 80, 97, 100, 287, 429
<code>mark</code>	38, 101, 427
<code>\markdown</code>	141, 142, 396

markdown	140, 140, 141, 395
markdown*	140, 140, 145, 395
\markdown_jekyll_data_concatenate_address:NN	379
\markdown_jekyll_data_pop:	379
\markdown_jekyll_data_push:nN	379
\markdown_jekyll_data_push_address_segment:n	378
\markdown_jekyll_data_set_keyval:Nn	380
\markdown_jekyll_data_set_keyvals:nn	380
\markdown_jekyll_data_update_address_tls:	379
\markdownBegin	53, 53–56, 138, 140, 141, 151, 152
\markdownCleanup	385
\markdownConvert	385
\markdownEnd	53, 53–56, 138, 140–142, 151, 152
\markdownError	138, 138
\markdownEscape	53, 56, 394
\markdownIfOption	58
\markdownIfSnippetExists	74
\markdownInfo	138, 138
\markdownInput	53, 55, 56, 140, 142, 143, 145, 152, 391, 395
\markdownInputFilename	384
\markdownInputFileStream	386
\markdownInputPlainTeX	395
\markdownLoadPlainTeXTheme	146, 155, 372
\markdownLuaExecute	388, 391
\markdownLuaOptions	382, 385
\markdownMakeOther	138, 431, 432
\markdownOptionFinalizeCache	59
\markdownOptionFrozenCache	59
\markdownOptionHybrid	62
\markdownOptionInputTempFileName	59
\markdownOptionNoDefaults	61
\markdownOptionOutputDir	60, 60, 63
\markdownOptionPlain	60
\markdownOptionStripPercentSigns	61
\markdownOutputFileStream	386
\markdownPrepare	385
\markdownPrepareInputFilename	384
\markdownPrepareLuaOptions	382
\markdownReadAndConvert	138, 386, 395, 396, 433
\markdownReadAndConvertProcessLine	387, 388
\markdownReadAndConvertStripPercentSigns	387
\markdownReadAndConvertTab	386

<code>\markdownRendererAttributeClassName</code>	80
<code>\markdownRendererAttributeIdentifier</code>	80
<code>\markdownRendererAttributeKeyValue</code>	80
<code>\markdownRendererBlockQuoteBegin</code>	81
<code>\markdownRendererBlockQuoteEnd</code>	81
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	82
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	82
<code>\markdownRendererCite</code>	84, 85
<code>\markdownRendererCodeSpan</code>	86
<code>\markdownRendererCodeSpanAttributeContextBegin</code>	86
<code>\markdownRendererCodeSpanAttributeContextEnd</code>	86
<code>\markdownRendererContentBlock</code>	87, 87
<code>\markdownRendererContentBlockCode</code>	87
<code>\markdownRendererContentBlockOnlineImage</code>	87
<code>\markdownRendererDisplayMath</code>	114
<code>\markdownRendererDlBegin</code>	88
<code>\markdownRendererDlBeginTight</code>	89
<code>\markdownRendererDlDefinitionBegin</code>	89
<code>\markdownRendererDlDefinitionEnd</code>	90
<code>\markdownRendererDlEnd</code>	90
<code>\markdownRendererDlEndTight</code>	90
<code>\markdownRendererDlItem</code>	89
<code>\markdownRendererDlItemEnd</code>	89
<code>\markdownRendererDocumentBegin</code>	101
<code>\markdownRendererDocumentEnd</code>	101
<code>\markdownRendererEllipsis</code>	42, 91
<code>\markdownRendererEmphasis</code>	91, 125
<code>\markdownRendererError</code>	116
<code>\markdownRendererFancyOlBegin</code>	103, 104
<code>\markdownRendererFancyOlBeginTight</code>	104
<code>\markdownRendererFancyOlEnd</code>	107
<code>\markdownRendererFancyOlEndTight</code>	107
<code>\markdownRendererFancyOlItem</code>	105
<code>\markdownRendererFancyOlItemEnd</code>	105
<code>\markdownRendererFancyOlItemWithNumber</code>	106
<code>\markdownRendererFencedCodeAttributeContextBegin</code>	92
<code>\markdownRendererFencedCodeAttributeContextEnd</code>	92
<code>\markdownRendererFencedDivAttributeContextBegin</code>	93
<code>\markdownRendererFencedDivAttributeContextEnd</code>	93
<code>\markdownRendererHalfTickedBox</code>	115
<code>\markdownRendererHardLineBreak</code>	99
<code>\markdownRendererHeaderAttributeContextBegin</code>	93

<code>\markdownRendererHeaderAttributeContextEnd</code>	93
<code>\markdownRendererHeadingFive</code>	95
<code>\markdownRendererHeadingFour</code>	95
<code>\markdownRendererHeadingOne</code>	94
<code>\markdownRendererHeadingSix</code>	95
<code>\markdownRendererHeadingThree</code>	94
<code>\markdownRendererHeadingTwo</code>	94
<code>\markdownRendererImage</code>	97
<code>\markdownRendererImageAttributeContextBegin</code>	97
<code>\markdownRendererImageAttributeContextEnd</code>	97
<code>\markdownRendererInlineHtmlComment</code>	96
<code>\markdownRendererInlineHtmlTag</code>	96
<code>\markdownRendererInlineMath</code>	114
<code>\markdownRendererInputBlockHtmlElement</code>	96
<code>\markdownRendererInputFencedCode</code>	85
<code>\markdownRendererInputRawBlock</code>	108
<code>\markdownRendererInputRawInline</code>	107
<code>\markdownRendererInputVerbatim</code>	85
<code>\markdownRendererInterblockSeparator</code>	98
<code>\markdownRendererJekyllDataBegin</code>	117
<code>\markdownRendererJekyllDataBoolean</code>	119
<code>\markdownRendererJekyllDataEmpty</code>	122
<code>\markdownRendererJekyllDataEnd</code>	117
<code>\markdownRendererJekyllDataMappingBegin</code>	117
<code>\markdownRendererJekyllDataMappingEnd</code>	118
<code>\markdownRendererJekyllDataNumber</code>	119
<code>\markdownRendererJekyllDataProgrammaticString</code>	120, 120
<code>\markdownRendererJekyllDataSequenceBegin</code>	118
<code>\markdownRendererJekyllDataSequenceEnd</code>	119
<code>\markdownRendererJekyllDataString</code>	120, 124
<code>\markdownRendererJekyllDataStringPrototype</code>	134
<code>\markdownRendererJekyllDataTypographicString</code>	120, 120, 357
<code>\markdownRendererLineBlockBegin</code>	99
<code>\markdownRendererLineBlockEnd</code>	99
<code>\markdownRendererLink</code>	100, 125
<code>\markdownRendererLinkAttributeContextBegin</code>	100
<code>\markdownRendererLinkAttributeContextEnd</code>	100
<code>\markdownRendererMark</code>	101
<code>\markdownRendererNbsp</code>	102
<code>\markdownRendererNote</code>	102
<code>\markdownRendererOlBegin</code>	103
<code>\markdownRendererOlBeginTight</code>	103



<code>\markdownRendererOlEnd</code>	106
<code>\markdownRendererOlEndTight</code>	106
<code>\markdownRendererOlItem</code>	42, 104
<code>\markdownRendererOlItemEnd</code>	104
<code>\markdownRendererOlItemWithNumber</code>	42, 105
<code>\markdownRendererParagraphSeparator</code>	98
<code>\markdownRendererReplacementCharacter</code>	109
<code>\markdownRendererSectionBegin</code>	108
<code>\markdownRendererSectionEnd</code>	108
<code>\markdownRendererSoftLineBreak</code>	99
<code>\markdownRendererStrikeThrough</code>	112
<code>\markdownRendererStrongEmphasis</code>	92
<code>\markdownRendererSubscript</code>	112
<code>\markdownRendererSuperscript</code>	113
<code>\markdownRendererTable</code>	114
<code>\markdownRendererTableAttributeContextBegin</code>	113
<code>\markdownRendererTableAttributeContextEnd</code>	113
<code>\markdownRendererTextCite</code>	85
<code>\markdownRendererThematicBreak</code>	115
<code>\markdownRendererTickedBox</code>	115
<code>\markdownRendererUlBegin</code>	82
<code>\markdownRendererUlBeginTight</code>	83
<code>\markdownRendererUlEnd</code>	84
<code>\markdownRendererUlEndTight</code>	84
<code>\markdownRendererUlItem</code>	83
<code>\markdownRendererUlItemEnd</code>	83
<code>\markdownRendererUntickedBox</code>	115
<code>\markdownRendererWarning</code>	116
<code>\markdownSetup</code>	57, 58, 62, 144, 145, 154, 396, 404
<code>\markdownSetupSnippet</code>	73, 73
<code>\markdownThemeVersion</code>	69, 69
<code>\markdownWarning</code>	138, 138
<code>\markinline</code>	53, 55, 140, 142, 389, 394
<code>\markinlinePlainTeX</code>	394
<code>new</code>	8, 18, 360, 362
<code>notes</code>	38, 102
<code>parsers</code>	227, 265, 266
<code>parsers.punctuation</code>	228
<code>pipeTables</code>	7, 39, 45, 114
<code>preserveTabs</code>	40, 43, 266

rawAttribute	35, 40, 40, 108
reader	9, 30, 156, 227, 265, 311
reader->add_special_character	9, 10, 30, 305
reader->auto_link_email	295
reader->auto_link_url	295
reader->create_parser	266
reader->finalize_grammar	301, 367
reader->initialize_named_group	306
reader->insert_pattern	9, 9, 10, 30, 301, 302, 307, 308
reader->lookup_note_reference	280
reader->lookup_reference	280
reader->normalize_tag	266
reader->options	265
reader->parser_functions	266
reader->parser_functions.name	266
reader->parsers	265, 265, 266
reader->register_link	280
reader->update_rule	301, 304, 308
reader->writer	265
reader.new	265, 265, 367
relativeReferences	41
\setupmarkdown	154, 154
\setupyaml	154
shiftHeadings	7, 41
singletonCache	18
slice	7, 41, 208, 220, 221
smartEllipses	42, 91, 146
\startmarkdown	151, 151, 433
startNumber	42, 104–106
\startyaml	151, 152, 433
\stopmarkdown	151, 151, 433
\stopyaml	151, 152, 433
strikeThrough	43, 112, 428
stripIndent	43, 267
stripPercentSigns	386, 387
subscripts	44, 112
superscripts	44, 113
syntax	303, 307
tableAttributes	44, 113
tableCaptions	7, 44, 45, 113

taskLists	45, 115, 416
texComments	46, 267
texMathDollars	35, 46, 114
texMathDoubleBackslash	35, 47, 114
texMathSingleBackslash	35, 47, 114
tightLists	47, 83, 84, 89, 90, 103, 104, 106, 107, 404
underscores	48
unicodeNormalization	19, 19
unicodeNormalizationForm	19, 19
util.cache	156, 157
util.cache_verbatim	157
util.encode_json_string	157
util.err	156
util.escaper	159
util.expand_tabs_in_line	157
util.flatten	158
util.intersperse	159
util.map	159
util.pathname	160
util.rope_last	159
util.rope_to_string	158
util.salt	161
util.table_copy	157
util.walk	157, 158, 159
util.warning	161
walkable_syntax	9, 20, 26, 301, 302, 304–307
writer	156, 156, 208, 311
writer->active_attributes	219, 219–221
writer->attribute_type_levels	219
writer->attributes	217
writer->block_html_element	216
writer->blockquote	216
writer->bulletitem	214
writer->bulletlist	214
writer->citations	313
writer->code	212
writer->contentblock	318
writer->defer_call	227, 227
writer->definitionlist	320
writer->display_math	353

writer->div_begin	334
writer->div_end	334
writer->document	217
writer->ellipsis	210
writer->emphasis	216
writer->error	212
writer->escape	212
writer->escaped_chars	211, 211
writer->escaped_minimal_strings	210, 211
writer->escaped_strings	211
writer->escaped_uri_chars	210, 211
writer->fancyitem	323
writer->fancylist	322
writer->fencedCode	329
writer->flatten_inlines	208, 208
writer->get_state	226
writer->hard_line_break	210
writer->heading	224
writer->identifier	212
writer->image	213
writer->infostring	212
writer->inline_html_comment	215
writer->inline_html_tag	215
writer->inline_math	353
writer->interblocksep	209
writer->is_writing	208, 208
writer->jekyllData	357
writer->lineblock	340
writer->link	213
writer->mark	341
writer->math	212
writer->nbsp	209
writer->note	343
writer->options	208
writer->ordereditem	215
writer->orderedlist	214
writer->paragraph	209
writer->paragraphsep	209
writer->plain	209
writer->pop_attributes	220, 220, 221
writer->push_attributes	220, 220, 221
writer->rawBlock	329

<code>writer-&gt;rawInline</code>	350
<code>writer-&gt;set_state</code>	226
<code>writer-&gt;slice_begin</code>	208
<code>writer-&gt;slice_end</code>	208
<code>writer-&gt;soft_line_break</code>	210
<code>writer-&gt;space</code>	209
<code>writer-&gt;span</code>	312
<code>writer-&gt;strike_through</code>	351
<code>writer-&gt;string</code>	212
<code>writer-&gt;strong</code>	216
<code>writer-&gt;subscript</code>	351
<code>writer-&gt;superscript</code>	352
<code>writer-&gt;table</code>	347
<code>writer-&gt;thematic_break</code>	210
<code>writer-&gt;checkbox</code>	216
<code>writer-&gt;undosep</code>	210, 310
<code>writer-&gt;uri</code>	212
<code>writer-&gt;verbatim</code>	216
<code>writer-&gt;warning</code>	161, 212
<code>writer.new</code>	208, 208, 367
<code>\yaml</code>	142
<code>yaml</code>	140, 141, 142, 395
<code>\yamlBegin</code>	53, 54, 138, 141, 152
<code>\yamlEnd</code>	53, 54, 138, 141, 142, 152
<code>\yamlInput</code>	53, 56, 140, 143, 153, 395
<code>\yamlSetup</code>	58