

A Markdown Interpreter for T_EX

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

Version 3.6.2-0-g6c30af7e
2024-07-14

Contents

1	Introduction	1	3	Implementation	140
1.1	Requirements	2	3.1	Lua Implementation	140
1.2	Feedback	6	3.2	Plain T _E X Implementation	339
1.3	Acknowledgements	7	3.3	L ^A T _E X Implementation	361
2	Interfaces	7	3.4	ConT _E Xt Implementation	392
2.1	Lua Interface	7		References	400
2.2	Plain T _E X Interface	50		Index	401
2.3	L ^A T _E X Interface	129			
2.4	ConT _E Xt Interface	137			

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	46
3	A sequence diagram of typesetting a document using the Lua CLI	47
4	Various formats of mathematical formulae	135
5	The banner of the Markdown package	136

1 Introduction

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

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

¹See <https://ctan.org/pkg/markdown>.

²See <https://commonmark.org/>.

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

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

1.1 Requirements

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

1.1.1 Lua Requirements

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

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

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

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

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

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

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

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

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

```
15 (function()
```

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

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

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

lua-uni-algos A package that implements Unicode case-folding in _{TEX} Live \geq 2020.

```
23 hard lua-uni-algos
24 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.

```
25 # hard lua-tinyyaml # TODO: Uncomment after TeX Live 2022 has been deprecated.
```

1.1.2 Plain _{TEX} Requirements

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

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

```
26 hard l3kernel
27 \unprotect
28 \ifx\ExplSyntaxOn\undefined
29   \input expl3-generic
30 \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.

```
31 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^ATeX Requirements

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

```
32 \NeedsTeXFormat{LaTeX2e}
33 \RequirePackage{expl3}
```

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

The following packages are soft prerequisites. They are only used to provide default token renderer prototypes (see sections 2.2.6 and 3.3.4) or L^ATeX themes (see Section 2.3.3) and will not be loaded if the option `plain` has been enabled (see Section 2.2.2.3):

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

```
34 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.

35 `soft graphics`

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

36 `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 2.3.3).

37 `soft latex`

38 `soft epstopdf-pkg # required by `latex``

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

39 `soft fancyvrb`

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

40 `soft csvsimple`

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

42 `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.

43 `soft gobble`

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

44 `soft amsmath`

45 `soft amsfonts`

catchfile A package that catches the contents of a file and puts it in a macro. It is used in the `witiko/graphicx/http` L^AT_EX theme, see Section 2.3.3.

46 `soft catchfile`

grffile A package that extends the name processing of the graphics package to support a larger range of file names in $2006 \leq$ T_EX Live \leq 2019. Since T_EX Live \geq 2020, the functionality of the package has been integrated in the L^AT_EX 2 _{ϵ} kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` L^AT_EX themes, see Section 2.3.3.

47 `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 3.3.4.8, and also in the default renderer prototype for identifier attributes.

48 `soft etoolbox`

soulutf8 A package that is used in the default renderer prototype for strike-throughs and marked text.

49 `soft soul`

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.

50 `soft ltxcmds`

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

51 `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.

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

⁵See <https://tex.stackexchange.com>.

1.3 Acknowledgements

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

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

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

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

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

2 Interfaces

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

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

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

2.1 Lua Interface

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

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

```
52 local M = {metadata = metadata}
```

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

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

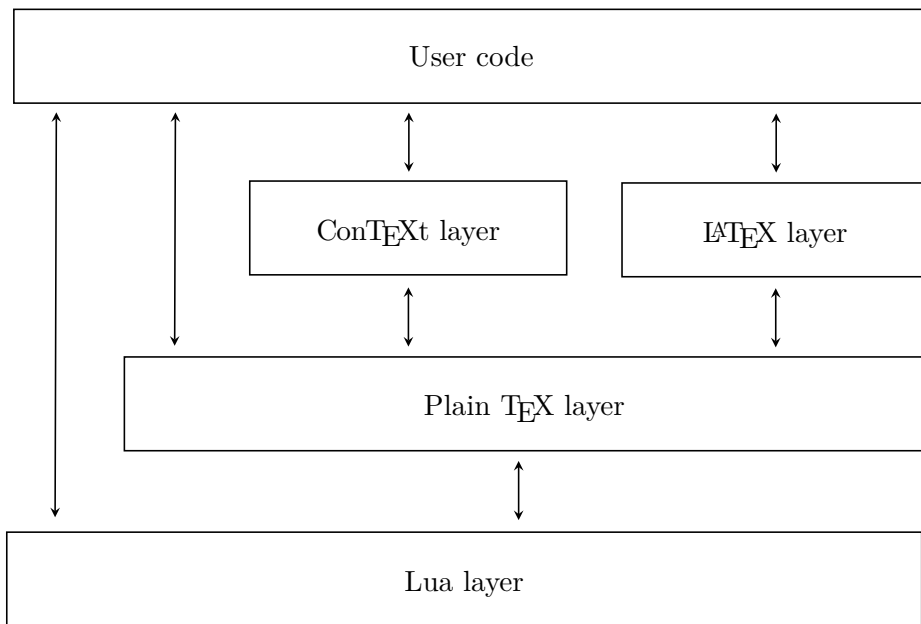


Figure 1: A block diagram of the Markdown package

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

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

```

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

```

2.1.2 User-Defined Syntax Extensions

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

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

```

53 local walkable_syntax = {

```



```

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

```

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.

```
85 local defaultOptions = {}
```

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

```
86 \ExplSyntaxOn
87 \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.

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

```

121 { Vnnn }
122 \tl_const:Nn \c_@@_option_value_true_tl { true }
123 \tl_const:Nn \c_@@_option_value_false_tl { false }
124 \cs_new:Nn \@@_typecheck_option:n
125 {
126   \@@_get_option_type:nN
127   { #1 }
128   \l_tmpa_tl
129   \str_case_e:Vn
130   \l_tmpa_tl
131   {
132     { \c_@@_option_type_boolean_tl }
133     {
134       \@@_get_option_value:nN
135       { #1 }
136       \l_tmpa_tl
137       \bool_if:nF
138       {
139         \str_if_eq_p:VV
140         \l_tmpa_tl
141         \c_@@_option_value_true_tl ||
142         \str_if_eq_p:VV
143         \l_tmpa_tl
144         \c_@@_option_value_false_tl
145       }
146       {
147         \msg_error:nnnV
148         { markdown }
149         { failed-typecheck-for-boolean-option }
150         { #1 }
151         \l_tmpa_tl
152       }
153     }
154   }
155 }
156 \msg_new:nnn
157 { markdown }
158 { failed-typecheck-for-boolean-option }
159 {
160   Option~#1~has~value~#2,~
161   but~a~boolean~(true~or~false)~was~expected.
162 }
163 \cs_generate_variant:Nn
164 \str_case_e:nn
165 { Vn }
166 \cs_generate_variant:Nn
167 \msg_error:nnnn

```

```

168 { nnnV }
169 \seq_new:N \g_@@_option_types_seq
170 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
171 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
172 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
173 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
174 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
175 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
176 \tl_const:Nn \c_@@_option_type_number_tl { number }
177 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
178 \tl_const:Nn \c_@@_option_type_path_tl { path }
179 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
180 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
181 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
182 \tl_const:Nn \c_@@_option_type_string_tl { string }
183 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
184 \cs_new:Nn
185 \@@_get_option_type:nN
186 {
187   \bool_set_false:N
188     \l_tmpa_bool
189   \seq_map_inline:Nn
190     \g_@@_option_layers_seq
191     {
192       \prop_get:cnNT
193         { g_@@_ ##1 _option_types_prop }
194         { #1 }
195       \l_tmpa_tl
196       {
197         \bool_set_true:N
198           \l_tmpa_bool
199         \seq_map_break:
200       }
201     }
202   \bool_if:nF
203     \l_tmpa_bool
204     {
205       \msg_error:nnn
206         { markdown }
207         { undefined-option }
208         { #1 }
209     }
210   \seq_if_in:NVF
211     \g_@@_option_types_seq
212     \l_tmpa_tl
213     {
214       \msg_error:nnnV

```

```

215         { markdown }
216         { unknown-option-type }
217         { #1 }
218         \l_tmpa_tl
219     }
220 \tl_set_eq:NN
221     #2
222     \l_tmpa_tl
223 }
224 \msg_new:nnn
225 { markdown }
226 { unknown-option-type }
227 {
228     Option~#1~has~unknown~type~#2.
229 }
230 \msg_new:nnn
231 { markdown }
232 { undefined-option }
233 {
234     Option~#1~is~undefined.
235 }
236 \cs_new:Nn
237 \@@_get_default_option_value:nN
238 {
239     \bool_set_false:N
240         \l_tmpa_bool
241     \seq_map_inline:Nn
242         \g_@@_option_layers_seq
243         {
244             \prop_get:cnNT
245                 { g_@@_default_ ##1 _options_prop }
246                 { #1 }
247                 #2
248                 {
249                     \bool_set_true:N
250                         \l_tmpa_bool
251                     \seq_map_break:
252                 }
253             }
254 \bool_if:nF
255     \l_tmpa_bool
256     {
257         \msg_error:nnn
258             { markdown }
259             { undefined-option }
260             { #1 }
261     }

```

```

262 }
263 \cs_new:Nn
264 \@@_get_option_value:nN
265 {
266   \@@_option_tl_to_csname:nN
267   { #1 }
268   \l_tmpa_tl
269   \cs_if_free:cTF
270   { \l_tmpa_tl }
271   {
272     \@@_get_default_option_value:nN
273     { #1 }
274     #2
275   }
276   {
277     \@@_get_option_type:nN
278     { #1 }
279     \l_tmpa_tl
280     \str_if_eq:NNTF
281     \c_@@_option_type_counter_tl
282     \l_tmpa_tl
283     {
284       \@@_option_tl_to_csname:nN
285       { #1 }
286       \l_tmpa_tl
287       \tl_set:Nx
288       #2
289       { \the \cs:w \l_tmpa_tl \cs_end: }
290     }
291     {
292       \@@_option_tl_to_csname:nN
293       { #1 }
294       \l_tmpa_tl
295       \tl_set:Nv
296       #2
297       { \l_tmpa_tl }
298     }
299   }
300 }
301 \cs_new:Nn \@@_option_tl_to_csname:nN
302 {
303   \tl_set:Nn
304   \l_tmpa_tl
305   { \str_uppercase:n { #1 } }
306   \tl_set:Nx
307   #2
308   {

```

```

309     markdownOption
310     \tl_head:f { \l_tmpa_tl }
311     \tl_tail:n { #1 }
312   }
313 }

```

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.

```

314 \cs_new:Nn \@@_with_various_cases:nn
315 {
316   \seq_clear:N
317   \l_tmpa_seq
318   \seq_map_inline:Nn
319   \g_@@_cases_seq
320   {
321     \tl_set:Nn
322     \l_tmpa_tl
323     { #1 }
324     \use:c { ##1 }
325     \l_tmpa_tl
326     \seq_put_right:NV
327     \l_tmpa_seq
328     \l_tmpa_tl
329   }
330   \seq_map_inline:Nn
331   \l_tmpa_seq
332   { #2 }
333 }

```

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

```

334 \cs_new:Nn \@@_with_various_cases_break:
335 {
336   \seq_map_break:
337 }

```

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

```

338 \seq_new:N \g_@@_cases_seq
339 \cs_new:Nn \@@_camel_case:N
340 {
341   \regex_replace_all:nnN
342   { _ ([a-z]) }
343   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
344   #1
345   \tl_set:Nx
346   #1

```

```

347     { #1 }
348   }
349 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
350 \cs_new:Nn \@@_snake_case:N
351   {
352     \regex_replace_all:nnN
353     { ([a-z])([A-Z]) }
354     { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
355     #1
356     \tl_set:Nx
357     #1
358     { #1 }
359   }
360 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }

```

2.1.4 General Behavior

`eagerCache=true, false`

default: `false`

- true** Converted markdown documents will be cached in `cacheDir`. This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. However, it also produces a large number of auxiliary files on the disk and obscures the output of the Lua command-line interface when it is used for plumbing. This behavior will always be used if the `finalizeCache` option is enabled.
- false** Converted markdown documents will not be cached. This decreases the number of auxiliary files that we produce and makes it easier to use the Lua command-line interface for plumbing. This behavior will only be used when the `finalizeCache` option is disabled.

```

361 \@@_add_lua_option:nnn
362   { eagerCache }
363   { boolean }
364   { false }
365 defaultOptions.eagerCache = false

```

`singletonCache=true, false`

default: `true`

- true** Conversion functions produced by the function `new(options)` will be cached in an LRU cache of size 1 keyed by `options`. This is more time-

and space-efficient than always producing a new conversion function but may expose bugs related to the idempotence of conversion functions.

This has been the default behavior since version 3.0.0 of the Markdown package.

false Every call to the function `new(options)` will produce a new conversion function that will not be cached. This is slower than caching conversion functions and may expose bugs related to memory leaks in the creation of conversion functions, see also issue #226⁶.

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

```
366 \@@_add_lua_option:nnn
367   { singletonCache }
368   { boolean }
369   { true }

370 defaultOptions.singletonCache = true

371 local singletonCache = {
372   convert = nil,
373   options = nil,
374 }
```

`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.

```
375 \@@_add_lua_option:nnn
376   { unicodeNormalization }
377   { boolean }
378   { true }

379 defaultOptions.unicodeNormalization = true
```

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

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

`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.

```
380 \@@_add_lua_option:nnn
381   { unicodeNormalizationForm }
382   { string }
383   { nfc }

384 defaultOptions.unicodeNormalizationForm = "nfc"
```

2.1.5 File and Directory Names

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

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

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

```
385 \@@_add_lua_option:nnn
386   { cacheDir }
387   { path }
388   { \markdownOptionOutputDir / _markdown_\jobname }

389 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.

```
390 \@@_add_lua_option:nnn
391   { contentBlocksLanguageMap }
392   { path }
393   { markdown-languages.json }
394 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.

```
395 \@@_add_lua_option:nnn
396   { debugExtensionsFileName }
397   { path }
398   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
399 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.

```
400 \@@_add_lua_option:nnn
401   { frozenCacheFileName }
402   { path }
403   { \markdownOptionCacheDir / frozenCache.tex }
404 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`.

```
405 \@@_add_lua_option:nnn
406   { autoIdentifiers }
407   { boolean }
408   { false }
409 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.

```
410 \@@_add_lua_option:nnn
411   { blankBeforeBlockquote }
412   { boolean }
413   { false }
414 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.

```
415 \@@_add_lua_option:nnn
416   { blankBeforeCodeFence }
417   { boolean }
418   { false }
419 defaultOptions.blankBeforeCodeFence = false
```

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

`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.

```
420 \@@_add_lua_option:nnn
421   { blankBeforeDivFence }
422   { boolean }
423   { false }

424 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.

```
425 \@@_add_lua_option:nnn
426   { blankBeforeHeading }
427   { boolean }
428   { false }

429 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.

```
430 \@@_add_lua_option:nnn
431   { blankBeforeList }
432   { boolean }
433   { false }

434 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.

```
435 \@@_add_lua_option:nnn
436 { bracketedSpans }
437 { boolean }
438 { false }

439 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.

```
440 \@@_add_lua_option:nnn
441 { breakableBlockquotes }
442 { boolean }
443 { true }

444 defaultOptions.breakableBlockquotes = true
```

`citationNbsps=true, false` default: false

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

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

```
445 \@@_add_lua_option:nnn
446 { citationNbsps }
447 { boolean }
448 { true }

449 defaultOptions.citationNbsps = true
```

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

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension¹⁰:

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

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

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

`false` Disable the Pandoc citation syntax extension.

```
450 \@@_add_lua_option:nnn
451   { citations }
452   { boolean }
453   { false }
454 defaultOptions.citations = false
```

`codeSpans=true, false`

default: true

`true` Enable the code span syntax:

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

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

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

```
455 \@@_add_lua_option:nnn
456   { codeSpans }
457   { boolean }
458   { true }
459 defaultOptions.codeSpans = true
```

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

`contentBlocks=true, false`

default: `false`

`true`

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

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

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

```
460 \@@_add_lua_option:nnn
461 { contentBlocks }
462 { boolean }
463 { false }

464 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
```

```
465 \@@_add_lua_option:nnn
466 { contentLevel }
467 { string }
468 { block }

469 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.

```
470 \@@_add_lua_option:nnn
471 { debugExtensions }
472 { boolean }
473 { false }

474 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.

```
475 \@@_add_lua_option:nnn
476 { definitionLists }
477 { boolean }
478 { false }

479 defaultOptions.definitionLists = false
```

`expectJekyllData=true, false`

default: `false`

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

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

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

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

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

```

\begin{markdown}
- this
- is
- YAML
\end{markdown}
\end{document}

```

```

480 \@@_add_lua_option:nnn
481 { expectJekyllData }
482 { boolean }
483 { false }

484 defaultOptions.expectJekyllData = false

```

`extensions=<filenames>`

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the kpathsea library is available, files will be searched for not only in the current working directory but also in the T<sub>E</sub>X 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 nonpacechar = lpeg.P(1) - lpeg.S("\t ")
 local doubleslashes = lpeg.P("//")
 local function between(p, starter, ender)
 ender = lpeg.B(nonpacechar) * ender
 return (starter * #nonpacechar
 * 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:

```
485 metadata.user_extension_api_version = 2
486 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.

```
487 \cs_generate_variant:Nn
488 \@@_add_lua_option:nnn
489 { nnV }
490 \@@_add_lua_option:nnV
491 { extensions }
492 { clist }
493 \c_empty_clist
494 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: `false`

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

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

`false` Disable the Pandoc fancy list syntax extension.

```
495 \@@_add_lua_option:nnn
496 { fancyLists }
497 { boolean }
498 { false }
499 defaultOptions.fancyLists = false
```

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

`fencedCode=true, false`

default: true

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

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

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

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

```
500 \@@_add_lua_option:nnn
501 { fencedCode }
502 { boolean }
503 { true }

504 defaultOptions.fencedCode = true
```

`fencedCodeAttributes=true, false`

default: false

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

```
~~~~ {#mycode .haskell .numberLines startFrom=100}
qsort [] = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
               qsort (filter (>= x) xs)
~~~~~
```

**false** Disable the Pandoc fenced code attribute syntax extension.

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

```

505 \@@_add_lua_option:nnn
506 { fencedCodeAttributes }
507 { boolean }
508 { false }

509 defaultOptions.fencedCodeAttributes = false

```

`fencedDivs=true, false`

default: false

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

```

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

And another.
:::::

```

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

```

510 \@@_add_lua_option:nnn
511 { fencedDivs }
512 { boolean }
513 { false }

514 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<sub>E</sub>X document that contains markdown documents without invoking Lua using the `frozenCache` plain T<sub>E</sub>X option. As a result, the plain T<sub>E</sub>X document becomes more portable, but further changes in the order and the content of markdown documents will not be reflected.

```

515 \@@_add_lua_option:nnn
516 { finalizeCache }
517 { boolean }
518 { false }

519 defaultOptions.finalizeCache = false

```

<sup>13</sup>See [https://pandoc.org/MANUAL.html#extension-fenced\\_divs](https://pandoc.org/MANUAL.html#extension-fenced_divs).

`frozenCacheCounter`= $\langle number \rangle$  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<sub>E</sub>X macro `\markdownFrozenCache` $\langle number \rangle$  that will typeset markdown document number  $\langle number \rangle$ .

```
520 \@@_add_lua_option:nmn
521 { frozenCacheCounter }
522 { counter }
523 { 0 }

524 defaultOptions.frozenCacheCounter = 0
```

`gfmAutoIdentifiers`=true, false default: false

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

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

```
123 Sesame Street
```

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

See also the option `autoIdentifiers`.

```
525 \@@_add_lua_option:nmn
526 { gfmAutoIdentifiers }
527 { boolean }
528 { false }

529 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.

---

<sup>14</sup>See [https://pandoc.org/MANUAL.html#extension-gfm\\_auto\\_identifiers](https://pandoc.org/MANUAL.html#extension-gfm_auto_identifiers).

```

530 \@@_add_lua_option:nnn
531 { hashEnumerators }
532 { boolean }
533 { false }

534 defaultOptions.hashEnumerators = false

```

`headerAttributes=true, false`

default: false

**true** Enable the assignment of HTML attributes to headings:

```

My first heading {#foo}

My second heading ## {#bar .baz}

Yet another heading {key=value}
=====

```

**false** Disable the assignment of HTML attributes to headings.

```

535 \@@_add_lua_option:nnn
536 { headerAttributes }
537 { boolean }
538 { false }

539 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.

```

540 \@@_add_lua_option:nnn
541 { html }
542 { boolean }
543 { true }

544 defaultOptions.html = true

```



`hybrid=true, false`

default: `false`

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

```
545 \@@_add_lua_option:nmn
546 { hybrid }
547 { boolean }
548 { false }
549 defaultOptions.hybrid = false
```

`inlineCodeAttributes=true, false`

default: `false`

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

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

- false** Enable the Pandoc inline code span attribute extension.

```
550 \@@_add_lua_option:nmn
551 { inlineCodeAttributes }
552 { boolean }
553 { false }
554 defaultOptions.inlineCodeAttributes = false
```

`inlineNotes=true, false`

default: `false`

- true** Enable the Pandoc inline note syntax extension<sup>16</sup>:

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

---

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

<sup>16</sup>See [https://pandoc.org/MANUAL.html#extension-inline\\_notes](https://pandoc.org/MANUAL.html#extension-inline_notes).

`false` Disable the Pandoc inline note syntax extension.

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

`jeekyllData=true, false` default: false

`true` Enable the Pandoc YAML metadata block syntax extension<sup>17</sup> for entering metadata in YAML:

```

title: 'This is the title: it contains a colon'
author:
- Author One
- Author Two
keywords: [nothing, nothingness]
abstract: |
 This is the abstract.

 It consists of two paragraphs.

```

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

```
560 \@@_add_lua_option:nnn
561 { jeekyllData }
562 { boolean }
563 { false }
564 defaultOptions.jekyllData = false
```

`linkAttributes=true, false` default: false

`true` Enable the Pandoc link and image attribute syntax extension<sup>18</sup>:

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

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

<sup>17</sup>See [https://pandoc.org/MANUAL.html#extension-yaml\\_metadata\\_block](https://pandoc.org/MANUAL.html#extension-yaml_metadata_block).

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

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

```
565 \@@_add_lua_option:nnn
566 { linkAttributes }
567 { boolean }
568 { false }

569 defaultOptions.linkAttributes = false
```

`lineBlocks=true, false` default: `false`

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

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

`false` Disable the Pandoc line block syntax extension.

```
570 \@@_add_lua_option:nnn
571 { lineBlocks }
572 { boolean }
573 { false }

574 defaultOptions.lineBlocks = false
```

`mark=true, false` default: `false`

`true` Enable the Pandoc mark syntax extension<sup>20</sup>:

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

`false` Disable the Pandoc mark syntax extension.

```
575 \@@_add_lua_option:nnn
576 { mark }
577 { boolean }
578 { false }

579 defaultOptions.mark = false
```

---

<sup>19</sup>See [https://pandoc.org/MANUAL.html#extension-line\\_blocks](https://pandoc.org/MANUAL.html#extension-line_blocks).

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

`notes=true, false`

default: false

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

```
Here is a note reference, [^1] and another. [^longnote]

[^1]: Here is the note.

[^longnote]: Here's one with multiple blocks.

 Subsequent paragraphs are indented to show that they
 belong to the previous note.

 { some.code }

 The whole paragraph can be indented, or just the
 first line. In this way, multi-paragraph notes
 work like multi-paragraph list items.

This paragraph won't be part of the note, because it
isn't indented.
```

`false` Disable the Pandoc note syntax extension.

```
580 \@@_add_lua_option:nnn
581 { notes }
582 { boolean }
583 { false }

584 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.

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

```

585 \@@_add_lua_option:nnn
586 { pipeTables }
587 { boolean }
588 { false }

589 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.

```

590 \@@_add_lua_option:nnn
591 { preserveTabs }
592 { boolean }
593 { true }

594 defaultOptions.preserveTabs = true

```

`rawAttribute=true, false`

default: false

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

```

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

```

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

```

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

```

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.

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

```

595 \@@_add_lua_option:nnn
596 { rawAttribute }
597 { boolean }
598 { false }

599 defaultOptions.rawAttribute = false

```

`relativeReferences=true, false`

default: false

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

I conclude in Section <#conclusion>.

**Conclusion {#conclusion}**

=====

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

`false` Disable relative references in autolinks.

```

600 \@@_add_lua_option:nnn
601 { relativeReferences }
602 { boolean }
603 { false }

604 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.

```

605 \@@_add_lua_option:nnn
606 { shiftHeadings }
607 { number }
608 { 0 }

609 defaultOptions.shiftHeadings = 0

```

---

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

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

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

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

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

```
610 \@@_add_lua_option:nnn
611 { slice }
612 { slice }
613 { ^~$ }
614 defaultOptions.slice = "^ $"
```

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

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

`false` Preserve all ellipses in the input.

```
615 \@@_add_lua_option:nnn
616 { smartEllipses }
617 { boolean }
618 { false }
619 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 `\markdownRendererOliItemWithNumber` T<sub>E</sub>X macro.
- false** Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOliItem` T<sub>E</sub>X macro.

```
620 \@@_add_lua_option:nnn
621 { startNumber }
622 { boolean }
623 { true }

624 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.

```
625 \@@_add_lua_option:nnn
626 { strikeThrough }
627 { boolean }
628 { false }

629 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}
```

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



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

```
630 \@@_add_lua_option:nnn
631 { stripIndent }
632 { boolean }
633 { false }

634 defaultOptions.stripIndent = false
```

`subscripts=true, false` default: false

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

H<sub>2</sub>O is a liquid.

`false` Disable the Pandoc subscript syntax extension.

```
635 \@@_add_lua_option:nnn
636 { subscripts }
637 { boolean }
638 { false }

639 defaultOptions.subscripts = false
```

`superscripts=true, false` default: false

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

2<sup>10</sup> is 1024.

`false` Disable the Pandoc superscript syntax extension.

```
640 \@@_add_lua_option:nnn
641 { superscripts }
642 { boolean }
643 { false }

644 defaultOptions.superscripts = false
```

---

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

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

`tableAttributes=true, false`

default: `false`

`true`

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

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

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

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

```
645 \@@_add_lua_option:nnm
646 { tableAttributes }
647 { boolean }
648 { false }

649 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.

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

```

650 \@@_add_lua_option:nnn
651   { tableCaptions }
652   { boolean }
653   { false }

654 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.

```

655 \@@_add_lua_option:nnn
656   { taskLists }
657   { boolean }
658   { false }

659 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.

```

660 \@@_add_lua_option:nnn
661   { texComments }
662   { boolean }
663   { false }

664 defaultOptions.texComments = false

```

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

`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.

```
665 \@@_add_lua_option:nmn  
666 { texMathDollars }  
667 { boolean }  
668 { false }  
  
669 defaultOptions.texMathDollars = false
```

`texMathDoubleBackslash=true, false`

default: `false`

`true` Enable the Pandoc double backslash math syntax extension³⁰:

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

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

```
670 \@@_add_lua_option:nmn  
671 { texMathDoubleBackslash }  
672 { boolean }  
673 { false }  
  
674 defaultOptions.texMathDoubleBackslash = false
```

`texMathSingleBackslash=true, false`

default: `false`

`true` Enable the Pandoc single backslash math syntax extension³¹:

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

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

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

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

³¹See https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash.

```

675 \@@_add_lua_option:nmn
676 { texMathSingleBackslash }
677 { boolean }
678 { false }

679 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:

```

- 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.

```

680 \@@_add_lua_option:nmn
681 { tightLists }
682 { boolean }
683 { true }

684 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.

```

685 \@_add_lua_option:nnn
686   { underscores }
687   { boolean }
688   { true }
689 \ExplSyntaxOff

690 defaultOptions.underscores = true

```

2.1.7 Command-Line Interface

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

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

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

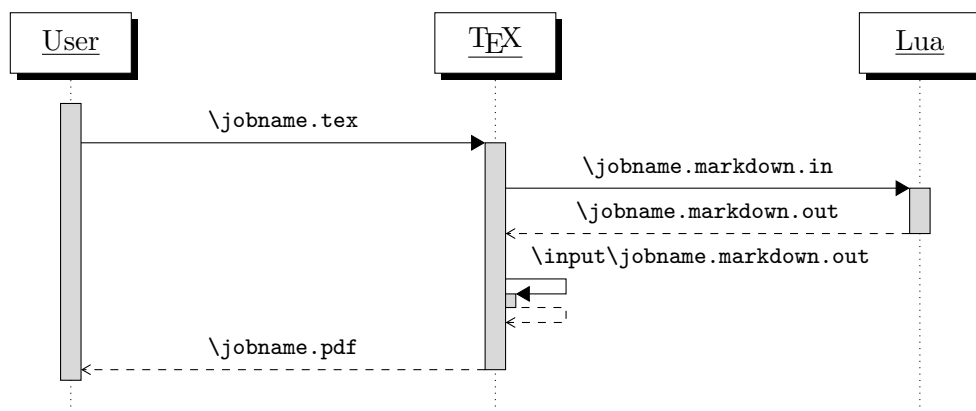


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the $\text{T}_{\text{E}}\text{X}$ interface

691

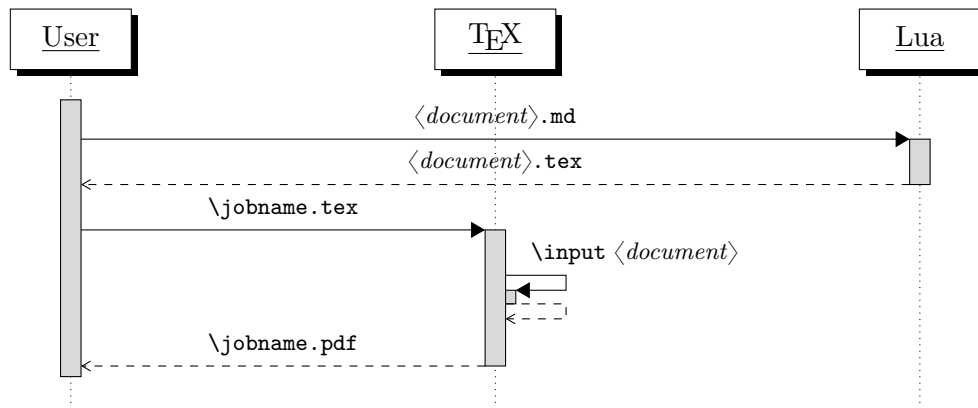


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

692 local HELP_STRING = [[
693 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
694 where OPTIONS are documented in the Lua interface section of the
695 technical Markdown package documentation.
696
697 When OUTPUT_FILE is unspecified, the result of the conversion will be
698 written to the standard output. When INPUT_FILE is also unspecified, the
699 result of the conversion will be read from the standard input.
700
701 Report bugs to: witiko@mail.muni.cz
702 Markdown package home page: <https://github.com/witiko/markdown>]]
703
704 local VERSION_STRING = [[
705 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
706
707 Copyright (C) ]] .. table.concat(metadata.copyright,
708                                     "\nCopyright (C) ") .. [[
709
710 License: ]] .. metadata.license
711
712 local function warn(s)
713   io.stderr:write("Warning: " .. s .. "\n") end
714
715 local function error(s)
716   io.stderr:write("Error: " .. s .. "\n")
717   os.exit(1)
718 end
  
```

To make it easier to copy-and-paste options from Pandoc [4] such as `fancy_lists`, `header_attributes`, and `pipe_tables`, we accept `snake_case` in addition to camel-

Case variants of options. As a bonus, studies [5] also show that `snake_case` is faster to read than `camelCase`.

```
719 local function camel_case(option_name)
720   local cased_option_name = option_name:gsub("_(%l)", function(match)
721     return match:sub(2, 2):upper()
722   end)
723   return cased_option_name
724 end
725
726 local function snake_case(option_name)
727   local cased_option_name = option_name:gsub("%l%u", function(match)
728     return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
729   end)
730   return cased_option_name
731 end
732
733 local cases = {camel_case, snake_case}
734 local various_case_options = {}
735 for option_name, _ in pairs(defaultOptions) do
736   for _, case in ipairs(cases) do
737     various_case_options[case(option_name)] = option_name
738   end
739 end
740
741 local process_options = true
742 local options = {}
743 local input_filename
744 local output_filename
745 for i = 1, #arg do
746   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.

```
747     if arg[i] == "--" then
748       process_options = false
749       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.

```
750     elseif arg[i]:match("=") then
751       local key, value = arg[i]:match("(.-)=(.*)")
752       if defaultOptions[key] == nil and
753         various_case_options[key] ~= nil then
754         key = various_case_options[key]
```



```
755     end
```

The `defaultOptions` table is consulted to identify whether $\langle value \rangle$ should be parsed as a string, number, table, or boolean.

```
756     local default_type = type(defaultOptions[key])
757     if default_type == "boolean" then
758         options[key] = (value == "true")
759     elseif default_type == "number" then
760         options[key] = tonumber(value)
761     elseif default_type == "table" then
762         options[key] = {}
763         for item in value:gmatch("[^,]+") do
764             table.insert(options[key], item)
765         end
766     else
767         if default_type ~= "string" then
768             if default_type == "nil" then
769                 warn('Option "' .. key .. '" not recognized.')
770             else
771                 warn('Option "' .. key .. '" type not recognized, please file ' ..
772                     'a report to the package maintainer.')
773             end
774             warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
775                 key .. '" as a string.')
776         end
777         options[key] = value
778     end
779     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.

```
780     elseif arg[i] == "--help" or arg[i] == "-h" then
781         print(HELP_STRING)
782         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.

```
783     elseif arg[i] == "--version" or arg[i] == "-v" then
784         print(VERSION_STRING)
785         os.exit()
786     end
787 end
```

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

```

788   if input_filename == nil then
789     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.

```

790   elseif output_filename == nil then
791     output_filename = arg[i]
792   else
793     error('Unexpected argument: "' .. arg[i] .. "'.')
794   end
795   ::continue::
796 end

```

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

```

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

```

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

```



```

2.2 Plain \TeX Interface

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

```

797 \def\markdownLastModified{((LASTMODIFIED))}%
798 \def\markdownVersion{((VERSION))}%

```

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

```



```

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

2.2.1 Typesetting Markdown

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

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

```
799 \let\markdownBegin\relax
800 \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 \TeX [6, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

```
801 \let\markinline\relax
```

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

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

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

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

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

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

```
802 \let\markdownInput\relax
```

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

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

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

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

```
803 \let\markdownEscape\relax
```

2.2.2 Options

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

To determine whether plain 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.

```

804 \ExplSyntaxOn
805 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
806 \cs_generate_variant:Nn
807   \tl_const:Nn
808   { NV }
809 \tl_if_exist:NF
810   \c_@@_top_layer_tl
811   {
812     \tl_const:NV
813       \c_@@_top_layer_tl
814       \c_@@_option_layer_plain_tex_tl
815   }

```

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

```
816 \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.

```

817 \prop_new:N \g_@@_plain_tex_option_types_prop
818 \prop_new:N \g_@@_default_plain_tex_options_prop
819 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
820 \cs_new:Nn
821   \@@_add_plain_tex_option:nnn
822   {
823     \@@_add_option:Vnnn
824     \c_@@_option_layer_plain_tex_tl
825     { #1 }
826     { #2 }
827     { #3 }
828   }

```

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 `<key>=<value>` pairs. For boolean options, the `=<value>` part is optional, and `<key>` will be interpreted as `<key>=true` if the `=<value>` part has been omitted. The `\markdownSetup` macro receives the options to set up as its only argument.

```

829 \cs_new:Nn
830   \@@_setup:n
831   {
832     \keys_set:nn
833       { markdown/options }
834       { #1 }

```

```

835 }
836 \cs_gset_eq:NN
837 \markdownSetup
838 \@@_setup:n

```

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

```

839 \prg_new_conditional:Nnn
840 \@@_if_option:n
841 { TF, T, F }
842 {
843   \@@_get_option_type:nN
844   { #1 }
845   \l_tmpa_tl
846   \str_if_eq:NNF
847   \l_tmpa_tl
848   \c_@@_option_type_boolean_tl
849   {
850     \msg_error:nxxx
851     { markdown }
852     { expected-boolean-option }
853     { #1 }
854     { \l_tmpa_tl }
855   }
856   \@@_get_option_value:nN
857   { #1 }
858   \l_tmpa_tl
859   \str_if_eq:NNTF
860   \l_tmpa_tl
861   \c_@@_option_value_true_tl
862   { \prg_return_true: }
863   { \prg_return_false: }
864 }
865 \msg_new:nnn
866 { markdown }
867 { expected-boolean-option }
868 {
869   Option~#1~has~type~#2,~
870   but~a~boolean~was~expected.
871 }
872 \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 \TeX document and their auxiliary files cached in the `cacheDir` directory.

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

```
873 \@@_add_plain_tex_option:nnn
874   { frozenCache }
875   { boolean }
876   { false }
```

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

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

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

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

```
877 \@@_add_plain_tex_option:nnn
878   { inputTempFileName }
879   { path }
880   { \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.

```
881 \@@_add_plain_tex_option:nnn
882   { outputDir }
```

```
883 { path }
884 { . }
```

2.2.2.3 No default token renderer prototypes

The Markdown package provides default definitions for token renderer prototypes using the `witiko/markdown/defaults` theme (see Section `sec:#themes`). Although these default definitions provide a useful starting point for authors, they use extra resources, especially with higher-level T_EX 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 T_EX formats should only use the plain T_EX default definitions or whether they should also use the format-specific default definitions. Whereas plain T_EX default definitions only provide definitions for simple elements such as emphasis, strong emphasis, and paragraph separators, format-specific default definitions add support for more complex elements such as lists, tables, and citations. On the flip side, plain T_EX 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.

```
885 \@_add_plain_tex_option:nnn
886 { plain }
887 { boolean }
888 { 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 L^AT_EX 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.

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

```
893 \seq_gput_right:Nn
894   \g_@@_plain_tex_options_seq
895   { stripPercentSigns }
896 \prop_gput:Nnn
897   \g_@@_plain_tex_option_types_prop
898   { stripPercentSigns }
899   { boolean }
900 \prop_gput:Nnx
901   \g_@@_default_plain_tex_options_prop
902   { stripPercentSigns }
903   { false }
```

2.2.2.5 Generating Plain TeX Option Macros and Key-Values

We define the command `\@@_define_option_commands_and_keyvals:` that defines plain TeX macros and the key-value interface of the `\markdownSetup` macro for the above plain TeX 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 TeX implementation, only passed along to Lua.

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

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.

```

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

```

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

```

915           \@@_with_various_cases:nn
916           { #####1 }
917           {
918             \@@_define_option_keyval:nnn
919             { ##1 }
920             { #####1 }
921             { #####1 }
922           }
923       }
924   }
925 }
926 \cs_new:Nn
927   \@@_define_option_command:n
928   {

```

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.

```

929   \str_if_eq:nnTF
930     { #1 }
931     { outputDir }
932     { \@@_define_option_command_output_dir: }
933     {

```

Do not override options defined before loading the package.

```

934     \@@_option_tl_to_csname:nN
935     { #1 }
936     \l_tmpa_tl

```

```

937     \cs_if_exist:cF
938     { \l_tmpa_tl }
939     {
940         \@@_get_default_option_value:nN
941         { #1 }
942         \l_tmpa_tl
943         \@@_set_option_value:nV
944         { #1 }
945         \l_tmpa_tl
946     }
947 }
948 }
949 \ExplSyntaxOff
950 \input lt3luabridge.tex
951 \ExplSyntaxOn
952 \cs_new:Nn
953   \@@_define_option_command_output_dir:
954   {
955     \cs_if_free:NT
956       \markdownOptionOutputDir
957     {
958       \bool_if:nTF
959         {
960           \cs_if_exist_p:N
961             \luabridge_tl_set:Nn &&
962           (
963             \int_compare_p:nNn
964               { \g_luabridge_method_int }
965               =
966               { \c_luabridge_method_directlua_int } ||
967             \sys_if_shell_unrestricted_p:
968           )
969         }
970         {
971           \luabridge_tl_set:Nn
972             \l_tmpa_tl
973             { print(os.getenv("TEXMF_OUTPUT_DIRECTORY") or ".") }
974           \tl_gset:NV
975             \markdownOptionOutputDir
976             \l_tmpa_tl
977         }
978         {
979           \tl_gset:Nn
980             \markdownOptionOutputDir
981             { . }
982         }
983     }

```

```

984 }
985 \cs_new:Nn
986 \@@_set_option_value:nn
987 {
988   \@@_define_option:n
989   { #1 }
990   \@@_get_option_type:nN
991   { #1 }
992   \l_tmpa_tl
993   \str_if_eq:NNTF
994   \c_@@_option_type_counter_tl
995   \l_tmpa_tl
996   {
997     \@@_option_tl_to_csname:nN
998     { #1 }
999     \l_tmpa_tl
1000     \int_gset:cn
1001     { \l_tmpa_tl }
1002     { #2 }
1003   }
1004   {
1005     \@@_option_tl_to_csname:nN
1006     { #1 }
1007     \l_tmpa_tl
1008     \cs_set:cpn
1009     { \l_tmpa_tl }
1010     { #2 }
1011   }
1012 }
1013 \cs_generate_variant:Nn
1014 \@@_set_option_value:nn
1015 { nV }
1016 \cs_new:Nn
1017 \@@_define_option:n
1018 {
1019   \@@_option_tl_to_csname:nN
1020   { #1 }
1021   \l_tmpa_tl
1022   \cs_if_free:cT
1023   { \l_tmpa_tl }
1024   {
1025     \@@_get_option_type:nN
1026     { #1 }
1027     \l_tmpb_tl
1028     \str_if_eq:NNT
1029     \c_@@_option_type_counter_tl
1030     \l_tmpb_tl

```

```

1031     {
1032         \@@_option_tl_to_csname:nN
1033         { #1 }
1034         \l_tmpa_tl
1035         \int_new:c
1036         { \l_tmpa_tl }
1037     }
1038 }
1039 }
1040 \cs_new:Nn
1041 \@@_define_option_keyval:nnn
1042 {
1043     \prop_get:cnN
1044     { g_@@_ #1 _option_types_prop }
1045     { #2 }
1046     \l_tmpa_tl
1047     \str_if_eq:VVTF
1048     \l_tmpa_tl
1049     \c_@@_option_type_boolean_tl
1050     {
1051         \keys_define:nn
1052         { markdown/options }
1053         {

```

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

```

1054         #3 .code:n = {
1055             \tl_set:Nx
1056             \l_tmpa_tl
1057             {
1058                 \str_case:nnF
1059                 { ##1 }
1060                 {
1061                     { yes } { true }
1062                     { no } { false }
1063                 }
1064                 { ##1 }
1065             }
1066             \@@_set_option_value:nV
1067             { #2 }
1068             \l_tmpa_tl
1069         },
1070         #3 .default:n = { true },
1071     }
1072 }
1073 {
1074     \keys_define:nn

```

```

1075         { markdown/options }
1076     {
1077         #3 .code:n = {
1078             \@@_set_option_value:nn
1079             { #2 }
1080             { ##1 }
1081         },
1082     }
1083 }

```

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.

```

1084     \str_if_eq:VVT
1085     \l_tmpa_tl
1086     \c_@@_option_type_clist_tl
1087     {
1088         \tl_set:Nn
1089         \l_tmpa_tl
1090         { #3 }
1091         \tl_reverse:N
1092         \l_tmpa_tl
1093         \str_if_eq:enF
1094         {
1095             \tl_head:V
1096             \l_tmpa_tl
1097         }
1098         { s }
1099         {
1100             \msg_error:nnn
1101             { markdown }
1102             { malformed-name-for-clist-option }
1103             { #3 }
1104         }
1105         \tl_set:Nx
1106         \l_tmpa_tl
1107         {
1108             \tl_tail:V
1109             \l_tmpa_tl
1110         }
1111         \tl_reverse:N
1112         \l_tmpa_tl
1113         \tl_put_right:Nn
1114         \l_tmpa_tl
1115         {

```

```

1116         .code:n = {
1117             \@@_get_option_value:nN
1118             { #2 }
1119             \l_tmpa_tl
1120             \clist_set:NV
1121             \l_tmpa_clist
1122             { \l_tmpa_tl, { ##1 } }
1123             \@@_set_option_value:nV
1124             { #2 }
1125             \l_tmpa_clist
1126         }
1127     }
1128     \keys_define:nV
1129     { markdown/options }
1130     \l_tmpa_tl
1131 }
1132 }
1133 \cs_generate_variant:Nn
1134 \clist_set:Nn
1135 { NV }
1136 \cs_generate_variant:Nn
1137 \keys_define:nn
1138 { nV }
1139 \cs_generate_variant:Nn
1140 \@@_set_option_value:nn
1141 { nV }
1142 \prg_generate_conditional_variant:Nnn
1143 \str_if_eq:nn
1144 { en }
1145 { F }
1146 \msg_new:nnn
1147 { markdown }
1148 { malformed-name-for-clist-option }
1149 {
1150     Clist~option~name~#1~does~not~end~with~-s.
1151 }

```

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

```

1152 \str_if_eq:VVT
1153 \c_@@_top_layer_tl
1154 \c_@@_option_layer_plain_tex_tl
1155 {
1156     \@@_define_option_commands_and_keyvals:
1157 }
1158 \ExplSyntaxOff

```

2.2.3 Themes

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

The key-values `theme=<theme name>` and `import=<theme name>` load a T_EX document (further referred to as a *theme*) named `markdowntheme<munged theme name>.tex`, where the *munged theme name* is the *theme name* after the substitution of all forward slashes (/) for an underscore (_). The theme name is *qualified* and contains no underscores. A theme name is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer L^AT_EX package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

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

Theme names are munged to allow structure inside theme names without dictating where the themes should be located inside the 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`.

```
1159 \ExplSyntaxOn
1160 \keys_define:nn
1161   { markdown/options }
1162   {
1163     theme .code:n = {
1164       @@_set_theme:n
1165       { #1 }
1166     },
1167     import .code:n = {
1168       \tl_set:Nn
1169       \l_tmpa_tl
1170       { #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.

```
1171     \tl_replace_all:NnV
1172     \l_tmpa_tl
1173     { / }
1174     \c_backslash_str
1175     \keys_set:nV
1176     { markdown/options/import }

```



```

1177     \l_tmpa_tl
1178   },
1179 }

```

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

```

1180 \seq_new:N
1181   \g_@@_themes_seq
1182 \tl_new:N
1183   \g_@@_current_theme_tl
1184 \tl_gset:Nn
1185   \g_@@_current_theme_tl
1186   { }
1187 \seq_gput_right:NV
1188   \g_@@_themes_seq
1189   \g_@@_current_theme_tl
1190 \cs_new:Nn
1191   \@@_set_theme:n
1192   {

```

First, we validate the theme name.

```

1193   \str_if_in:nnF
1194     { #1 }
1195     { / }
1196     {
1197       \msg_error:nnn
1198         { markdown }
1199         { unqualified-theme-name }
1200         { #1 }
1201     }
1202   \str_if_in:nnT
1203     { #1 }
1204     { _ }
1205     {
1206       \msg_error:nnn
1207         { markdown }
1208         { underscores-in-theme-name }
1209         { #1 }
1210     }

```

Next, we munge the theme name.

```

1211   \str_set:Nn
1212     \l_tmpa_str
1213     { #1 }
1214   \str_replace_all:Nnn
1215     \l_tmpa_str
1216     { / }

```

```
1217     { _ }
```

Finally, we load the theme.

```
1218     \tl_gset:Nn
1219     \g_@@_current_theme_tl
1220     { #1 / }
1221     \seq_gput_right:NV
1222     \g_@@_themes_seq
1223     \g_@@_current_theme_tl
1224     \@@_load_theme:nV
1225     { #1 }
1226     \l_tmpa_str
1227     \seq_gpop_right:NN
1228     \g_@@_themes_seq
1229     \l_tmpa_tl
1230     \seq_get_right:NN
1231     \g_@@_themes_seq
1232     \l_tmpa_tl
1233     \tl_gset:NV
1234     \g_@@_current_theme_tl
1235     \l_tmpa_tl
1236   }
1237 \msg_new:nnnn
1238   { markdown }
1239   { unqualified-theme-name }
1240   { Won't-load-theme-with-unqualified-name-#1 }
1241   { Theme-names-must-contain-at-least-one-forward-slash }
1242 \msg_new:nnnn
1243   { markdown }
1244   { underscores-in-theme-name }
1245   { Won't-load-theme-with-an-underscore-in-its-name-#1 }
1246   { Theme-names-must-not-contain-underscores-in-their-names }
1247 \cs_generate_variant:Nn
1248   \tl_replace_all:Nnn
1249   { NnV }
1250 \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.

```
1251 \ExplSyntaxOn
1252 \prop_new:N
1253   \g_@@_snippets_prop
1254 \cs_new:Nn
1255   \@@_setup_snippet:nm
1256   {
1257     \tl_if_empty:nT
1258       { #1 }
1259     {
1260       \msg_error:nnn
1261         { markdown }
1262         { empty-snippet-name }
1263         { #1 }
1264     }
1265     \tl_set:NV
1266       \l_tmpa_tl
1267       \g_@@_current_theme_tl
1268     \tl_put_right:Nn
1269       \l_tmpa_tl
1270       { #1 }
1271     \@@_if_snippet_exists:nT
1272       { #1 }
1273     {
1274       \msg_warning:nnV
1275         { markdown }
1276         { redefined-snippet }
1277         \l_tmpa_tl
1278     }
```

```

1279     \prop_gput:NVn
1280     \g_@@_snippets_prop
1281     \l_tmpa_tl
1282     { #2 }
1283   }
1284 \cs_gset_eq:NN
1285   \markdownSetupSnippet
1286   \@@_setup_snippet:nn
1287 \msg_new:nnnn
1288   { markdown }
1289   { empty-snippet-name }
1290   { Empty-snippet-name~#1 }
1291   { Pick-a-non-empty-name-for-your-snippet }
1292 \msg_new:nnn
1293   { markdown }
1294   { redefined-snippet }
1295   { Redefined~snippet~#1 }

```

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

```

1296 \prg_new_conditional:Nnn
1297   \@@_if_snippet_exists:n
1298   { TF, T, F }
1299   {
1300     \tl_set:NV
1301     \l_tmpa_tl
1302     \g_@@_current_theme_tl
1303     \tl_put_right:Nn
1304     \l_tmpa_tl
1305     { #1 }
1306     \prop_get:NVNTF
1307     \g_@@_snippets_prop
1308     \l_tmpa_tl
1309     \l_tmpb_tl
1310     { \prg_return_true: }
1311     { \prg_return_false: }
1312   }
1313 \cs_gset_eq:NN
1314   \markdownIfSnippetExists
1315   \@@_if_snippet_exists:nTF

```

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

```

1316 \keys_define:nn
1317   { markdown/options }
1318   {
1319     snippet .code:n = {
1320       \tl_set:NV
1321       \l_tmpa_tl

```

```

1322     \g_@@_current_theme_tl
1323     \tl_put_right:Nn
1324     \l_tmpa_tl
1325     { #1 }
1326     \@@_if_snippet_exists:nTF
1327     { #1 }
1328     {
1329         \prop_get:NVN
1330         \g_@@_snippets_prop
1331         \l_tmpa_tl
1332         \l_tmpb_tl
1333         \@@_setup:V
1334         \l_tmpb_tl
1335     }
1336     {
1337         \msg_error:nnV
1338         { markdown }
1339         { undefined-snippet }
1340         \l_tmpa_tl
1341     }
1342 }
1343 }
1344 \msg_new:nnn
1345 { markdown }
1346 { undefined-snippet }
1347 { Can't invoke undefined snippet ~#1 }
1348 \cs_generate_variant:Nn
1349 \@@_setup:n
1350 { V }
1351 \ExplSyntaxOff

```

Here is how we can use snippets to store options and invoke them later in L^AT_EX:

```

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

```

The following ordered list will be preceded by arabic numerals:

1. wahid
2. aithnayn

```

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

```

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

Furthermore, we can also specify the name of the snippet in the current namespace, which can be different from the name of the snippet in the `jdooe/lists` theme. For

example, we can make the snippet `jdoo/lists/romanNumerals` available under the name `roman`.

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

```
\end{markdown}
```

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

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

```
1352 \ExplSyntaxOn
1353 \tl_new:N
1354 \l_@@_import_current_theme_tl
1355 \keys_define:nn
1356 { markdown/options/import }
1357 {
```

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

```

1358     unknown .default:n = {},
1359     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.

```

1360     \tl_set_eq:NN
1361     \l_@@_import_current_theme_tl
1362     \l_keys_key_str
1363     \tl_replace_all:Nvn
1364     \l_@@_import_current_theme_tl
1365     \c_backslash_str
1366     { / }

```

Here, we import the snippets.

```

1367     \clist_map_inline:nn
1368     { #1 }
1369     {
1370         \regex_extract_once:nnNTF
1371         { ^(.*)\s+as\s+(.*)$ }
1372         { ##1 }
1373         \l_tmpa_seq
1374         {
1375             \seq_pop:NN
1376             \l_tmpa_seq
1377             \l_tmpa_tl
1378             \seq_pop:NN
1379             \l_tmpa_seq
1380             \l_tmpa_tl
1381             \seq_pop:NN
1382             \l_tmpa_seq
1383             \l_tmpb_tl
1384         }
1385         {
1386             \tl_set:Nn
1387             \l_tmpa_tl
1388             { ##1 }
1389             \tl_set:Nn
1390             \l_tmpb_tl
1391             { ##1 }
1392         }
1393     \tl_put_left:Nn
1394     \l_tmpa_tl
1395     { / }
1396     \tl_put_left:Nv

```



```

1397         \l_tmpa_tl
1398         \l_@@_import_current_theme_tl
1399         \@@_setup_snippet:Vx
1400         \l_tmpb_tl
1401         { snippet = { \l_tmpa_tl } }
1402     }

```

Here, we load the theme.

```

1403     \@@_set_theme:V
1404     \l_@@_import_current_theme_tl
1405 },
1406 }
1407 \cs_generate_variant:Nn
1408 \tl_replace_all:Nnn
1409 { NVn }
1410 \cs_generate_variant:Nn
1411 \@@_set_theme:n
1412 { V }
1413 \cs_generate_variant:Nn
1414 \@@_setup_snippet:nn
1415 { Vx }

```

2.2.5 Token Renderers

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

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

```

1416 \ExplSyntaxOn
1417 \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.

```

1418 \prop_new:N \g_@@_renderer_arities_prop
1419 \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.

```

1420 \def\markdownRendererAttributeIdentifier{%
1421   \markdownRendererAttributeIdentifierPrototype}%
1422 \ExplSyntaxOn
1423 \seq_gput_right:Nn
1424   \g_@@_renderers_seq
1425   { attributeIdentifier }
1426 \prop_gput:Nnn
1427   \g_@@_renderer_arities_prop
1428   { attributeIdentifier }
1429   { 1 }
1430 \ExplSyntaxOff
1431 \def\markdownRendererAttributeClassName{%
1432   \markdownRendererAttributeClassNamePrototype}%
1433 \ExplSyntaxOn
1434 \seq_gput_right:Nn
1435   \g_@@_renderers_seq
1436   { attributeClassName }
1437 \prop_gput:Nnn
1438   \g_@@_renderer_arities_prop
1439   { attributeClassName }
1440   { 1 }
1441 \ExplSyntaxOff
1442 \def\markdownRendererAttributeKeyValue{%
1443   \markdownRendererAttributeKeyValuePrototype}%
1444 \ExplSyntaxOn
1445 \seq_gput_right:Nn
1446   \g_@@_renderers_seq
1447   { attributeKeyValue }
1448 \prop_gput:Nnn
1449   \g_@@_renderer_arities_prop
1450   { attributeKeyValue }
1451   { 2 }
1452 \ExplSyntaxOff

```

2.2.5.2 Block Quote Renderers

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

```
1453 \def\markdownRendererBlockQuoteBegin{%
1454   \markdownRendererBlockQuoteBeginPrototype}%
1455 \ExplSyntaxOn
1456 \seq_gput_right:Nn
1457   \g_@@_renderers_seq
1458   { blockQuoteBegin }
1459 \prop_gput:Nnn
1460   \g_@@_renderer_arities_prop
1461   { blockQuoteBegin }
1462   { 0 }
1463 \ExplSyntaxOff
```

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

```
1464 \def\markdownRendererBlockQuoteEnd{%
1465   \markdownRendererBlockQuoteEndPrototype}%
1466 \ExplSyntaxOn
1467 \seq_gput_right:Nn
1468   \g_@@_renderers_seq
1469   { blockQuoteEnd }
1470 \prop_gput:Nnn
1471   \g_@@_renderer_arities_prop
1472   { blockQuoteEnd }
1473   { 0 }
1474 \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.

```
1475 \def\markdownRendererBracketedSpanAttributeContextBegin{%
1476   \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
1477 \ExplSyntaxOn
1478 \seq_gput_right:Nn
1479   \g_@@_renderers_seq
1480   { bracketedSpanAttributeContextBegin }
1481 \prop_gput:Nnn
1482   \g_@@_renderer_arities_prop
1483   { bracketedSpanAttributeContextBegin }
1484   { 0 }
```

```

1485 \ExplSyntaxOff
1486 \def\markdownRendererBracketedSpanAttributeContextEnd{%
1487   \markdownRendererBracketedSpanAttributeContextEndPrototype}%
1488 \ExplSyntaxOn
1489 \seq_gput_right:Nn
1490   \g_@@_renderers_seq
1491   { bracketedSpanAttributeContextEnd }
1492 \prop_gput:Nnn
1493   \g_@@_renderer_arities_prop
1494   { bracketedSpanAttributeContextEnd }
1495   { 0 }
1496 \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.

```

1497 \def\markdownRendererUlBegin{%
1498   \markdownRendererUlBeginPrototype}%
1499 \ExplSyntaxOn
1500 \seq_gput_right:Nn
1501   \g_@@_renderers_seq
1502   { ulBegin }
1503 \prop_gput:Nnn
1504   \g_@@_renderer_arities_prop
1505   { ulBegin }
1506   { 0 }
1507 \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.

```

1508 \def\markdownRendererUlBeginTight{%
1509   \markdownRendererUlBeginTightPrototype}%
1510 \ExplSyntaxOn
1511 \seq_gput_right:Nn
1512   \g_@@_renderers_seq
1513   { ulBeginTight }
1514 \prop_gput:Nnn
1515   \g_@@_renderer_arities_prop
1516   { ulBeginTight }
1517   { 0 }
1518 \ExplSyntaxOff

```

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

```
1519 \def\markdownRendererUListItem{%
1520   \markdownRendererUListItemPrototype}%
1521 \ExplSyntaxOn
1522 \seq_gput_right:Nn
1523   \g_@@_renderers_seq
1524   { ulItem }
1525 \prop_gput:Nnn
1526   \g_@@_renderer_arities_prop
1527   { ulItem }
1528   { 0 }
1529 \ExplSyntaxOff
```

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

```
1530 \def\markdownRendererUListItemEnd{%
1531   \markdownRendererUListItemEndPrototype}%
1532 \ExplSyntaxOn
1533 \seq_gput_right:Nn
1534   \g_@@_renderers_seq
1535   { ulItemEnd }
1536 \prop_gput:Nnn
1537   \g_@@_renderer_arities_prop
1538   { ulItemEnd }
1539   { 0 }
1540 \ExplSyntaxOff
```

The `\markdownRendererUListEnd` 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.

```
1541 \def\markdownRendererUListEnd{%
1542   \markdownRendererUListEndPrototype}%
1543 \ExplSyntaxOn
1544 \seq_gput_right:Nn
1545   \g_@@_renderers_seq
1546   { ulEnd }
1547 \prop_gput:Nnn
1548   \g_@@_renderer_arities_prop
1549   { ulEnd }
1550   { 0 }
1551 \ExplSyntaxOff
```

The `\markdownRendererUListEndTight` 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.

```

1552 \def\markdownRendererUleEndTight{%
1553   \markdownRendererUleEndTightPrototype}%
1554 \ExplSyntaxOn
1555 \seq_gput_right:Nn
1556   \g_@@_renderers_seq
1557   { ulEndTight }
1558 \prop_gput:Nnn
1559   \g_@@_renderers_arities_prop
1560   { ulEndTight }
1561   { 0 }
1562 \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.

```

1563 \def\markdownRendererCite{%
1564   \markdownRendererCitePrototype}%
1565 \ExplSyntaxOn
1566 \seq_gput_right:Nn
1567   \g_@@_renderers_seq
1568   { cite }
1569 \prop_gput:Nnn
1570   \g_@@_renderers_arities_prop
1571   { cite }
1572   { 1 }
1573 \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.

```

1574 \def\markdownRendererTextCite{%
1575   \markdownRendererTextCitePrototype}%
1576 \ExplSyntaxOn
1577 \seq_gput_right:Nn
1578   \g_@@_renderers_seq
1579   { textCite }
1580 \prop_gput:Nnn

```

```

1581 \g_@@_renderer_arities_prop
1582 { textCite }
1583 { 1 }
1584 \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.

```

1585 \def\markdownRendererInputVerbatim{%
1586 \markdownRendererInputVerbatimPrototype}%
1587 \ExplSyntaxOn
1588 \seq_gput_right:Nn
1589 \g_@@_renderers_seq
1590 { inputVerbatim }
1591 \prop_gput:Nnn
1592 \g_@@_renderer_arities_prop
1593 { inputVerbatim }
1594 { 1 }
1595 \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.

```

1596 \def\markdownRendererInputFencedCode{%
1597 \markdownRendererInputFencedCodePrototype}%
1598 \ExplSyntaxOn
1599 \seq_gput_right:Nn
1600 \g_@@_renderers_seq
1601 { inputFencedCode }
1602 \prop_gput:Nnn
1603 \g_@@_renderer_arities_prop
1604 { inputFencedCode }
1605 { 3 }
1606 \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.

```

1607 \def\markdownRendererCodeSpan{%
1608 \markdownRendererCodeSpanPrototype}%
1609 \ExplSyntaxOn

```

```

1610 \seq_gput_right:Nn
1611   \g_@@_renderers_seq
1612   { codeSpan }
1613 \prop_gput:Nnn
1614   \g_@@_renderer_arities_prop
1615   { codeSpan }
1616   { 1 }
1617 \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.

```

1618 \def\markdownRendererCodeSpanAttributeContextBegin{%
1619   \markdownRendererCodeSpanAttributeContextBeginPrototype}%
1620 \ExplSyntaxOn
1621 \seq_gput_right:Nn
1622   \g_@@_renderers_seq
1623   { codeSpanAttributeContextBegin }
1624 \prop_gput:Nnn
1625   \g_@@_renderer_arities_prop
1626   { codeSpanAttributeContextBegin }
1627   { 0 }
1628 \ExplSyntaxOff
1629 \def\markdownRendererCodeSpanAttributeContextEnd{%
1630   \markdownRendererCodeSpanAttributeContextEndPrototype}%
1631 \ExplSyntaxOn
1632 \seq_gput_right:Nn
1633   \g_@@_renderers_seq
1634   { codeSpanAttributeContextEnd }
1635 \prop_gput:Nnn
1636   \g_@@_renderer_arities_prop
1637   { codeSpanAttributeContextEnd }
1638   { 0 }
1639 \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.

```

1640 \def\markdownRendererContentBlock{%

```



```

1641 \markdownRendererContentBlockPrototype}%
1642 \ExplSyntaxOn
1643 \seq_gput_right:Nn
1644 \g_@@_renderers_seq
1645 { contentBlock }
1646 \prop_gput:Nnn
1647 \g_@@_renderer_arities_prop
1648 { contentBlock }
1649 { 4 }
1650 \ExplSyntaxOff

```

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

```

1651 \def\markdownRendererContentBlockOnlineImage{%
1652 \markdownRendererContentBlockOnlineImagePrototype}%
1653 \ExplSyntaxOn
1654 \seq_gput_right:Nn
1655 \g_@@_renderers_seq
1656 { contentBlockOnlineImage }
1657 \prop_gput:Nnn
1658 \g_@@_renderer_arities_prop
1659 { contentBlockOnlineImage }
1660 { 4 }
1661 \ExplSyntaxOff

```

The `\markdownRendererContentBlockCode` macro represents an iA Writer content block that was recognized as a file in a known programming language by its filename extension s . If any `markdown-languages.json` file found by `kpathsea`³² contains a record (k, v) , then a non-online-image content block with the filename extension $s, s:\text{lower}() = k$ is considered to be in a known programming language v . The macro receives five arguments: the local file name extension s cast to the lower case, the language v , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

Note that you will need to place a `markdown-languages.json` file inside your working directory or inside your local T_EX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```

1662 \def\markdownRendererContentBlockCode{%
1663 \markdownRendererContentBlockCodePrototype}%

```

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

```

1664 \ExplSyntaxOn
1665 \seq_gput_right:Nn
1666   \g_@@_renderers_seq
1667   { contentBlockCode }
1668 \prop_gput:Nnn
1669   \g_@@_renderer_arities_prop
1670   { contentBlockCode }
1671   { 5 }
1672 \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.

```

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

```

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

```

1684 \def\markdownRendererDlBeginTight{%
1685   \markdownRendererDlBeginTightPrototype}%
1686 \ExplSyntaxOn
1687 \seq_gput_right:Nn
1688   \g_@@_renderers_seq
1689   { dlBeginTight }
1690 \prop_gput:Nnn
1691   \g_@@_renderer_arities_prop
1692   { dlBeginTight }
1693   { 0 }
1694 \ExplSyntaxOff

```

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

```

1695 \def\markdownRendererDlItem{%
1696   \markdownRendererDlItemPrototype}%
1697 \ExplSyntaxOn
1698 \seq_gput_right:Nn
1699   \g_@@_renderers_seq
1700   { dlItem }
1701 \prop_gput:Nnn
1702   \g_@@_renderer_arities_prop
1703   { dlItem }
1704   { 1 }
1705 \ExplSyntaxOff

```

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

```

1706 \def\markdownRendererDlItemEnd{%
1707   \markdownRendererDlItemEndPrototype}%
1708 \ExplSyntaxOn
1709 \seq_gput_right:Nn
1710   \g_@@_renderers_seq
1711   { dlItemEnd }
1712 \prop_gput:Nnn
1713   \g_@@_renderer_arities_prop
1714   { dlItemEnd }
1715   { 0 }
1716 \ExplSyntaxOff

```

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

```

1717 \def\markdownRendererDlDefinitionBegin{%
1718   \markdownRendererDlDefinitionBeginPrototype}%
1719 \ExplSyntaxOn
1720 \seq_gput_right:Nn
1721   \g_@@_renderers_seq
1722   { dlDefinitionBegin }
1723 \prop_gput:Nnn
1724   \g_@@_renderer_arities_prop
1725   { dlDefinitionBegin }
1726   { 0 }
1727 \ExplSyntaxOff

```

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

```

1728 \def\markdownRendererDlDefinitionEnd{%
1729   \markdownRendererDlDefinitionEndPrototype}%

```

```

1730 \ExplSyntaxOn
1731 \seq_gput_right:Nn
1732   \g_@@_renderers_seq
1733   { dlDefinitionEnd }
1734 \prop_gput:Nnn
1735   \g_@@_renderer_arities_prop
1736   { dlDefinitionEnd }
1737   { 0 }
1738 \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.

```

1739 \def\markdownRendererDlEnd{%
1740   \markdownRendererDlEndPrototype}%
1741 \ExplSyntaxOn
1742 \seq_gput_right:Nn
1743   \g_@@_renderers_seq
1744   { dlEnd }
1745 \prop_gput:Nnn
1746   \g_@@_renderer_arities_prop
1747   { dlEnd }
1748   { 0 }
1749 \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.

```

1750 \def\markdownRendererDlEndTight{%
1751   \markdownRendererDlEndTightPrototype}%
1752 \ExplSyntaxOn
1753 \seq_gput_right:Nn
1754   \g_@@_renderers_seq
1755   { dlEndTight }
1756 \prop_gput:Nnn
1757   \g_@@_renderer_arities_prop
1758   { dlEndTight }
1759   { 0 }
1760 \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.

```

1761 \def\markdownRendererEllipsis{%
1762   \markdownRendererEllipsisPrototype}%
1763 \ExplSyntaxOn
1764 \seq_gput_right:Nn
1765   \g_@@_renderers_seq
1766   { ellipsis }
1767 \prop_gput:Nnn
1768   \g_@@_renderer_arities_prop
1769   { ellipsis }
1770   { 0 }
1771 \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.

```

1772 \def\markdownRendererEmphasis{%
1773   \markdownRendererEmphasisPrototype}%
1774 \ExplSyntaxOn
1775 \seq_gput_right:Nn
1776   \g_@@_renderers_seq
1777   { emphasis }
1778 \prop_gput:Nnn
1779   \g_@@_renderer_arities_prop
1780   { emphasis }
1781   { 1 }
1782 \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.

```

1783 \def\markdownRendererStrongEmphasis{%
1784   \markdownRendererStrongEmphasisPrototype}%
1785 \ExplSyntaxOn
1786 \seq_gput_right:Nn
1787   \g_@@_renderers_seq
1788   { strongEmphasis }
1789 \prop_gput:Nnn
1790   \g_@@_renderer_arities_prop
1791   { strongEmphasis }
1792   { 1 }
1793 \ExplSyntaxOff

```

2.2.5.13 Fenced Code Attribute Context Renderers

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

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

```

1794 \def\markdownRendererFencedCodeAttributeContextBegin{%
1795   \markdownRendererFencedCodeAttributeContextBeginPrototype}%
1796 \ExplSyntaxOn
1797 \seq_gput_right:Nn
1798   \g_@@_renderers_seq
1799   { fencedCodeAttributeContextBegin }
1800 \prop_gput:Nnn
1801   \g_@@_renderer_arities_prop
1802   { fencedCodeAttributeContextBegin }
1803   { 0 }
1804 \ExplSyntaxOff
1805 \def\markdownRendererFencedCodeAttributeContextEnd{%
1806   \markdownRendererFencedCodeAttributeContextEndPrototype}%
1807 \ExplSyntaxOn
1808 \seq_gput_right:Nn
1809   \g_@@_renderers_seq
1810   { fencedCodeAttributeContextEnd }
1811 \prop_gput:Nnn
1812   \g_@@_renderer_arities_prop
1813   { fencedCodeAttributeContextEnd }
1814   { 0 }
1815 \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.

```

1816 \def\markdownRendererFencedDivAttributeContextBegin{%
1817   \markdownRendererFencedDivAttributeContextBeginPrototype}%
1818 \ExplSyntaxOn
1819 \seq_gput_right:Nn
1820   \g_@@_renderers_seq
1821   { fencedDivAttributeContextBegin }
1822 \prop_gput:Nnn
1823   \g_@@_renderer_arities_prop
1824   { fencedDivAttributeContextBegin }
1825   { 0 }
1826 \ExplSyntaxOff
1827 \def\markdownRendererFencedDivAttributeContextEnd{%
1828   \markdownRendererFencedDivAttributeContextEndPrototype}%
1829 \ExplSyntaxOn

```

```

1830 \seq_gput_right:Nn
1831   \g_@@_renderers_seq
1832   { fencedDivAttributeContextEnd }
1833 \prop_gput:Nnn
1834   \g_@@_renderer_arities_prop
1835   { fencedDivAttributeContextEnd }
1836   { 0 }
1837 \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.

```

1838 \def\markdownRendererHeaderAttributeContextBegin{%
1839   \markdownRendererHeaderAttributeContextBeginPrototype}%
1840 \ExplSyntaxOn
1841 \seq_gput_right:Nn
1842   \g_@@_renderers_seq
1843   { headerAttributeContextBegin }
1844 \prop_gput:Nnn
1845   \g_@@_renderer_arities_prop
1846   { headerAttributeContextBegin }
1847   { 0 }
1848 \ExplSyntaxOff
1849 \def\markdownRendererHeaderAttributeContextEnd{%
1850   \markdownRendererHeaderAttributeContextEndPrototype}%
1851 \ExplSyntaxOn
1852 \seq_gput_right:Nn
1853   \g_@@_renderers_seq
1854   { headerAttributeContextEnd }
1855 \prop_gput:Nnn
1856   \g_@@_renderer_arities_prop
1857   { headerAttributeContextEnd }
1858   { 0 }
1859 \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.

```

1860 \def\markdownRendererHeadingOne{%
1861   \markdownRendererHeadingOnePrototype}%
1862 \ExplSyntaxOn

```

```

1863 \seq_gput_right:Nn
1864   \g_@@_renderers_seq
1865   { headingOne }
1866 \prop_gput:Nnn
1867   \g_@@_renderer_arities_prop
1868   { headingOne }
1869   { 1 }
1870 \ExplSyntaxOff

```

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

```

1871 \def\markdownRendererHeadingTwo{%
1872   \markdownRendererHeadingTwoPrototype}%
1873 \ExplSyntaxOn
1874 \seq_gput_right:Nn
1875   \g_@@_renderers_seq
1876   { headingTwo }
1877 \prop_gput:Nnn
1878   \g_@@_renderer_arities_prop
1879   { headingTwo }
1880   { 1 }
1881 \ExplSyntaxOff

```

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

```

1882 \def\markdownRendererHeadingThree{%
1883   \markdownRendererHeadingThreePrototype}%
1884 \ExplSyntaxOn
1885 \seq_gput_right:Nn
1886   \g_@@_renderers_seq
1887   { headingThree }
1888 \prop_gput:Nnn
1889   \g_@@_renderer_arities_prop
1890   { headingThree }
1891   { 1 }
1892 \ExplSyntaxOff

```

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

```

1893 \def\markdownRendererHeadingFour{%
1894   \markdownRendererHeadingFourPrototype}%
1895 \ExplSyntaxOn
1896 \seq_gput_right:Nn
1897   \g_@@_renderers_seq
1898   { headingFour }
1899 \prop_gput:Nnn
1900   \g_@@_renderer_arities_prop

```



```

1901 { headingFour }
1902 { 1 }
1903 \ExplSyntaxOff

```

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

```

1904 \def\markdownRendererHeadingFive{%
1905   \markdownRendererHeadingFivePrototype}%
1906 \ExplSyntaxOn
1907 \seq_gput_right:Nn
1908   \g_@@_renderers_seq
1909   { headingFive }
1910 \prop_gput:Nnn
1911   \g_@@_renderer_arities_prop
1912   { headingFive }
1913   { 1 }
1914 \ExplSyntaxOff

```

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

```

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

```

1926 \def\markdownRendererInlineHtmlComment{%
1927   \markdownRendererInlineHtmlCommentPrototype}%
1928 \ExplSyntaxOn
1929 \seq_gput_right:Nn
1930   \g_@@_renderers_seq
1931   { inlineHtmlComment }
1932 \prop_gput:Nnn
1933   \g_@@_renderer_arities_prop

```

```

1934 { inlineHtmlComment }
1935 { 1 }
1936 \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.

```

1937 \def\markdownRendererInlineHtmlTag{%
1938   \markdownRendererInlineHtmlTagPrototype}%
1939 \ExplSyntaxOn
1940 \seq_gput_right:Nn
1941   \g_@@_renderers_seq
1942   { inlineHtmlTag }
1943 \prop_gput:Nnn
1944   \g_@@_renderer_arities_prop
1945   { inlineHtmlTag }
1946   { 1 }
1947 \ExplSyntaxOff
1948 \def\markdownRendererInputBlockHtmlElement{%
1949   \markdownRendererInputBlockHtmlElementPrototype}%
1950 \ExplSyntaxOn
1951 \seq_gput_right:Nn
1952   \g_@@_renderers_seq
1953   { inputBlockHtmlElement }
1954 \prop_gput:Nnn
1955   \g_@@_renderer_arities_prop
1956   { inputBlockHtmlElement }
1957   { 1 }
1958 \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.

```

1959 \def\markdownRendererImage{%
1960   \markdownRendererImagePrototype}%
1961 \ExplSyntaxOn
1962 \seq_gput_right:Nn

```

```

1963 \g_@@_renderers_seq
1964 { image }
1965 \prop_gput:Nnn
1966 \g_@@_renderer_arities_prop
1967 { image }
1968 { 4 }
1969 \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.

```

1970 \def\markdownRendererImageAttributeContextBegin{%
1971 \markdownRendererImageAttributeContextBeginPrototype}%
1972 \ExplSyntaxOn
1973 \seq_gput_right:Nn
1974 \g_@@_renderers_seq
1975 { imageAttributeContextBegin }
1976 \prop_gput:Nnn
1977 \g_@@_renderer_arities_prop
1978 { imageAttributeContextBegin }
1979 { 0 }
1980 \ExplSyntaxOff
1981 \def\markdownRendererImageAttributeContextEnd{%
1982 \markdownRendererImageAttributeContextEndPrototype}%
1983 \ExplSyntaxOn
1984 \seq_gput_right:Nn
1985 \g_@@_renderers_seq
1986 { imageAttributeContextEnd }
1987 \prop_gput:Nnn
1988 \g_@@_renderer_arities_prop
1989 { imageAttributeContextEnd }
1990 { 0 }
1991 \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.

```

1992 \def\markdownRendererInterblockSeparator{%
1993 \markdownRendererInterblockSeparatorPrototype}%
1994 \ExplSyntaxOn
1995 \seq_gput_right:Nn

```

```

1996 \g_@@_renderers_seq
1997 { interblockSeparator }
1998 \prop_gput:Nnn
1999 \g_@@_renderer_arities_prop
2000 { interblockSeparator }
2001 { 0 }
2002 \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.

```

2003 \def\markdownRendererParagraphSeparator{%
2004 \markdownRendererParagraphSeparatorPrototype}%
2005 \ExplSyntaxOn
2006 \seq_gput_right:Nn
2007 \g_@@_renderers_seq
2008 { paragraphSeparator }
2009 \prop_gput:Nnn
2010 \g_@@_renderer_arities_prop
2011 { paragraphSeparator }
2012 { 0 }
2013 \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.

```

2014 \def\markdownRendererLineBlockBegin{%
2015 \markdownRendererLineBlockBeginPrototype}%
2016 \ExplSyntaxOn
2017 \seq_gput_right:Nn
2018 \g_@@_renderers_seq
2019 { lineBlockBegin }
2020 \prop_gput:Nnn
2021 \g_@@_renderer_arities_prop
2022 { lineBlockBegin }
2023 { 0 }
2024 \ExplSyntaxOff
2025 \def\markdownRendererLineBlockEnd{%
2026 \markdownRendererLineBlockEndPrototype}%
2027 \ExplSyntaxOn

```

```

2028 \seq_gput_right:Nn
2029   \g_@@_renderers_seq
2030   { lineBlockEnd }
2031 \prop_gput:Nnn
2032   \g_@@_renderer_arities_prop
2033   { lineBlockEnd }
2034   { 0 }
2035 \ExplSyntaxOff

```

2.2.5.23 Line Break Renderers

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

```

2036 \def\markdownRendererSoftLineBreak{%
2037   \markdownRendererSoftLineBreakPrototype}%
2038 \ExplSyntaxOn
2039 \seq_gput_right:Nn
2040   \g_@@_renderers_seq
2041   { softLineBreak }
2042 \prop_gput:Nnn
2043   \g_@@_renderer_arities_prop
2044   { softLineBreak }
2045   { 0 }
2046 \ExplSyntaxOff

```

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

```

2047 \def\markdownRendererHardLineBreak{%
2048   \markdownRendererHardLineBreakPrototype}%
2049 \ExplSyntaxOn
2050 \seq_gput_right:Nn
2051   \g_@@_renderers_seq
2052   { hardLineBreak }
2053 \prop_gput:Nnn
2054   \g_@@_renderer_arities_prop
2055   { hardLineBreak }
2056   { 0 }
2057 \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.

```

2058 \def\markdownRendererLink{%
2059   \markdownRendererLinkPrototype}%

```

```

2060 \ExplSyntaxOn
2061 \seq_gput_right:Nn
2062   \g_@@_renderers_seq
2063   { link }
2064 \prop_gput:Nnn
2065   \g_@@_renderer_arities_prop
2066   { link }
2067   { 4 }
2068 \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.

```

2069 \def\markdownRendererLinkAttributeContextBegin{%
2070   \markdownRendererLinkAttributeContextBeginPrototype}%
2071 \ExplSyntaxOn
2072 \seq_gput_right:Nn
2073   \g_@@_renderers_seq
2074   { linkAttributeContextBegin }
2075 \prop_gput:Nnn
2076   \g_@@_renderer_arities_prop
2077   { linkAttributeContextBegin }
2078   { 0 }
2079 \ExplSyntaxOff
2080 \def\markdownRendererLinkAttributeContextEnd{%
2081   \markdownRendererLinkAttributeContextEndPrototype}%
2082 \ExplSyntaxOn
2083 \seq_gput_right:Nn
2084   \g_@@_renderers_seq
2085   { linkAttributeContextEnd }
2086 \prop_gput:Nnn
2087   \g_@@_renderer_arities_prop
2088   { linkAttributeContextEnd }
2089   { 0 }
2090 \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.

```

2091 \def\markdownRendererMark{%

```

```

2092 \markdownRendererMarkPrototype}%
2093 \ExplSyntaxOn
2094 \seq_gput_right:Nn
2095 \g_@@_renderers_seq
2096 { mark }
2097 \prop_gput:Nnn
2098 \g_@@_renderer_arities_prop
2099 { mark }
2100 { 1 }
2101 \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.

```

2102 \def\markdownRendererDocumentBegin{%
2103 \markdownRendererDocumentBeginPrototype}%
2104 \ExplSyntaxOn
2105 \seq_gput_right:Nn
2106 \g_@@_renderers_seq
2107 { documentBegin }
2108 \prop_gput:Nnn
2109 \g_@@_renderer_arities_prop
2110 { documentBegin }
2111 { 0 }
2112 \ExplSyntaxOff
2113 \def\markdownRendererDocumentEnd{%
2114 \markdownRendererDocumentEndPrototype}%
2115 \ExplSyntaxOn
2116 \seq_gput_right:Nn
2117 \g_@@_renderers_seq
2118 { documentEnd }
2119 \prop_gput:Nnn
2120 \g_@@_renderer_arities_prop
2121 { documentEnd }
2122 { 0 }
2123 \ExplSyntaxOff

```

2.2.5.28 Non-Breaking Space Renderer

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

```

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

```

2135 \def\markdownRendererNote{%
2136   \markdownRendererNotePrototype}%
2137 \ExplSyntaxOn
2138 \seq_gput_right:Nn
2139   \g_@@_renderers_seq
2140   { note }
2141 \prop_gput:Nnn
2142   \g_@@_renderer_arities_prop
2143   { note }
2144   { 1 }
2145 \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.

```

2146 \def\markdownRendererOlBegin{%
2147   \markdownRendererOlBeginPrototype}%
2148 \ExplSyntaxOn
2149 \seq_gput_right:Nn
2150   \g_@@_renderers_seq
2151   { olBegin }
2152 \prop_gput:Nnn
2153   \g_@@_renderer_arities_prop
2154   { olBegin }
2155   { 0 }
2156 \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.

```

2157 \def\markdownRendererOlBeginTight{%
2158   \markdownRendererOlBeginTightPrototype}%
2159 \ExplSyntaxOn
2160 \seq_gput_right:Nn
2161   \g_@@_renderers_seq
2162   { olBeginTight }
2163 \prop_gput:Nnn
2164   \g_@@_renderer_arities_prop
2165   { olBeginTight }
2166   { 0 }
2167 \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`).

```

2168 \def\markdownRendererFancyOlBegin{%
2169   \markdownRendererFancyOlBeginPrototype}%
2170 \ExplSyntaxOn
2171 \seq_gput_right:Nn
2172   \g_@@_renderers_seq
2173   { fancyOlBegin }
2174 \prop_gput:Nnn
2175   \g_@@_renderer_arities_prop
2176   { fancyOlBegin }
2177   { 2 }
2178 \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.

```

2179 \def\markdownRendererFancyOlBeginTight{%
2180   \markdownRendererFancyOlBeginTightPrototype}%
2181 \ExplSyntaxOn
2182 \seq_gput_right:Nn
2183   \g_@@_renderers_seq

```

```

2184 { fancyOlBeginTight }
2185 \prop_gput:Nnn
2186 \g_@@_renderer_arities_prop
2187 { fancyOlBeginTight }
2188 { 2 }
2189 \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.

```

2190 \def\markdownRendererOlItem{%
2191 \markdownRendererOlItemPrototype}%
2192 \ExplSyntaxOn
2193 \seq_gput_right:Nn
2194 \g_@@_renderers_seq
2195 { olItem }
2196 \prop_gput:Nnn
2197 \g_@@_renderer_arities_prop
2198 { olItem }
2199 { 0 }
2200 \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.

```

2201 \def\markdownRendererOlItemEnd{%
2202 \markdownRendererOlItemEndPrototype}%
2203 \ExplSyntaxOn
2204 \seq_gput_right:Nn
2205 \g_@@_renderers_seq
2206 { olItemEnd }
2207 \prop_gput:Nnn
2208 \g_@@_renderer_arities_prop
2209 { olItemEnd }
2210 { 0 }
2211 \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.

```

2212 \def\markdownRendererOlItemWithNumber{%
2213 \markdownRendererOlItemWithNumberPrototype}%
2214 \ExplSyntaxOn
2215 \seq_gput_right:Nn
2216 \g_@@_renderers_seq

```

```

2217 { olItemWithNumber }
2218 \prop_gput:Nnn
2219 \g_@@_renderer_arities_prop
2220 { olItemWithNumber }
2221 { 1 }
2222 \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.

```

2223 \def\markdownRendererFancyOlItem{%
2224 \markdownRendererFancyOlItemPrototype}%
2225 \ExplSyntaxOn
2226 \seq_gput_right:Nn
2227 \g_@@_renderers_seq
2228 { fancyOlItem }
2229 \prop_gput:Nnn
2230 \g_@@_renderer_arities_prop
2231 { fancyOlItem }
2232 { 0 }
2233 \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.

```

2234 \def\markdownRendererFancyOlItemEnd{%
2235 \markdownRendererFancyOlItemEndPrototype}%
2236 \ExplSyntaxOn
2237 \seq_gput_right:Nn
2238 \g_@@_renderers_seq
2239 { fancyOlItemEnd }
2240 \prop_gput:Nnn
2241 \g_@@_renderer_arities_prop
2242 { fancyOlItemEnd }
2243 { 0 }
2244 \ExplSyntaxOff

```

The `\markdownRendererFancyOlItemWithNumber` 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.

```

2245 \def\markdownRendererFancyOlItemWithNumber{%
2246 \markdownRendererFancyOlItemWithNumberPrototype}%
2247 \ExplSyntaxOn
2248 \seq_gput_right:Nn
2249 \g_@@_renderers_seq

```

```

2250 { fancyO1ItemWithNumber }
2251 \prop_gput:Nnn
2252 \g_@@_renderer_arities_prop
2253 { fancyO1ItemWithNumber }
2254 { 1 }
2255 \ExplSyntaxOff

```

The `\markdownRendererO1End` macro represents the end of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

2256 \def\markdownRendererO1End{%
2257 \markdownRendererO1EndPrototype}%
2258 \ExplSyntaxOn
2259 \seq_gput_right:Nn
2260 \g_@@_renderers_seq
2261 { olEnd }
2262 \prop_gput:Nnn
2263 \g_@@_renderer_arities_prop
2264 { olEnd }
2265 { 0 }
2266 \ExplSyntaxOff

```

The `\markdownRendererO1EndTight` macro represents the end of an ordered list that contains no item with several paragraphs of text (the list is tight). This macro will only be produced, when the `tightLists` option is enabled and the `fancyLists` option is disabled. The macro receives no arguments.

```

2267 \def\markdownRendererO1EndTight{%
2268 \markdownRendererO1EndTightPrototype}%
2269 \ExplSyntaxOn
2270 \seq_gput_right:Nn
2271 \g_@@_renderers_seq
2272 { olEndTight }
2273 \prop_gput:Nnn
2274 \g_@@_renderer_arities_prop
2275 { olEndTight }
2276 { 0 }
2277 \ExplSyntaxOff

```

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

```

2278 \def\markdownRendererFancyO1End{%
2279 \markdownRendererFancyO1EndPrototype}%
2280 \ExplSyntaxOn

```

```

2281 \seq_gput_right:Nn
2282   \g_@@_renderers_seq
2283   { fancyO1End }
2284 \prop_gput:Nnn
2285   \g_@@_renderer_arities_prop
2286   { fancyO1End }
2287   { 0 }
2288 \ExplSyntaxOff

```

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

```

2289 \def\markdownRendererFancyO1EndTight{%
2290   \markdownRendererFancyO1EndTightPrototype}%
2291 \ExplSyntaxOn
2292 \seq_gput_right:Nn
2293   \g_@@_renderers_seq
2294   { fancyO1EndTight }
2295 \prop_gput:Nnn
2296   \g_@@_renderer_arities_prop
2297   { fancyO1EndTight }
2298   { 0 }
2299 \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.

```

2300 \def\markdownRendererInputRawInline{%
2301   \markdownRendererInputRawInlinePrototype}%
2302 \ExplSyntaxOn
2303 \seq_gput_right:Nn
2304   \g_@@_renderers_seq
2305   { inputRawInline }
2306 \prop_gput:Nnn
2307   \g_@@_renderer_arities_prop
2308   { inputRawInline }
2309   { 2 }
2310 \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.

```

2311 \def\markdownRendererInputRawBlock{%
2312   \markdownRendererInputRawBlockPrototype}%
2313 \ExplSyntaxOn
2314 \seq_gput_right:Nn
2315   \g_@@_renderers_seq
2316   { inputRawBlock }
2317 \prop_gput:Nnn
2318   \g_@@_renderer_arities_prop
2319   { inputRawBlock }
2320   { 2 }
2321 \ExplSyntaxOff

```

2.2.5.32 Section Renderers

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

```

2322 \def\markdownRendererSectionBegin{%
2323   \markdownRendererSectionBeginPrototype}%
2324 \ExplSyntaxOn
2325 \seq_gput_right:Nn
2326   \g_@@_renderers_seq
2327   { sectionBegin }
2328 \prop_gput:Nnn
2329   \g_@@_renderer_arities_prop
2330   { sectionBegin }
2331   { 0 }
2332 \ExplSyntaxOff
2333 \def\markdownRendererSectionEnd{%
2334   \markdownRendererSectionEndPrototype}%
2335 \ExplSyntaxOn
2336 \seq_gput_right:Nn
2337   \g_@@_renderers_seq
2338   { sectionEnd }
2339 \prop_gput:Nnn
2340   \g_@@_renderer_arities_prop
2341   { sectionEnd }
2342   { 0 }
2343 \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.

```

2344 \def\markdownRendererReplacementCharacter{%

```

```

2345 \markdownRendererReplacementCharacterPrototype}%
2346 \ExplSyntaxOn
2347 \seq_gput_right:Nn
2348 \g_@@_renderers_seq
2349 { replacementCharacter }
2350 \prop_gput:Nnn
2351 \g_@@_renderer_arities_prop
2352 { replacementCharacter }
2353 { 0 }
2354 \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`.

```

2355 \def\markdownRendererLeftBrace{%
2356 \markdownRendererLeftBracePrototype}%
2357 \ExplSyntaxOn
2358 \seq_gput_right:Nn
2359 \g_@@_renderers_seq
2360 { leftBrace }
2361 \prop_gput:Nnn
2362 \g_@@_renderer_arities_prop
2363 { leftBrace }
2364 { 0 }
2365 \ExplSyntaxOff
2366 \def\markdownRendererRightBrace{%
2367 \markdownRendererRightBracePrototype}%
2368 \ExplSyntaxOn
2369 \seq_gput_right:Nn
2370 \g_@@_renderers_seq
2371 { rightBrace }
2372 \prop_gput:Nnn
2373 \g_@@_renderer_arities_prop
2374 { rightBrace }
2375 { 0 }
2376 \ExplSyntaxOff
2377 \def\markdownRendererDollarSign{%
2378 \markdownRendererDollarSignPrototype}%
2379 \ExplSyntaxOn
2380 \seq_gput_right:Nn
2381 \g_@@_renderers_seq
2382 { dollarSign }
2383 \prop_gput:Nnn
2384 \g_@@_renderer_arities_prop
2385 { dollarSign }

```

```

2386 { 0 }
2387 \ExplSyntaxOff
2388 \def\markdownRendererPercentSign{%
2389 \markdownRendererPercentSignPrototype}%
2390 \ExplSyntaxOn
2391 \seq_gput_right:Nn
2392 \g_@@_renderers_seq
2393 { percentSign }
2394 \prop_gput:Nnn
2395 \g_@@_renderer_arities_prop
2396 { percentSign }
2397 { 0 }
2398 \ExplSyntaxOff
2399 \def\markdownRendererAmpersand{%
2400 \markdownRendererAmpersandPrototype}%
2401 \ExplSyntaxOn
2402 \seq_gput_right:Nn
2403 \g_@@_renderers_seq
2404 { ampersand }
2405 \prop_gput:Nnn
2406 \g_@@_renderer_arities_prop
2407 { ampersand }
2408 { 0 }
2409 \ExplSyntaxOff
2410 \def\markdownRendererUnderscore{%
2411 \markdownRendererUnderscorePrototype}%
2412 \ExplSyntaxOn
2413 \seq_gput_right:Nn
2414 \g_@@_renderers_seq
2415 { underscore }
2416 \prop_gput:Nnn
2417 \g_@@_renderer_arities_prop
2418 { underscore }
2419 { 0 }
2420 \ExplSyntaxOff
2421 \def\markdownRendererHash{%
2422 \markdownRendererHashPrototype}%
2423 \ExplSyntaxOn
2424 \seq_gput_right:Nn
2425 \g_@@_renderers_seq
2426 { hash }
2427 \prop_gput:Nnn
2428 \g_@@_renderer_arities_prop
2429 { hash }
2430 { 0 }
2431 \ExplSyntaxOff
2432 \def\markdownRendererCircumflex{%

```



```

2433 \markdownRendererCircumflexPrototype}%
2434 \ExplSyntaxOn
2435 \seq_gput_right:Nn
2436 \g_@@_renderers_seq
2437 { circumflex }
2438 \prop_gput:Nnn
2439 \g_@@_renderer_arities_prop
2440 { circumflex }
2441 { 0 }
2442 \ExplSyntaxOff
2443 \def\markdownRendererBackslash{%
2444 \markdownRendererBackslashPrototype}%
2445 \ExplSyntaxOn
2446 \seq_gput_right:Nn
2447 \g_@@_renderers_seq
2448 { backslash }
2449 \prop_gput:Nnn
2450 \g_@@_renderer_arities_prop
2451 { backslash }
2452 { 0 }
2453 \ExplSyntaxOff
2454 \def\markdownRendererTilde{%
2455 \markdownRendererTildePrototype}%
2456 \ExplSyntaxOn
2457 \seq_gput_right:Nn
2458 \g_@@_renderers_seq
2459 { tilde }
2460 \prop_gput:Nnn
2461 \g_@@_renderer_arities_prop
2462 { tilde }
2463 { 0 }
2464 \ExplSyntaxOff
2465 \def\markdownRendererPipe{%
2466 \markdownRendererPipePrototype}%
2467 \ExplSyntaxOn
2468 \seq_gput_right:Nn
2469 \g_@@_renderers_seq
2470 { pipe }
2471 \prop_gput:Nnn
2472 \g_@@_renderer_arities_prop
2473 { pipe }
2474 { 0 }
2475 \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.

```
2476 \def\markdownRendererStrikeThrough{%
2477   \markdownRendererStrikeThroughPrototype}%
2478 \ExplSyntaxOn
2479 \seq_gput_right:Nn
2480   \g_@@_renderers_seq
2481   { strikeThrough }
2482 \prop_gput:Nnn
2483   \g_@@_renderer_arities_prop
2484   { strikeThrough }
2485   { 1 }
2486 \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.

```
2487 \def\markdownRendererSubscript{%
2488   \markdownRendererSubscriptPrototype}%
2489 \ExplSyntaxOn
2490 \seq_gput_right:Nn
2491   \g_@@_renderers_seq
2492   { subscript }
2493 \prop_gput:Nnn
2494   \g_@@_renderer_arities_prop
2495   { subscript }
2496   { 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.

```
2497 \def\markdownRendererSuperscript{%
2498   \markdownRendererSuperscriptPrototype}%
2499 \ExplSyntaxOn
2500 \seq_gput_right:Nn
2501   \g_@@_renderers_seq
2502   { superscript }
2503 \prop_gput:Nnn
2504   \g_@@_renderer_arities_prop
2505   { superscript }
2506   { 1 }
```

2507 \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.

```
2508 \def\markdownRendererTableAttributeContextBegin{%
2509   \markdownRendererTableAttributeContextBeginPrototype}%
2510 \ExplSyntaxOn
2511 \seq_gput_right:Nn
2512   \g_@@_renderers_seq
2513   { tableAttributeContextBegin }
2514 \prop_gput:Nnn
2515   \g_@@_renderer_arities_prop
2516   { tableAttributeContextBegin }
2517   { 0 }
2518 \ExplSyntaxOff
2519 \def\markdownRendererTableAttributeContextEnd{%
2520   \markdownRendererTableAttributeContextEndPrototype}%
2521 \ExplSyntaxOn
2522 \seq_gput_right:Nn
2523   \g_@@_renderers_seq
2524   { tableAttributeContextEnd }
2525 \prop_gput:Nnn
2526   \g_@@_renderer_arities_prop
2527   { tableAttributeContextEnd }
2528   { 0 }
2529 \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.

```

2530 \def\markdownRendererTable{%
2531   \markdownRendererTablePrototype}%
2532 \ExplSyntaxOn
2533 \seq_gput_right:Nn
2534   \g_@@_renderers_seq
2535   { table }
2536 \prop_gput:Nnn
2537   \g_@@_renderer_arities_prop
2538   { table }
2539   { 3 }
2540 \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.

```

2541 \def\markdownRendererInlineMath{%
2542   \markdownRendererInlineMathPrototype}%
2543 \ExplSyntaxOn
2544 \seq_gput_right:Nn
2545   \g_@@_renderers_seq
2546   { inlineMath }
2547 \prop_gput:Nnn
2548   \g_@@_renderer_arities_prop
2549   { inlineMath }
2550   { 1 }
2551 \ExplSyntaxOff
2552 \def\markdownRendererDisplayMath{%
2553   \markdownRendererDisplayMathPrototype}%
2554 \ExplSyntaxOn
2555 \seq_gput_right:Nn
2556   \g_@@_renderers_seq
2557   { displayMath }
2558 \prop_gput:Nnn
2559   \g_@@_renderer_arities_prop
2560   { displayMath }
2561   { 1 }
2562 \ExplSyntaxOff

```

2.2.5.41 Thematic Break Renderer

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

```

2563 \def\markdownRendererThematicBreak{%
2564   \markdownRendererThematicBreakPrototype}%
2565 \ExplSyntaxOn
2566 \seq_gput_right:Nn
2567   \g_@@_renderers_seq
2568   { thematicBreak }
2569 \prop_gput:Nnn
2570   \g_@@_renderer_arities_prop
2571   { thematicBreak }
2572   { 0 }
2573 \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.

```

2574 \def\markdownRendererTickedBox{%
2575   \markdownRendererTickedBoxPrototype}%
2576 \ExplSyntaxOn
2577 \seq_gput_right:Nn
2578   \g_@@_renderers_seq
2579   { tickedBox }
2580 \prop_gput:Nnn
2581   \g_@@_renderer_arities_prop
2582   { tickedBox }
2583   { 0 }
2584 \ExplSyntaxOff
2585 \def\markdownRendererHalfTickedBox{%
2586   \markdownRendererHalfTickedBoxPrototype}%
2587 \ExplSyntaxOn
2588 \seq_gput_right:Nn
2589   \g_@@_renderers_seq
2590   { halfTickedBox }
2591 \prop_gput:Nnn
2592   \g_@@_renderer_arities_prop
2593   { halfTickedBox }
2594   { 0 }
2595 \ExplSyntaxOff
2596 \def\markdownRendererUntickedBox{%
2597   \markdownRendererUntickedBoxPrototype}%
2598 \ExplSyntaxOn
2599 \seq_gput_right:Nn
2600   \g_@@_renderers_seq
2601   { untickedBox }

```

```

2602 \prop_gput:Nnn
2603   \g_@@_renderer_arities_prop
2604   { untickedBox }
2605   { 0 }
2606 \ExplSyntaxOff

```

2.2.5.43 YAML Metadata Renderers

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

```

2607 \def\markdownRendererJekyllDataBegin{%
2608   \markdownRendererJekyllDataBeginPrototype}%
2609 \ExplSyntaxOn
2610 \seq_gput_right:Nn
2611   \g_@@_renderers_seq
2612   { jekyllDataBegin }
2613 \prop_gput:Nnn
2614   \g_@@_renderer_arities_prop
2615   { jekyllDataBegin }
2616   { 0 }
2617 \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.

```

2618 \def\markdownRendererJekyllDataEnd{%
2619   \markdownRendererJekyllDataEndPrototype}%
2620 \ExplSyntaxOn
2621 \seq_gput_right:Nn
2622   \g_@@_renderers_seq
2623   { jekyllDataEnd }
2624 \prop_gput:Nnn
2625   \g_@@_renderer_arities_prop
2626   { jekyllDataEnd }
2627   { 0 }
2628 \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.

```

2629 \def\markdownRendererJekyllDataMappingBegin{%
2630   \markdownRendererJekyllDataMappingBeginPrototype}%
2631 \ExplSyntaxOn

```

```

2632 \seq_gput_right:Nn
2633   \g_@@_renderers_seq
2634   { jekyllDataMappingBegin }
2635 \prop_gput:Nnn
2636   \g_@@_renderer_arities_prop
2637   { jekyllDataMappingBegin }
2638   { 2 }
2639 \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.

```

2640 \def\markdownRendererJekyllDataMappingEnd{%
2641   \markdownRendererJekyllDataMappingEndPrototype}%
2642 \ExplSyntaxOn
2643 \seq_gput_right:Nn
2644   \g_@@_renderers_seq
2645   { jekyllDataMappingEnd }
2646 \prop_gput:Nnn
2647   \g_@@_renderer_arities_prop
2648   { jekyllDataMappingEnd }
2649   { 0 }
2650 \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.

```

2651 \def\markdownRendererJekyllDataSequenceBegin{%
2652   \markdownRendererJekyllDataSequenceBeginPrototype}%
2653 \ExplSyntaxOn
2654 \seq_gput_right:Nn
2655   \g_@@_renderers_seq
2656   { jekyllDataSequenceBegin }
2657 \prop_gput:Nnn
2658   \g_@@_renderer_arities_prop
2659   { jekyllDataSequenceBegin }
2660   { 2 }
2661 \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.

```

2662 \def\markdownRendererJekyllDataSequenceEnd{%
2663   \markdownRendererJekyllDataSequenceEndPrototype}%

```

```

2664 \ExplSyntaxOn
2665 \seq_gput_right:Nn
2666   \g_@@_renderers_seq
2667   { jekyllDataSequenceEnd }
2668 \prop_gput:Nnn
2669   \g_@@_renderer_arities_prop
2670   { jekyllDataSequenceEnd }
2671   { 0 }
2672 \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.

```

2673 \def\markdownRendererJekyllDataBoolean{%
2674   \markdownRendererJekyllDataBooleanPrototype}%
2675 \ExplSyntaxOn
2676 \seq_gput_right:Nn
2677   \g_@@_renderers_seq
2678   { jekyllDataBoolean }
2679 \prop_gput:Nnn
2680   \g_@@_renderer_arities_prop
2681   { jekyllDataBoolean }
2682   { 2 }
2683 \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.

```

2684 \def\markdownRendererJekyllDataNumber{%
2685   \markdownRendererJekyllDataNumberPrototype}%
2686 \ExplSyntaxOn
2687 \seq_gput_right:Nn
2688   \g_@@_renderers_seq
2689   { jekyllDataNumber }
2690 \prop_gput:Nnn
2691   \g_@@_renderer_arities_prop
2692   { jekyllDataNumber }
2693   { 2 }
2694 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataString` macro represents a string scalar value in a YAML document. This macro will only be produced when the `jekyllData` option

is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the scalar value.

```

2695 \def\markdownRendererJekyllDataString{%
2696   \markdownRendererJekyllDataStringPrototype}%
2697 \ExplSyntaxOn
2698 \seq_gput_right:Nn
2699   \g_@@_renderers_seq
2700   { jekyllDataString }
2701 \prop_gput:Nnn
2702   \g_@@_renderer_arities_prop
2703   { jekyllDataString }
2704   { 2 }
2705 \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.

```

2706 \def\markdownRendererJekyllDataEmpty{%
2707   \markdownRendererJekyllDataEmptyPrototype}%
2708 \ExplSyntaxOn
2709 \seq_gput_right:Nn
2710   \g_@@_renderers_seq
2711   { jekyllDataEmpty }
2712 \prop_gput:Nnn
2713   \g_@@_renderer_arities_prop
2714   { jekyllDataEmpty }
2715   { 1 }
2716 \ExplSyntaxOff

```

2.2.5.44 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.

```

2717 \ExplSyntaxOn
2718 \cs_new:Nn \@@_define_renderers:
2719   {
2720     \seq_map_function:NN
2721       \g_@@_renderers_seq
2722       \@@_define_renderer:n

```

```

2723 }
2724 \cs_new:Nn \@@_define_renderer:n
2725 {
2726   \@@_renderer_tl_to_csname:nN
2727   { #1 }
2728   \l_tmpa_tl
2729   \prop_get:NnN
2730   \g_@@_renderer_arities_prop
2731   { #1 }
2732   \l_tmpb_tl
2733   \@@_define_renderer:ncV
2734   { #1 }
2735   { \l_tmpa_tl }
2736   \l_tmpb_tl
2737 }
2738 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2739 {
2740   \tl_set:Nn
2741   \l_tmpa_tl
2742   { \str_uppercase:n { #1 } }
2743   \tl_set:Nx
2744   #2
2745   {
2746     markdownRenderer
2747     \tl_head:f { \l_tmpa_tl }
2748     \tl_tail:n { #1 }
2749   }
2750 }
2751 \tl_new:N
2752 \l_@@_renderer_definition_tl
2753 \bool_new:N
2754 \g_@@_appending_renderer_bool
2755 \cs_new:Nn \@@_define_renderer:nNn
2756 {
2757   \keys_define:nn
2758   { markdown/options/renderers }
2759   {
2760     #1 .code:n = {
2761       \tl_set:Nn
2762       \l_@@_renderer_definition_tl
2763       { ##1 }
2764       \regex_replace_all:nnN
2765       { \cP\#0 }
2766       { #1 }
2767       \l_@@_renderer_definition_tl
2768       \bool_if:NT
2769       \g_@@_appending_renderer_bool

```

```

2770         {
2771         \@@_tl_set_from_cs:NNn
2772         \l_tmpa_tl
2773         #2
2774         { #3 }
2775         \tl_put_left:NV
2776         \l_@@_renderer_definition_tl
2777         \l_tmpa_tl
2778         }
2779     \cs_generate_from_arg_count:NNnV
2780     #2
2781     \cs_set:Npn
2782     { #3 }
2783     \l_@@_renderer_definition_tl
2784 },
2785 }
2786 }

```

We define the function `\@@_tl_set_from_cs:NNn` [9]. The function takes a token list, a control sequence with undelimited parameters, and the number of parameters the control sequence accepts, and locally assigns the replacement text of the control sequence to the token list.

```

2787 \cs_new_protected:Nn
2788 \@@_tl_set_from_cs:NNn
2789 {
2790     \tl_set:Nn
2791     \l_tmpa_tl
2792     { #2 }
2793     \int_step_inline:nn
2794     { #3 }
2795     {
2796         \exp_args:Nnc
2797         \tl_put_right:Nn
2798         \l_tmpa_tl
2799         { @@_tl_set_from_cs_parameter_ ##1 }
2800     }
2801     \exp_args:NNV
2802     \tl_set:No
2803     \l_tmpb_tl
2804     \l_tmpa_tl
2805     \regex_replace_all:nnN
2806     { \cP. }
2807     { \0\0 }
2808     \l_tmpb_tl
2809     \int_step_inline:nn
2810     { #3 }
2811     {

```

```

2812     \regex_replace_all:nnN
2813     { \c { @@_tl_set_from_cs_parameter_ ##1 } }
2814     { \cP\# ##1 }
2815     \l_tmpb_tl
2816   }
2817   \tl_set:NV
2818   #1
2819   \l_tmpb_tl
2820 }
2821 \cs_generate_variant:Nn
2822 \@@_define_renderer:nNn
2823 { ncV }
2824 \cs_generate_variant:Nn
2825 \cs_generate_from_arg_count:NNnn
2826 { NNnV }
2827 \cs_generate_variant:Nn
2828 \tl_put_left:Nn
2829 { Nv }
2830 \keys_define:nn
2831 { markdown/options }
2832 {
2833   renderers .code:n = {
2834     \keys_set:nn
2835     { markdown/options/renderers }
2836     { #1 }
2837   },
2838 }

```

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.
  }
}

```

```

2839 \tl_new:N
2840 \l_@@_renderer_glob_definition_tl
2841 \seq_new:N
2842 \l_@@_renderer_glob_results_seq
2843 \regex_const:Nn
2844 \c_@@_appending_key_regex
2845 { \s*+$ }
2846 \keys_define:nn

```

```

2847 { markdown/options/renderers }
2848 {
2849     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 += {...},
        },
      }
    },
  },
}

```

```

2850     \regex_match:NVTF
2851     \c_@@_appending_key_regex
2852     \l_keys_key_str
2853     {
2854         \bool_gset_true:N
2855         \g_@@_appending_renderer_bool
2856         \tl_set:NV
2857         \l_tmpa_tl
2858         \l_keys_key_str
2859         \regex_replace_once:NnN
2860         \c_@@_appending_key_regex

```

```

2861         { }
2862         \l_tmpa_tl
2863         \tl_set:Nx
2864         \l_tmpb_tl
2865         { { \l_tmpa_tl } = }
2866         \tl_put_right:Nn
2867         \l_tmpb_tl
2868         { { #1 } }
2869         \keys_set:nV
2870         { markdown/options/renderers }
2871         \l_tmpb_tl
2872         \bool_gset_false:N
2873         \g_@@_appending_renderer_bool
2874     }

```

In addition to exact token renderer names, we also support wildcards (*) and enumerations (1) that match multiple token renderer names:

```

\markdownSetup{
  renderers = {
    heading* = {{\bf #1}},      % Render headings using the bold face.
    jekyllData(String|Number) = {% % Render YAML string and numbers
      {\it #2}%                % using italics.
    },
  }
}

```

Wildcards and enumerations can be combined:

```

\markdownSetup{
  renderers = {
    *1Item(|End) = {"},      % Quote ordered/bullet list items.
  }
}

```

To determine the current token renderer, you can use the pseudo-parameter #0:

```

\markdownSetup{
  renderers = {
    heading* = {#0: #1},      % Render headings as the renderer name
                                % followed by the heading text.
  }
}

```

```

2875     {
2876         \@_glob_seq:VnN
2877         \l_keys_key_str
2878         { g_@_renderers_seq }
2879         \l_@_renderer_glob_results_seq
2880         \seq_if_empty:NTF
2881         \l_@_renderer_glob_results_seq
2882         {
2883             \msg_error:nnV
2884             { markdown }
2885             { undefined-renderer }
2886             \l_keys_key_str
2887         }
2888         {
2889             \tl_set:Nn
2890             \l_@_renderer_glob_definition_tl
2891             { \exp_not:n { #1 } }
2892             \seq_map_inline:Nn
2893             \l_@_renderer_glob_results_seq
2894             {
2895                 \tl_set:Nn
2896                 \l_tmpa_tl
2897                 { { ##1 } = }
2898                 \tl_put_right:Nx
2899                 \l_tmpa_tl
2900                 { { \l_@_renderer_glob_definition_tl } }
2901                 \keys_set:nV
2902                 { markdown/options/renderers }
2903                 \l_tmpa_tl
2904             }
2905         }
2906     }
2907 },
2908 }
2909 \msg_new:nnn
2910 { markdown }
2911 { undefined-renderer }
2912 {
2913     Renderer~#1~is~undefined.
2914 }
2915 \cs_generate_variant:Nn
2916 \@_glob_seq:nnN
2917 { VnN }
2918 \cs_generate_variant:Nn
2919 \cs_generate_from_arg_count:NNnn
2920 { cNVV }
2921 \cs_generate_variant:Nn

```

```

2922 \msg_error:nnn
2923 { nnV }
2924 \prg_generate_conditional_variant:Nnn
2925 \regex_match:Nn
2926 { NV }
2927 { TF }
2928 \prop_new:N
2929 \g_@@_glob_cache_prop
2930 \tl_new:N
2931 \l_@@_current_glob_tl
2932 \cs_new:Nn
2933 \@@_glob_seq:nnN
2934 {
2935   \tl_set:Nn
2936     \l_@@_current_glob_tl
2937     { ^ #1 $ }
2938   \prop_get:NeNTF
2939     \g_@@_glob_cache_prop
2940     { #2 / \l_@@_current_glob_tl }
2941     \l_tmpa_clist
2942     {
2943       \seq_set_from_clist:NN
2944         #3
2945         \l_tmpa_clist
2946     }
2947   {
2948     \seq_clear:N
2949       #3
2950     \regex_replace_all:nnN
2951       { \* }
2952       { .* }
2953       \l_@@_current_glob_tl
2954     \regex_set:NV
2955       \l_tmpa_regex
2956       \l_@@_current_glob_tl
2957     \seq_map_inline:cn
2958       { #2 }
2959     {
2960       \regex_match:NnT
2961         \l_tmpa_regex
2962         { ##1 }
2963         {
2964           \seq_put_right:Nn
2965             #3
2966             { ##1 }
2967         }
2968     }

```



```

2969     \clist_set_from_seq:NN
2970     \l_tmpa_clist
2971     #3
2972     \prop_gput:NeV
2973     \g_@@_glob_cache_prop
2974     { #2 / \l_@@_current_glob_tl }
2975     \l_tmpa_clist
2976   }
2977 }
2978 % TODO: Remove in TeX Live 2023.
2979 \prg_generate_conditional_variant:Nnn
2980   \prop_get:NnN
2981   { NeN }
2982   { TF }
2983 \cs_generate_variant:Nn
2984   \regex_set:Nn
2985   { NV }
2986 \cs_generate_variant:Nn
2987   \prop_gput:Nnn
2988   { 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.

```

2989 \str_if_eq:VVT
2990   \c_@@_top_layer_tl
2991   \c_@@_option_layer_plain_tex_tl
2992   {
2993     \@@_define_renderers:
2994   }
2995 \ExplSyntaxOff

```

2.2.6 Token Renderer Prototypes

2.2.6.1 YAML Metadata Renderer Prototypes

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

```

2996 \ExplSyntaxOn
2997 \keys_define:nn
2998   { markdown/jekyllData }
2999   { }
3000 \ExplSyntaxOff

```

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

```

3001 \ExplSyntaxOn
3002 \@@_with_various_cases:nn
3003 { jekyllDataRenderers }
3004 {
3005   \keys_define:nn
3006     { markdown/options }
3007     {
3008       #1 .code:n = {
3009         \tl_set:Nn
3010           \l_tmpa_tl
3011           { ##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.

```

3012         \tl_replace_all:NnV
3013           \l_tmpa_tl
3014           { / }
3015         \c_backslash_str
3016       \keys_set:nV
3017         { markdown/options/jekyll-data-renderers }
3018         \l_tmpa_tl
3019     },
3020 }
3021 }
3022 \keys_define:nn
3023 { markdown/options/jekyll-data-renderers }
3024 {
3025   unknown .code:n = {
3026     \tl_set_eq:NN
3027       \l_tmpa_tl
3028       \l_keys_key_str
3029     \tl_replace_all:NVn
3030       \l_tmpa_tl
3031       \c_backslash_str
3032       { / }
3033     \tl_put_right:Nn
3034       \l_tmpa_tl
3035       {
3036         .code:n = { #1 }
3037       }
3038     \keys_define:nV
3039       { markdown/jekyllData }
3040       \l_tmpa_tl
3041   }

```

```

3042 }
3043 \cs_generate_variant:Nn
3044 \keys_define:nn
3045 { nV }
3046 \ExplSyntaxOff

```

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

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

```

3047 \ExplSyntaxOn
3048 \cs_new:Nn \@@_define_renderer_prototypes:
3049 {
3050   \seq_map_function:NN
3051     \g_@@_renderers_seq
3052     \@@_define_renderer_prototype:n
3053 }
3054 \cs_new:Nn \@@_define_renderer_prototype:n
3055 {
3056   \@@_renderer_prototype_tl_to_csname:nN
3057     { #1 }
3058     \l_tmpa_tl
3059   \prop_get:NnN
3060     \g_@@_renderers_arities_prop
3061     { #1 }
3062     \l_tmpb_tl
3063   \@@_define_renderer_prototype:ncV
3064     { #1 }
3065     { \l_tmpa_tl }
3066     \l_tmpb_tl
3067 }
3068 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
3069 {
3070   \tl_set:Nn
3071     \l_tmpa_tl
3072     { \str_uppercase:n { #1 } }
3073   \tl_set:Nx
3074     #2
3075     {
3076       markdownRenderer
3077       \tl_head:f { \l_tmpa_tl }
3078       \tl_tail:n { #1 }
3079       Prototype

```

```

3080     }
3081   }
3082 \tl_new:N
3083   \l_@@_renderer_prototype_definition_tl
3084 \bool_new:N
3085   \g_@@_appending_renderer_prototype_bool
3086 \cs_new:Nn \@@_define_renderer_prototype:nNn
3087   {
3088     \keys_define:nn
3089       { markdown/options/renderer-prototypes }
3090       {
3091         #1 .code:n = {
3092           \tl_set:Nn
3093             \l_@@_renderer_prototype_definition_tl
3094             { ##1 }
3095           \regex_replace_all:nnN
3096             { \cP\#0 }
3097             { #1 }
3098           \l_@@_renderer_prototype_definition_tl
3099           \bool_if:NT
3100             \g_@@_appending_renderer_prototype_bool
3101             {
3102               \@@_tl_set_from_cs:NNn
3103               \l_tmpa_tl
3104               #2
3105               { #3 }
3106               \tl_put_left:NV
3107                 \l_@@_renderer_prototype_definition_tl
3108                 \l_tmpa_tl
3109             }
3110           \cs_generate_from_arg_count:NNnV
3111             #2
3112             \cs_set:Npn
3113               { #3 }
3114             \l_@@_renderer_prototype_definition_tl
3115         },
3116       }

```

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

```

3117   \cs_if_free:NT
3118     #2
3119     {
3120       \cs_generate_from_arg_count:NNnn
3121         #2
3122         \cs_set:Npn
3123           { #3 }

```

```

3124         { }
3125     }
3126 }
3127 \cs_generate_variant:Nn
3128   \@_define_renderer_prototype:nNn
3129   { 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.
  }
}

```

```

3130 \keys_define:nn
3131   { markdown/options/renderer-prototypes }
3132   {
3133     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 += {...},
    },
  },
},
}

```

```

3134   \regex_match:NVTF
3135   \c_@@_appending_key_regex
3136   \l_keys_key_str
3137   {
3138     \bool_gset_true:N
3139     \g_@@_appending_renderer_prototype_bool
3140     \tl_set:NV
3141     \l_tmpa_tl
3142     \l_keys_key_str
3143     \regex_replace_once:NnN
3144     \c_@@_appending_key_regex
3145     { }
3146     \l_tmpa_tl
3147     \tl_set:Nx
3148     \l_tmpb_tl
3149     { { \l_tmpa_tl } = }
3150     \tl_put_right:Nn
3151     \l_tmpb_tl
3152     { { #1 } }
3153     \keys_set:nV
3154     { markdown/options/renderer-prototypes }
3155     \l_tmpb_tl
3156     \bool_gset_false:N
3157     \g_@@_appending_renderer_prototype_bool
3158   }

```

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.
}
```

```
3159     {
3160         \@@_glob_seq:VnN
3161         \l_keys_key_str
3162         { g_@@_renderers_seq }
3163         \l_@@_renderer_glob_results_seq
3164         \seq_if_empty:NTF
3165         \l_@@_renderer_glob_results_seq
3166         {
3167             \msg_error:nnV
3168             { markdown }
3169             { undefined-renderer-prototype }
3170             \l_keys_key_str
3171         }
3172     {
3173         \tl_set:Nn
3174         \l_@@_renderer_glob_definition_tl
3175         { \exp_not:n { #1 } }
3176         \seq_map_inline:Nn
3177         \l_@@_renderer_glob_results_seq
3178         {
3179             \tl_set:Nn
3180             \l_tmpa_tl
3181             { { ##1 } = }
3182             \tl_put_right:Nx
3183             \l_tmpa_tl
3184             { { \l_@@_renderer_glob_definition_tl } }
3185             \keys_set:nV
3186             { markdown/options/renderer-prototypes }
```

```

3187             \l_tmpa_tl
3188         }
3189     }
3190 }
3191 },
3192 }
3193 \msg_new:nnn
3194 { markdown }
3195 { undefined-renderer-prototype }
3196 {
3197     Renderer~prototype~#1-is~undefined.
3198 }
3199 \@@_with_various_cases:nn
3200 { rendererPrototypes }
3201 {
3202     \keys_define:nn
3203     { markdown/options }
3204     {
3205         #1 .code:n = {
3206             \keys_set:nn
3207             { markdown/options/renderer-prototypes }
3208             { ##1 }
3209         },
3210     }
3211 }

```

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

```

3212 \str_if_eq:VVT
3213   \c_@@_top_layer_tl
3214   \c_@@_option_layer_plain_tex_tl
3215   {
3216     \@@_define_renderer_prototypes:
3217   }
3218 \ExplSyntaxOff

```

2.2.7 Logging Facilities

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

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

2.2.8 Miscellanea

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

```
3219 \let\markdownMakeOther\relax
```

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

```
3220 \let\markdownReadAndConvert\relax
3221 \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*.

```
3222 \catcode`\|=0\catcode`\=12%
3223 |gdef|markdownBegin{%
3224   |markdownReadAndConvert{\markdownEnd}%
3225                               {\|markdownEnd}}%
3226 |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.

```
3227 \ExplSyntaxOn
3228 \keys_define:nn
3229   { markdown/options }
3230   {
3231     code .code:n = { #1 },
3232   }
3233 \ExplSyntaxOff
```

This can be especially useful in snippets.

2.3 \LaTeX Interface

The \LaTeX interface provides \LaTeX environments for the typesetting of markdown input from within \LaTeX , facilities for setting Lua, plain \TeX , and \LaTeX options used during the conversion from markdown to plain \TeX , and facilities for changing

the way markdown tokens are rendered. The rest of the interface is inherited from the plain \TeX interface (see Section 2.2).

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

```

3234 \ExplSyntaxOn
3235 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
3236 \cs_generate_variant:Nn
3237   \tl_const:Nn
3238   { NV }
3239 \tl_if_exist:NF
3240   \c_@@_top_layer_tl
3241   {
3242     \tl_const:NV
3243       \c_@@_top_layer_tl
3244       \c_@@_option_layer_latex_tl
3245   }
3246 \ExplSyntaxOff
3247 \input markdown/markdown

```

The \LaTeX interface is implemented by the `markdown.sty` file, which can be loaded from the \LaTeX document preamble as follows:

```

\usepackage[<options>]{markdown}

```

where `<options>` are the \LaTeX interface options (see Section 2.3.2). Note that `<options>` inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.2.5.44) and `markdownRendererPrototypes` (see Section 2.2.6.2) keys. Furthermore, although the base variant of the `import` key that loads a single \LaTeX theme (see Section 2.3.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.2.4) cannot. This limitation is due to the way $\text{\LaTeX} 2_{\epsilon}$ parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` \LaTeX environments, and redefines the `\markinline` and `\markdownInput` commands.

The `markdown` and `markdown*` \LaTeX environments are used to typeset markdown document fragments. Both \LaTeX environments accept \LaTeX interface options (see Section 2.3.2) as the only argument. This argument is optional for the `markdown` environment and mandatory for the `markdown*` environment.

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

```

3248 \newenvironment{markdown}\relax\relax
3249 \newenvironment{markdown*}[1]\relax\relax

```

You may prepend your own code to the `\markdown` macro and append your own code to the `\markdownEnd` macro to produce special effects before and after the `markdown` L^AT_EX environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L^AT_EX environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T_EX interface.

The following example L^AT_EX code showcases the usage of the `markdown` and `markdown*` environments:

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

The `\markinline` macro accepts a single mandatory parameter containing inline markdown content and expands to the result of the conversion of the input markdown document to plain T_EX. Unlike the `\markinline` macro provided by the plain T_EX interface, this macro also accepts L^AT_EX interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown content.

The `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T_EX. Unlike the `\markdownInput` macro provided by the plain T_EX interface, this macro also accepts L^AT_EX interface options (see Section 2.3.2) 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:

<code>\documentclass{article}</code>
<code>\usepackage{markdown}</code>
<code>\begin{document}</code>
<code>\markdownInput[smartEllipses]{hello.md}</code>
<code>\end{document}</code>

2.3.2 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.

2.3.2.1 Finalizing and Freezing the Cache

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics to the `finalizeCache` and `frozenCache` plain \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.

```
3250 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
3251 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

2.3.2.2 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.

```
3252 \ExplSyntaxOn
3253 \str_if_eq:VVT
3254   \c_@@_top_layer_tl
3255   \c_@@_option_layer_latex_tl
```

```

3256 {
3257   \@@_define_option_commands_and_keyvals:
3258   \@@_define_renderers:
3259   \@@_define_renderer_prototypes:
3260 }
3261 \ExplSyntaxOff

```

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

```

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

```

2.3.3 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` \LaTeX package, and finally the `markdownthemewitiko_dot.sty` \LaTeX package:

```

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

```

```
3262 \newif\ifmarkdownLaTeXLoaded
3263 \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.

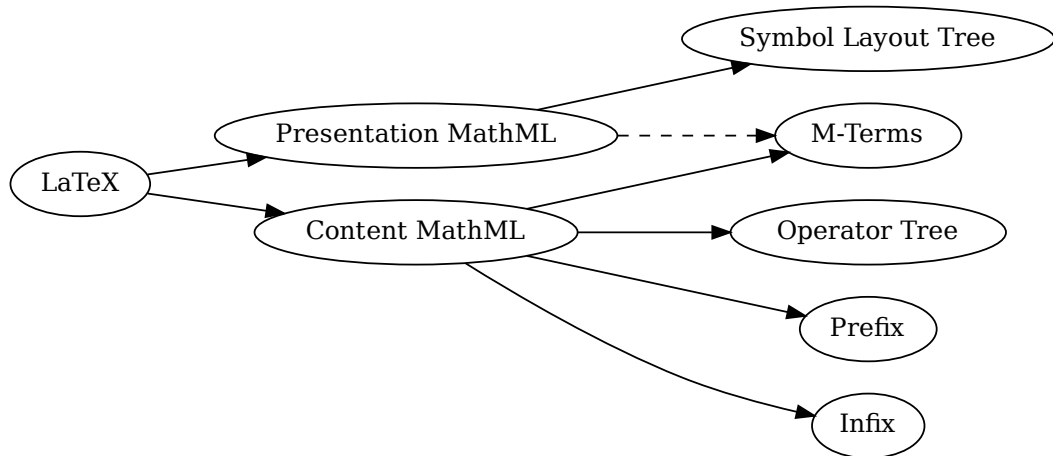


Figure 4: Various formats of mathematical formulae

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

```

3264 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%

```

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

```

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

```

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

```
3265 \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.

```

3266 \AtEndOfPackage{
3267   \markdownLaTeXLoadedtrue
At the end of the  $\LaTeX$  module, we load the witiko/markdown/defaults  $\LaTeX$  theme (see Section 2.2.3) with the default definitions for token renderer prototypes unless the option noDefaults has been enabled (see Section 2.2.2.3).
3268   \markdownIfOption{noDefaults}{}{
3269     \markdownSetup{theme=witiko/markdown/defaults}
3270   }
3271 }
3272 \ProvidesPackage{markdownthemewitiko_markdown_defaults}[2024/01/03]%

```

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

2.4 ConTeXt Interface

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

```
3273 \ExplSyntaxOn
3274 \tl_const:Nn \c_@@_option_layer_context_tl { context }
3275 \cs_generate_variant:Nn
3276   \tl_const:Nn
3277   { NV }
3278 \tl_if_exist:NF
3279   \c_@@_top_layer_tl
3280   {
3281     \tl_const:NV
3282       \c_@@_top_layer_tl
3283       \c_@@_option_layer_context_tl
3284   }
3285 \ExplSyntaxOff
```

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

```
3286 \writestatus{loading}{ConTeXt User Module / markdown}%
3287 \startmodule[markdown]
3288 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
3289   \do#\do\^\do\_do\%do\~}%
3290 \input markdown/markdown
```

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

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

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

2.4.1 Typesetting Markdown

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

```
3291 \let\startmarkdown\relax
3292 \let\stopmarkdown\relax
3293 \let\inputmarkdown\relax
```

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

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

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

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

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

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

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

2.4.2 Options

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

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

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

3294 `\ExplSyntaxOn`

```

3295 \cs_new:Npn
3296   \setupmarkdown
3297   [ #1 ]
3298   {
3299     \@@_setup:n
3300     { #1 }
3301   }
3302 \ExplSyntaxOff

```

2.4.2.1 Generating Plain T_EX Option Macros and Key-Values

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

```

3303 \ExplSyntaxOn
3304 \cs_new:Nn \@@_caseless:N
3305   {
3306     \regex_replace_all:nnN
3307     { ([a-z])([A-Z]) }
3308     { \1 \c { str_lowercase:n } \cB{\ 2 \cE\} }
3309     #1
3310     \tl_set:Nx
3311     #1
3312     { #1 }
3313   }
3314 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }

```

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

```

3315 \str_if_eq:VVT
3316   \c_@@_top_layer_tl
3317   \c_@@_option_layer_context_tl
3318   {
3319     \@@_define_option_commands_and_keyvals:
3320     \@@_define_renderers:
3321     \@@_define_renderer_prototypes:
3322   }
3323 \ExplSyntaxOff

```

2.4.3 Themes

In Section 2.2.3, we described the concept of themes. In ConT_EXt, we expand on the concept of themes by allowing a theme to be a full-blown ConT_EXt module. Specifically, the key-values `theme=<theme name>` and `import=<theme name>` load

a ConT_EXt module named `t-markdowntheme<munged theme name>.tex` if it exists and a T_EX document named `markdowntheme<munged theme name>.tex` otherwise.

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

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

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

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

witiko/markdown/defaults A ConT_EXt 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.

```
3324 \startmodule[markdownthemewitiko_markdown_defaults]
3325 \unprotect
```

Please, see Section 3.4.2 for implementation details of the built-in ConT_EXt 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_EX *token renderers* is performed by the Lua layer. The plain T_EX layer provides default definitions for the token renderers. The L^AT_EX and ConT_EXt layers correct idiosyncrasies of the respective T_EX formats, and provide format-specific default definitions for the token renderers.

3.1 Lua Implementation

The Lua implementation implements `writer` and `reader` objects, which provide the conversion from markdown to plain T_EX, and `extensions` objects, which provide syntax extensions for the `writer` and `reader` objects.

The Lunamark Lua module implements writers for the conversion to various other formats, such as DocBook, Groff, or HTML. These were stripped from the module

and the remaining markdown reader and plain T_EX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```
3326 local upper, format, length =
3327   string.upper, string.format, string.len
3328 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, Cp, any =
3329   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
3330   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, lpeg.Cp, lpeg.P(1)
```

3.1.1 Utility Functions

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

```
3331 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.

```
3332 function util.err(msg, exit_code)
3333   io.stderr:write("markdown.lua: " .. msg .. "\n")
3334   os.exit(exit_code or 1)
3335 end
```

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

```
3336 function util.cache(dir, string, salt, transform, suffix)
3337   local digest = md5.sumhexa(string .. (salt or ""))
3338   local name = util.pathname(dir, digest .. suffix)
3339   local file = io.open(name, "r")
3340   if file == nil then -- If no cache entry exists, then create a new one.
3341     file = assert(io.open(name, "w"),
3342       [[Could not open file ]] .. name .. [[ for writing]])
3343     local result = string
3344     if transform ~= nil then
3345       result = transform(result)
3346     end
3347     assert(file:write(result))
3348     assert(file:close())
3349   end
3350   return name
3351 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.

```

3352 function util.cache_verbatim(dir, string)
3353   local name = util.cache(dir, string, nil, nil, ".verbatim")
3354   return name
3355 end

```

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

```

3356 function util.table_copy(t)
3357   local u = { }
3358   for k, v in pairs(t) do u[k] = v end
3359   return setmetatable(u, getmetatable(t))
3360 end

```

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

```

3361 function util.encode_json_string(s)
3362   s = s:gsub([[\\]], [[\\]])
3363   s = s:gsub([[\"]], [[\"]])
3364   return [[\"]] .. s .. [[\"]]
3365 end

```

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

```

3366 function util.expand_tabs_in_line(s, tabstop)
3367   local tab = tabstop or 4
3368   local corr = 0
3369   return (s:gsub("(\\t)", function(p)
3370     local sp = tab - (p - 1 + corr) % tab
3371     corr = corr - 1 + sp
3372     return string.rep(" ", sp)
3373   end))
3374 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.

```

3375 function util.walk(t, f)
3376   local typ = type(t)
3377   if typ == "string" then
3378     f(t)
3379   elseif typ == "table" then
3380     local i = 1
3381     local n
3382     n = t[i]
3383     while n do
3384       util.walk(n, f)
3385       i = i + 1

```

```

3386     n = t[i]
3387     end
3388     elseif typ == "function" then
3389         local ok, val = pcall(t)
3390         if ok then
3391             util.walk(val,f)
3392         end
3393     else
3394         f(tostring(t))
3395     end
3396 end

```

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

```

3397 function util.flatten(ary)
3398     local new = {}
3399     for _,v in ipairs(ary) do
3400         if type(v) == "table" then
3401             for _,w in ipairs(util.flatten(v)) do
3402                 new[#new + 1] = w
3403             end
3404         else
3405             new[#new + 1] = v
3406         end
3407     end
3408     return new
3409 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.

```

3410 function util.rope_to_string(rope)
3411     local buffer = {}
3412     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
3413     return table.concat(buffer)
3414 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.

```

3415 function util.rope_last(rope)
3416     if #rope == 0 then
3417         return nil
3418     else
3419         local l = rope[#rope]
3420         if type(l) == "table" then
3421             return util.rope_last(l)
3422         else
3423             return l
3424         end
3425     end

```

```

3425 end
3426 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 $1 \leq i \leq \#ary$.

```

3427 function util.intersperse(ary, x)
3428   local new = {}
3429   local l = #ary
3430   for i,v in ipairs(ary) do
3431     local n = #new
3432     new[n + 1] = v
3433     if i ~= l then
3434       new[n + 2] = x
3435     end
3436   end
3437   return new
3438 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 $1 \leq i \leq \#ary$.

```

3439 function util.map(ary, f)
3440   local new = {}
3441   for i,v in ipairs(ary) do
3442     new[i] = f(v)
3443   end
3444   return new
3445 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.

```

3446 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

3447   local char_escapes_list = ""
3448   for i,_ in pairs(char_escapes) do
3449     char_escapes_list = char_escapes_list .. i
3450   end

```

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

```

3451   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.

```
3452  if string_escapes then
3453      for k,v in pairs(string_escapes) do
3454          escapable = P(k) / v + escapable
3455      end
3456  end
```

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

```
3457  local escape_string = Cs((escapable + any)^0)
```

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

```
3458  return function(s)
3459      return lpeg.match(escape_string, s)
3460  end
3461 end
```

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

```
3462 function util.pathname(dir, file)
3463     if #dir == 0 then
3464         return file
3465     else
3466         return dir .. "/" .. file
3467     end
3468 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.

```
3469 local entities = {}
3470
3471 local character_entities = {
3472     ["Tab"] = 9,
3473     ["NewLine"] = 10,
3474     ["excl"] = 33,
3475     ["QUOT"] = 34,
3476     ["quot"] = 34,
3477     ["num"] = 35,
3478     ["dollar"] = 36,
3479     ["percent"] = 37,
```

```

3480 ["AMP"] = 38,
3481 ["amp"] = 38,
3482 ["apos"] = 39,
3483 ["lpar"] = 40,
3484 ["rpar"] = 41,
3485 ["ast"] = 42,
3486 ["midast"] = 42,
3487 ["plus"] = 43,
3488 ["comma"] = 44,
3489 ["period"] = 46,
3490 ["sol"] = 47,
3491 ["colon"] = 58,
3492 ["semi"] = 59,
3493 ["LT"] = 60,
3494 ["lt"] = 60,
3495 ["nvlt"] = {60, 8402},
3496 ["bne"] = {61, 8421},
3497 ["equals"] = 61,
3498 ["GT"] = 62,
3499 ["gt"] = 62,
3500 ["nvgt"] = {62, 8402},
3501 ["quest"] = 63,
3502 ["commat"] = 64,
3503 ["lbrack"] = 91,
3504 ["lsqb"] = 91,
3505 ["bsol"] = 92,
3506 ["rbrack"] = 93,
3507 ["rsqb"] = 93,
3508 ["Hat"] = 94,
3509 ["UnderBar"] = 95,
3510 ["lowbar"] = 95,
3511 ["DiacriticalGrave"] = 96,
3512 ["grave"] = 96,
3513 ["fjlig"] = {102, 106},
3514 ["lbrace"] = 123,
3515 ["lcub"] = 123,
3516 ["VerticalLine"] = 124,
3517 ["verbar"] = 124,
3518 ["vert"] = 124,
3519 ["rbrace"] = 125,
3520 ["rcub"] = 125,
3521 ["NonBreakingSpace"] = 160,
3522 ["nbsp"] = 160,
3523 ["iexcl"] = 161,
3524 ["cent"] = 162,
3525 ["pound"] = 163,
3526 ["curren"] = 164,

```

3527 ["yen"] = 165,
3528 ["brvbar"] = 166,
3529 ["sect"] = 167,
3530 ["Dot"] = 168,
3531 ["DoubleDot"] = 168,
3532 ["die"] = 168,
3533 ["uml"] = 168,
3534 ["COPY"] = 169,
3535 ["copy"] = 169,
3536 ["ordf"] = 170,
3537 ["laquo"] = 171,
3538 ["not"] = 172,
3539 ["shy"] = 173,
3540 ["REG"] = 174,
3541 ["circledR"] = 174,
3542 ["reg"] = 174,
3543 ["macr"] = 175,
3544 ["strns"] = 175,
3545 ["deg"] = 176,
3546 ["PlusMinus"] = 177,
3547 ["plusmn"] = 177,
3548 ["pm"] = 177,
3549 ["sup2"] = 178,
3550 ["sup3"] = 179,
3551 ["DiacriticalAcute"] = 180,
3552 ["acute"] = 180,
3553 ["micro"] = 181,
3554 ["para"] = 182,
3555 ["CenterDot"] = 183,
3556 ["centerdot"] = 183,
3557 ["middot"] = 183,
3558 ["Cedilla"] = 184,
3559 ["cedil"] = 184,
3560 ["sup1"] = 185,
3561 ["ordm"] = 186,
3562 ["raquo"] = 187,
3563 ["frac14"] = 188,
3564 ["frac12"] = 189,
3565 ["half"] = 189,
3566 ["frac34"] = 190,
3567 ["iquest"] = 191,
3568 ["Agrave"] = 192,
3569 ["Aacute"] = 193,
3570 ["Acirc"] = 194,
3571 ["Atilde"] = 195,
3572 ["Auml"] = 196,
3573 ["Aring"] = 197,

3574 ["angst"] = 197,
3575 ["AElig"] = 198,
3576 ["Ccedil"] = 199,
3577 ["Egrave"] = 200,
3578 ["Eacute"] = 201,
3579 ["Ecirc"] = 202,
3580 ["Euml"] = 203,
3581 ["Igrave"] = 204,
3582 ["Iacute"] = 205,
3583 ["Icirc"] = 206,
3584 ["Iuml"] = 207,
3585 ["ETH"] = 208,
3586 ["Ntilde"] = 209,
3587 ["Ograve"] = 210,
3588 ["Oacute"] = 211,
3589 ["Ocirc"] = 212,
3590 ["Otilde"] = 213,
3591 ["Ouml"] = 214,
3592 ["times"] = 215,
3593 ["Oslash"] = 216,
3594 ["Ugrave"] = 217,
3595 ["Uacute"] = 218,
3596 ["Ucirc"] = 219,
3597 ["Uuml"] = 220,
3598 ["Yacute"] = 221,
3599 ["THORN"] = 222,
3600 ["szlig"] = 223,
3601 ["agrave"] = 224,
3602 ["aacute"] = 225,
3603 ["acirc"] = 226,
3604 ["atilde"] = 227,
3605 ["auml"] = 228,
3606 ["aring"] = 229,
3607 ["aelig"] = 230,
3608 ["ccedil"] = 231,
3609 ["egrave"] = 232,
3610 ["eacute"] = 233,
3611 ["ecirc"] = 234,
3612 ["euml"] = 235,
3613 ["igrave"] = 236,
3614 ["iacute"] = 237,
3615 ["icirc"] = 238,
3616 ["iuml"] = 239,
3617 ["eth"] = 240,
3618 ["ntilde"] = 241,
3619 ["ograve"] = 242,
3620 ["oacute"] = 243,

3621 ["ocirc"] = 244,
3622 ["otilde"] = 245,
3623 ["ouml"] = 246,
3624 ["div"] = 247,
3625 ["divide"] = 247,
3626 ["oslash"] = 248,
3627 ["ugrave"] = 249,
3628 ["uacute"] = 250,
3629 ["ucirc"] = 251,
3630 ["uuml"] = 252,
3631 ["yacute"] = 253,
3632 ["thorn"] = 254,
3633 ["yuml"] = 255,
3634 ["Amacr"] = 256,
3635 ["amacr"] = 257,
3636 ["Abreve"] = 258,
3637 ["abreve"] = 259,
3638 ["Aogon"] = 260,
3639 ["aogon"] = 261,
3640 ["Cacute"] = 262,
3641 ["cacute"] = 263,
3642 ["Ccirc"] = 264,
3643 ["ccirc"] = 265,
3644 ["Cdot"] = 266,
3645 ["cdot"] = 267,
3646 ["Ccaron"] = 268,
3647 ["ccaron"] = 269,
3648 ["Dcaron"] = 270,
3649 ["dcaron"] = 271,
3650 ["Dstrok"] = 272,
3651 ["dstrok"] = 273,
3652 ["Emacr"] = 274,
3653 ["emacr"] = 275,
3654 ["Edot"] = 278,
3655 ["edot"] = 279,
3656 ["Eogon"] = 280,
3657 ["eogon"] = 281,
3658 ["Ecaron"] = 282,
3659 ["ecaron"] = 283,
3660 ["Gcirc"] = 284,
3661 ["gcirc"] = 285,
3662 ["Gbreve"] = 286,
3663 ["gbreve"] = 287,
3664 ["Gdot"] = 288,
3665 ["gdot"] = 289,
3666 ["Gcedil"] = 290,
3667 ["Hcirc"] = 292,

3668 ["hcirc"] = 293,
3669 ["Hstrokr"] = 294,
3670 ["hstrokr"] = 295,
3671 ["Itilde"] = 296,
3672 ["itilde"] = 297,
3673 ["Imacr"] = 298,
3674 ["imacr"] = 299,
3675 ["Iogon"] = 302,
3676 ["iogon"] = 303,
3677 ["Idot"] = 304,
3678 ["imath"] = 305,
3679 ["inodot"] = 305,
3680 ["IJlig"] = 306,
3681 ["ijlig"] = 307,
3682 ["Jcirc"] = 308,
3683 ["jcirc"] = 309,
3684 ["Kcedil"] = 310,
3685 ["kcedil"] = 311,
3686 ["kgreen"] = 312,
3687 ["Lacute"] = 313,
3688 ["lacute"] = 314,
3689 ["Lcedil"] = 315,
3690 ["lcedil"] = 316,
3691 ["Lcaron"] = 317,
3692 ["lcaron"] = 318,
3693 ["Lmidot"] = 319,
3694 ["lmidot"] = 320,
3695 ["Lstrokr"] = 321,
3696 ["lstrokr"] = 322,
3697 ["Nacute"] = 323,
3698 ["nacute"] = 324,
3699 ["Ncedil"] = 325,
3700 ["ncedil"] = 326,
3701 ["Ncaron"] = 327,
3702 ["ncaron"] = 328,
3703 ["napos"] = 329,
3704 ["ENG"] = 330,
3705 ["eng"] = 331,
3706 ["Omacr"] = 332,
3707 ["omacr"] = 333,
3708 ["Odblac"] = 336,
3709 ["odblac"] = 337,
3710 ["OElig"] = 338,
3711 ["oelig"] = 339,
3712 ["Racute"] = 340,
3713 ["racute"] = 341,
3714 ["Rcedil"] = 342,

3715 ["rcedil"] = 343,
3716 ["Rcaron"] = 344,
3717 ["rcaron"] = 345,
3718 ["Sacute"] = 346,
3719 ["sacute"] = 347,
3720 ["Scirc"] = 348,
3721 ["scirc"] = 349,
3722 ["Scedil"] = 350,
3723 ["scedil"] = 351,
3724 ["Scaron"] = 352,
3725 ["scaron"] = 353,
3726 ["Tcedil"] = 354,
3727 ["tcedil"] = 355,
3728 ["Tcaron"] = 356,
3729 ["tcaron"] = 357,
3730 ["Tstrok"] = 358,
3731 ["tstrok"] = 359,
3732 ["Utilde"] = 360,
3733 ["utilde"] = 361,
3734 ["Umacr"] = 362,
3735 ["umacr"] = 363,
3736 ["Ubreve"] = 364,
3737 ["ubreve"] = 365,
3738 ["Uring"] = 366,
3739 ["uring"] = 367,
3740 ["Udblac"] = 368,
3741 ["udblac"] = 369,
3742 ["Uogon"] = 370,
3743 ["uogon"] = 371,
3744 ["Wcirc"] = 372,
3745 ["wcirc"] = 373,
3746 ["Ycirc"] = 374,
3747 ["ycirc"] = 375,
3748 ["Yuml"] = 376,
3749 ["Zacute"] = 377,
3750 ["zacute"] = 378,
3751 ["Zdot"] = 379,
3752 ["zdot"] = 380,
3753 ["Zcaron"] = 381,
3754 ["zcaron"] = 382,
3755 ["fnof"] = 402,
3756 ["imped"] = 437,
3757 ["gacute"] = 501,
3758 ["jmath"] = 567,
3759 ["circ"] = 710,
3760 ["Hacek"] = 711,
3761 ["caron"] = 711,

3762 ["Breve"] = 728,
3763 ["breve"] = 728,
3764 ["DiacriticalDot"] = 729,
3765 ["dot"] = 729,
3766 ["ring"] = 730,
3767 ["ogon"] = 731,
3768 ["DiacriticalTilde"] = 732,
3769 ["tilde"] = 732,
3770 ["DiacriticalDoubleAcute"] = 733,
3771 ["dblac"] = 733,
3772 ["DownBreve"] = 785,
3773 ["Alpha"] = 913,
3774 ["Beta"] = 914,
3775 ["Gamma"] = 915,
3776 ["Delta"] = 916,
3777 ["Epsilon"] = 917,
3778 ["Zeta"] = 918,
3779 ["Eta"] = 919,
3780 ["Theta"] = 920,
3781 ["Iota"] = 921,
3782 ["Kappa"] = 922,
3783 ["Lambda"] = 923,
3784 ["Mu"] = 924,
3785 ["Nu"] = 925,
3786 ["Xi"] = 926,
3787 ["Omicron"] = 927,
3788 ["Pi"] = 928,
3789 ["Rho"] = 929,
3790 ["Sigma"] = 931,
3791 ["Tau"] = 932,
3792 ["Upsilon"] = 933,
3793 ["Phi"] = 934,
3794 ["Chi"] = 935,
3795 ["Psi"] = 936,
3796 ["Omega"] = 937,
3797 ["ohm"] = 937,
3798 ["alpha"] = 945,
3799 ["beta"] = 946,
3800 ["gamma"] = 947,
3801 ["delta"] = 948,
3802 ["epsi"] = 949,
3803 ["epsilon"] = 949,
3804 ["zeta"] = 950,
3805 ["eta"] = 951,
3806 ["theta"] = 952,
3807 ["iota"] = 953,
3808 ["kappa"] = 954,

3809 ["lambda"] = 955,
3810 ["mu"] = 956,
3811 ["nu"] = 957,
3812 ["xi"] = 958,
3813 ["omicron"] = 959,
3814 ["pi"] = 960,
3815 ["rho"] = 961,
3816 ["sigmaf"] = 962,
3817 ["sigmav"] = 962,
3818 ["varsigma"] = 962,
3819 ["sigma"] = 963,
3820 ["tau"] = 964,
3821 ["upsilon"] = 965,
3822 ["upsilon"] = 965,
3823 ["phi"] = 966,
3824 ["chi"] = 967,
3825 ["psi"] = 968,
3826 ["omega"] = 969,
3827 ["thetasym"] = 977,
3828 ["thetav"] = 977,
3829 ["vartheta"] = 977,
3830 ["Upsilon"] = 978,
3831 ["upsih"] = 978,
3832 ["phiv"] = 981,
3833 ["straightphi"] = 981,
3834 ["varphi"] = 981,
3835 ["piv"] = 982,
3836 ["varpi"] = 982,
3837 ["Gammad"] = 988,
3838 ["digamma"] = 989,
3839 ["gammad"] = 989,
3840 ["kappav"] = 1008,
3841 ["varkappa"] = 1008,
3842 ["rhov"] = 1009,
3843 ["varrho"] = 1009,
3844 ["epsiv"] = 1013,
3845 ["straightepsilon"] = 1013,
3846 ["varepsilon"] = 1013,
3847 ["backepsilon"] = 1014,
3848 ["bepsi"] = 1014,
3849 ["IOcy"] = 1025,
3850 ["DJcy"] = 1026,
3851 ["GJcy"] = 1027,
3852 ["Jukcy"] = 1028,
3853 ["DScy"] = 1029,
3854 ["Iukcy"] = 1030,
3855 ["YIcy"] = 1031,

3856 ["Jsercy"] = 1032,
3857 ["LJcy"] = 1033,
3858 ["NJcy"] = 1034,
3859 ["TSHcy"] = 1035,
3860 ["KJcy"] = 1036,
3861 ["Ubrcy"] = 1038,
3862 ["DZcy"] = 1039,
3863 ["Acy"] = 1040,
3864 ["Bcy"] = 1041,
3865 ["Vcy"] = 1042,
3866 ["Gcy"] = 1043,
3867 ["Dcy"] = 1044,
3868 ["IEcy"] = 1045,
3869 ["ZHcy"] = 1046,
3870 ["Zcy"] = 1047,
3871 ["Icy"] = 1048,
3872 ["Jcy"] = 1049,
3873 ["Kcy"] = 1050,
3874 ["Lcy"] = 1051,
3875 ["Mcy"] = 1052,
3876 ["Ncy"] = 1053,
3877 ["Ocy"] = 1054,
3878 ["Pcy"] = 1055,
3879 ["Rcy"] = 1056,
3880 ["Scy"] = 1057,
3881 ["Tcy"] = 1058,
3882 ["Ucy"] = 1059,
3883 ["Fcy"] = 1060,
3884 ["KHcy"] = 1061,
3885 ["TScy"] = 1062,
3886 ["CHcy"] = 1063,
3887 ["SHcy"] = 1064,
3888 ["SHCHcy"] = 1065,
3889 ["HARDcy"] = 1066,
3890 ["Ycy"] = 1067,
3891 ["SOFTcy"] = 1068,
3892 ["Ecy"] = 1069,
3893 ["YUcy"] = 1070,
3894 ["YAcy"] = 1071,
3895 ["acy"] = 1072,
3896 ["bcy"] = 1073,
3897 ["vcy"] = 1074,
3898 ["gcy"] = 1075,
3899 ["dcy"] = 1076,
3900 ["iecy"] = 1077,
3901 ["zhcy"] = 1078,
3902 ["zcy"] = 1079,

3903 ["icy"] = 1080,
3904 ["jcy"] = 1081,
3905 ["kcy"] = 1082,
3906 ["lcy"] = 1083,
3907 ["mcy"] = 1084,
3908 ["ncy"] = 1085,
3909 ["ocy"] = 1086,
3910 ["pcy"] = 1087,
3911 ["rcy"] = 1088,
3912 ["scy"] = 1089,
3913 ["tcy"] = 1090,
3914 ["ucy"] = 1091,
3915 ["fcy"] = 1092,
3916 ["khcy"] = 1093,
3917 ["tscy"] = 1094,
3918 ["chcy"] = 1095,
3919 ["shcy"] = 1096,
3920 ["shchcy"] = 1097,
3921 ["hardcy"] = 1098,
3922 ["ycy"] = 1099,
3923 ["softcy"] = 1100,
3924 ["ecy"] = 1101,
3925 ["yucy"] = 1102,
3926 ["yacy"] = 1103,
3927 ["iocy"] = 1105,
3928 ["djcy"] = 1106,
3929 ["gjcy"] = 1107,
3930 ["jukcy"] = 1108,
3931 ["dscy"] = 1109,
3932 ["iukcy"] = 1110,
3933 ["yicy"] = 1111,
3934 ["jsercy"] = 1112,
3935 ["ljcy"] = 1113,
3936 ["njcy"] = 1114,
3937 ["tshcy"] = 1115,
3938 ["kjcy"] = 1116,
3939 ["ubrscy"] = 1118,
3940 ["dzcy"] = 1119,
3941 ["ensp"] = 8194,
3942 ["emsp"] = 8195,
3943 ["emsp13"] = 8196,
3944 ["emsp14"] = 8197,
3945 ["numsp"] = 8199,
3946 ["puncsp"] = 8200,
3947 ["ThinSpace"] = 8201,
3948 ["thinsp"] = 8201,
3949 ["VeryThinSpace"] = 8202,

3950 ["hairsp"] = 8202,
3951 ["NegativeMediumSpace"] = 8203,
3952 ["NegativeThickSpace"] = 8203,
3953 ["NegativeThinSpace"] = 8203,
3954 ["NegativeVeryThinSpace"] = 8203,
3955 ["ZeroWidthSpace"] = 8203,
3956 ["zwnj"] = 8204,
3957 ["zwj"] = 8205,
3958 ["lrm"] = 8206,
3959 ["rlm"] = 8207,
3960 ["dash"] = 8208,
3961 ["hyphen"] = 8208,
3962 ["ndash"] = 8211,
3963 ["mdash"] = 8212,
3964 ["horbar"] = 8213,
3965 ["Verbar"] = 8214,
3966 ["Vert"] = 8214,
3967 ["OpenCurlyQuote"] = 8216,
3968 ["lsquo"] = 8216,
3969 ["CloseCurlyQuote"] = 8217,
3970 ["rsquo"] = 8217,
3971 ["rsquor"] = 8217,
3972 ["lsquor"] = 8218,
3973 ["sbquo"] = 8218,
3974 ["OpenCurlyDoubleQuote"] = 8220,
3975 ["ldquo"] = 8220,
3976 ["CloseCurlyDoubleQuote"] = 8221,
3977 ["rdquo"] = 8221,
3978 ["rdquor"] = 8221,
3979 ["bdquo"] = 8222,
3980 ["ldquor"] = 8222,
3981 ["dagger"] = 8224,
3982 ["Dagger"] = 8225,
3983 ["ddagger"] = 8225,
3984 ["bull"] = 8226,
3985 ["bullet"] = 8226,
3986 ["nldr"] = 8229,
3987 ["hellip"] = 8230,
3988 ["mldr"] = 8230,
3989 ["permil"] = 8240,
3990 ["pertenk"] = 8241,
3991 ["prime"] = 8242,
3992 ["Prime"] = 8243,
3993 ["tprime"] = 8244,
3994 ["backprime"] = 8245,
3995 ["bprime"] = 8245,
3996 ["lsaquo"] = 8249,

3997 ["rsaquo"] = 8250,
 3998 ["OverBar"] = 8254,
 3999 ["oline"] = 8254,
 4000 ["caret"] = 8257,
 4001 ["hybull"] = 8259,
 4002 ["frasl"] = 8260,
 4003 ["bsemi"] = 8271,
 4004 ["qprime"] = 8279,
 4005 ["MediumSpace"] = 8287,
 4006 ["ThickSpace"] = {8287, 8202},
 4007 ["NoBreak"] = 8288,
 4008 ["ApplyFunction"] = 8289,
 4009 ["af"] = 8289,
 4010 ["InvisibleTimes"] = 8290,
 4011 ["it"] = 8290,
 4012 ["InvisibleComma"] = 8291,
 4013 ["ic"] = 8291,
 4014 ["euro"] = 8364,
 4015 ["TripleDot"] = 8411,
 4016 ["tdot"] = 8411,
 4017 ["DotDot"] = 8412,
 4018 ["Copf"] = 8450,
 4019 ["complexes"] = 8450,
 4020 ["incare"] = 8453,
 4021 ["gscr"] = 8458,
 4022 ["HilbertSpace"] = 8459,
 4023 ["Hscr"] = 8459,
 4024 ["hamilt"] = 8459,
 4025 ["Hfr"] = 8460,
 4026 ["Poincareplane"] = 8460,
 4027 ["Hopf"] = 8461,
 4028 ["quaternions"] = 8461,
 4029 ["planckh"] = 8462,
 4030 ["hbar"] = 8463,
 4031 ["hslash"] = 8463,
 4032 ["planck"] = 8463,
 4033 ["plankv"] = 8463,
 4034 ["Iscr"] = 8464,
 4035 ["imagline"] = 8464,
 4036 ["Ifr"] = 8465,
 4037 ["Im"] = 8465,
 4038 ["image"] = 8465,
 4039 ["imagpart"] = 8465,
 4040 ["Laplacetrif"] = 8466,
 4041 ["Lscr"] = 8466,
 4042 ["lagran"] = 8466,
 4043 ["ell"] = 8467,

4044 ["Nopf"] = 8469,
4045 ["naturals"] = 8469,
4046 ["numero"] = 8470,
4047 ["copysr"] = 8471,
4048 ["weierp"] = 8472,
4049 ["wp"] = 8472,
4050 ["Popf"] = 8473,
4051 ["primes"] = 8473,
4052 ["Qopf"] = 8474,
4053 ["rationals"] = 8474,
4054 ["Rscr"] = 8475,
4055 ["realine"] = 8475,
4056 ["Re"] = 8476,
4057 ["Rfr"] = 8476,
4058 ["real"] = 8476,
4059 ["realpart"] = 8476,
4060 ["Ropf"] = 8477,
4061 ["reals"] = 8477,
4062 ["rx"] = 8478,
4063 ["TRADE"] = 8482,
4064 ["trade"] = 8482,
4065 ["Zopf"] = 8484,
4066 ["integers"] = 8484,
4067 ["mho"] = 8487,
4068 ["Zfr"] = 8488,
4069 ["zeetrf"] = 8488,
4070 ["iiota"] = 8489,
4071 ["Bernoullis"] = 8492,
4072 ["Bscr"] = 8492,
4073 ["bernou"] = 8492,
4074 ["Cayleys"] = 8493,
4075 ["Cfr"] = 8493,
4076 ["escr"] = 8495,
4077 ["Escr"] = 8496,
4078 ["expectation"] = 8496,
4079 ["Fouriertrf"] = 8497,
4080 ["Fscr"] = 8497,
4081 ["Mellintrf"] = 8499,
4082 ["Mscr"] = 8499,
4083 ["phmmat"] = 8499,
4084 ["order"] = 8500,
4085 ["orderof"] = 8500,
4086 ["oscr"] = 8500,
4087 ["alefsym"] = 8501,
4088 ["aleph"] = 8501,
4089 ["beth"] = 8502,
4090 ["gimel"] = 8503,

4091 ["daleth"] = 8504,
 4092 ["CapitalDifferentialD"] = 8517,
 4093 ["DD"] = 8517,
 4094 ["DifferentialD"] = 8518,
 4095 ["dd"] = 8518,
 4096 ["ExponentialE"] = 8519,
 4097 ["ee"] = 8519,
 4098 ["exponentiale"] = 8519,
 4099 ["ImaginaryI"] = 8520,
 4100 ["ii"] = 8520,
 4101 ["frac13"] = 8531,
 4102 ["frac23"] = 8532,
 4103 ["frac15"] = 8533,
 4104 ["frac25"] = 8534,
 4105 ["frac35"] = 8535,
 4106 ["frac45"] = 8536,
 4107 ["frac16"] = 8537,
 4108 ["frac56"] = 8538,
 4109 ["frac18"] = 8539,
 4110 ["frac38"] = 8540,
 4111 ["frac58"] = 8541,
 4112 ["frac78"] = 8542,
 4113 ["LeftArrow"] = 8592,
 4114 ["ShortLeftArrow"] = 8592,
 4115 ["larr"] = 8592,
 4116 ["leftarrow"] = 8592,
 4117 ["slarr"] = 8592,
 4118 ["ShortUpArrow"] = 8593,
 4119 ["UpArrow"] = 8593,
 4120 ["uarr"] = 8593,
 4121 ["uparrow"] = 8593,
 4122 ["RightArrow"] = 8594,
 4123 ["ShortRightArrow"] = 8594,
 4124 ["rarr"] = 8594,
 4125 ["rightarrow"] = 8594,
 4126 ["srarr"] = 8594,
 4127 ["DownArrow"] = 8595,
 4128 ["ShortDownArrow"] = 8595,
 4129 ["darr"] = 8595,
 4130 ["downarrow"] = 8595,
 4131 ["LeftRightArrow"] = 8596,
 4132 ["harr"] = 8596,
 4133 ["leftrightarrow"] = 8596,
 4134 ["UpDownArrow"] = 8597,
 4135 ["updownarrow"] = 8597,
 4136 ["varr"] = 8597,
 4137 ["UpperLeftArrow"] = 8598,

4138 ["nwarr"] = 8598,
 4139 ["nwarrow"] = 8598,
 4140 ["UpperRightArrow"] = 8599,
 4141 ["nearr"] = 8599,
 4142 ["nearrow"] = 8599,
 4143 ["LowerRightArrow"] = 8600,
 4144 ["searr"] = 8600,
 4145 ["searrow"] = 8600,
 4146 ["LowerLeftArrow"] = 8601,
 4147 ["swarr"] = 8601,
 4148 ["swarrow"] = 8601,
 4149 ["nlarr"] = 8602,
 4150 ["nleftarrow"] = 8602,
 4151 ["nrarr"] = 8603,
 4152 ["nrightarrow"] = 8603,
 4153 ["nrarrw"] = {8605, 824},
 4154 ["rarrw"] = 8605,
 4155 ["rightsquigarrow"] = 8605,
 4156 ["Larr"] = 8606,
 4157 ["twoheadleftarrow"] = 8606,
 4158 ["Uarr"] = 8607,
 4159 ["Rarr"] = 8608,
 4160 ["twoheadrightarrow"] = 8608,
 4161 ["Darr"] = 8609,
 4162 ["larrtl"] = 8610,
 4163 ["leftarrowtail"] = 8610,
 4164 ["rarrtl"] = 8611,
 4165 ["rightarrowtail"] = 8611,
 4166 ["LeftTeeArrow"] = 8612,
 4167 ["mapstoleft"] = 8612,
 4168 ["UpTeeArrow"] = 8613,
 4169 ["mapstoup"] = 8613,
 4170 ["RightTeeArrow"] = 8614,
 4171 ["map"] = 8614,
 4172 ["mapsto"] = 8614,
 4173 ["DownTeeArrow"] = 8615,
 4174 ["mapstodown"] = 8615,
 4175 ["hookleftarrow"] = 8617,
 4176 ["larrhk"] = 8617,
 4177 ["hookrightarrow"] = 8618,
 4178 ["rarrhk"] = 8618,
 4179 ["larrlp"] = 8619,
 4180 ["looparrowleft"] = 8619,
 4181 ["looparrowright"] = 8620,
 4182 ["rarrlp"] = 8620,
 4183 ["harrw"] = 8621,
 4184 ["leftrightsquigarrow"] = 8621,

4185 ["nharr"] = 8622,
 4186 ["nleftrightharrow"] = 8622,
 4187 ["Lsh"] = 8624,
 4188 ["lsh"] = 8624,
 4189 ["Rsh"] = 8625,
 4190 ["rsh"] = 8625,
 4191 ["ldsh"] = 8626,
 4192 ["rdsh"] = 8627,
 4193 ["crarr"] = 8629,
 4194 ["cularr"] = 8630,
 4195 ["curvearrowleft"] = 8630,
 4196 ["curarr"] = 8631,
 4197 ["curvearrowright"] = 8631,
 4198 ["circlearrowleft"] = 8634,
 4199 ["olarr"] = 8634,
 4200 ["circlearrowright"] = 8635,
 4201 ["orarr"] = 8635,
 4202 ["LeftVector"] = 8636,
 4203 ["leftharpoonup"] = 8636,
 4204 ["lharu"] = 8636,
 4205 ["DownLeftVector"] = 8637,
 4206 ["leftharpoondown"] = 8637,
 4207 ["lhard"] = 8637,
 4208 ["RightUpVector"] = 8638,
 4209 ["uharr"] = 8638,
 4210 ["upharpoonright"] = 8638,
 4211 ["LeftUpVector"] = 8639,
 4212 ["uharl"] = 8639,
 4213 ["upharpoonleft"] = 8639,
 4214 ["RightVector"] = 8640,
 4215 ["rharu"] = 8640,
 4216 ["rightharpoonup"] = 8640,
 4217 ["DownRightVector"] = 8641,
 4218 ["rhard"] = 8641,
 4219 ["rightharpoondown"] = 8641,
 4220 ["RightDownVector"] = 8642,
 4221 ["dharr"] = 8642,
 4222 ["downharpoonright"] = 8642,
 4223 ["LeftDownVector"] = 8643,
 4224 ["dharl"] = 8643,
 4225 ["downharpoonleft"] = 8643,
 4226 ["RightArrowLeftArrow"] = 8644,
 4227 ["rightleftarrows"] = 8644,
 4228 ["rlarr"] = 8644,
 4229 ["UpArrowDownArrow"] = 8645,
 4230 ["udarr"] = 8645,
 4231 ["LeftArrowRightArrow"] = 8646,

4232 ["leftrightharpoons"] = 8646,
 4233 ["larr"] = 8646,
 4234 ["leftleftarrows"] = 8647,
 4235 ["llarr"] = 8647,
 4236 ["uparrows"] = 8648,
 4237 ["uarr"] = 8648,
 4238 ["rightrightarrows"] = 8649,
 4239 ["rrarr"] = 8649,
 4240 ["ddarr"] = 8650,
 4241 ["downdownarrows"] = 8650,
 4242 ["ReverseEquilibrium"] = 8651,
 4243 ["leftrightharpoons"] = 8651,
 4244 ["lrhar"] = 8651,
 4245 ["Equilibrium"] = 8652,
 4246 ["rightleftharpoons"] = 8652,
 4247 ["rlhar"] = 8652,
 4248 ["nLeftarrow"] = 8653,
 4249 ["nlArr"] = 8653,
 4250 ["nLeftrightarrow"] = 8654,
 4251 ["nhArr"] = 8654,
 4252 ["nrightarrow"] = 8655,
 4253 ["nrArr"] = 8655,
 4254 ["DoubleLeftArrow"] = 8656,
 4255 ["Leftarrow"] = 8656,
 4256 ["lArr"] = 8656,
 4257 ["DoubleUpArrow"] = 8657,
 4258 ["Uparrow"] = 8657,
 4259 ["uArr"] = 8657,
 4260 ["DoubleRightArrow"] = 8658,
 4261 ["Implies"] = 8658,
 4262 ["rightarrow"] = 8658,
 4263 ["rArr"] = 8658,
 4264 ["DoubleDownArrow"] = 8659,
 4265 ["Downarrow"] = 8659,
 4266 ["dArr"] = 8659,
 4267 ["DoubleLeftRightArrow"] = 8660,
 4268 ["Leftrightarrow"] = 8660,
 4269 ["hArr"] = 8660,
 4270 ["iff"] = 8660,
 4271 ["DoubleUpDownArrow"] = 8661,
 4272 ["Updownarrow"] = 8661,
 4273 ["vArr"] = 8661,
 4274 ["nwArr"] = 8662,
 4275 ["neArr"] = 8663,
 4276 ["seArr"] = 8664,
 4277 ["swArr"] = 8665,
 4278 ["Lleftarrow"] = 8666,

4279 ["lAarr"] = 8666,
 4280 ["Rightarrow"] = 8667,
 4281 ["rAarr"] = 8667,
 4282 ["zigrarr"] = 8669,
 4283 ["LeftArrowBar"] = 8676,
 4284 ["larrb"] = 8676,
 4285 ["RightArrowBar"] = 8677,
 4286 ["rarrb"] = 8677,
 4287 ["DownArrowUpArrow"] = 8693,
 4288 ["duarr"] = 8693,
 4289 ["loarr"] = 8701,
 4290 ["roarr"] = 8702,
 4291 ["hoarr"] = 8703,
 4292 ["ForAll"] = 8704,
 4293 ["forall"] = 8704,
 4294 ["comp"] = 8705,
 4295 ["complement"] = 8705,
 4296 ["PartialD"] = 8706,
 4297 ["npart"] = {8706, 824},
 4298 ["part"] = 8706,
 4299 ["Exists"] = 8707,
 4300 ["exist"] = 8707,
 4301 ["NotExists"] = 8708,
 4302 ["nexist"] = 8708,
 4303 ["nexists"] = 8708,
 4304 ["empty"] = 8709,
 4305 ["emptyset"] = 8709,
 4306 ["emptyv"] = 8709,
 4307 ["varnothing"] = 8709,
 4308 ["Del"] = 8711,
 4309 ["nabla"] = 8711,
 4310 ["Element"] = 8712,
 4311 ["in"] = 8712,
 4312 ["isin"] = 8712,
 4313 ["isinv"] = 8712,
 4314 ["NotElement"] = 8713,
 4315 ["notin"] = 8713,
 4316 ["notinva"] = 8713,
 4317 ["ReverseElement"] = 8715,
 4318 ["SuchThat"] = 8715,
 4319 ["ni"] = 8715,
 4320 ["niv"] = 8715,
 4321 ["NotReverseElement"] = 8716,
 4322 ["notni"] = 8716,
 4323 ["notniva"] = 8716,
 4324 ["Product"] = 8719,
 4325 ["prod"] = 8719,

4326 ["Coproduct"] = 8720,
 4327 ["coprod"] = 8720,
 4328 ["Sum"] = 8721,
 4329 ["sum"] = 8721,
 4330 ["minus"] = 8722,
 4331 ["MinusPlus"] = 8723,
 4332 ["mnplus"] = 8723,
 4333 ["mp"] = 8723,
 4334 ["dotplus"] = 8724,
 4335 ["plusdo"] = 8724,
 4336 ["Backslash"] = 8726,
 4337 ["setminus"] = 8726,
 4338 ["setmn"] = 8726,
 4339 ["smallsetminus"] = 8726,
 4340 ["ssetmn"] = 8726,
 4341 ["lowast"] = 8727,
 4342 ["SmallCircle"] = 8728,
 4343 ["compfn"] = 8728,
 4344 ["Sqrt"] = 8730,
 4345 ["radic"] = 8730,
 4346 ["Proportional"] = 8733,
 4347 ["prop"] = 8733,
 4348 ["propto"] = 8733,
 4349 ["varpropto"] = 8733,
 4350 ["vprop"] = 8733,
 4351 ["infin"] = 8734,
 4352 ["angrt"] = 8735,
 4353 ["ang"] = 8736,
 4354 ["angle"] = 8736,
 4355 ["nang"] = {8736, 8402},
 4356 ["angmsd"] = 8737,
 4357 ["measuredangle"] = 8737,
 4358 ["angsph"] = 8738,
 4359 ["VerticalBar"] = 8739,
 4360 ["mid"] = 8739,
 4361 ["shortmid"] = 8739,
 4362 ["smid"] = 8739,
 4363 ["NotVerticalBar"] = 8740,
 4364 ["nmid"] = 8740,
 4365 ["nshortmid"] = 8740,
 4366 ["nsmid"] = 8740,
 4367 ["DoubleVerticalBar"] = 8741,
 4368 ["par"] = 8741,
 4369 ["parallel"] = 8741,
 4370 ["shortparallel"] = 8741,
 4371 ["spar"] = 8741,
 4372 ["NotDoubleVerticalBar"] = 8742,

4373 ["npar"] = 8742,
4374 ["nparallel"] = 8742,
4375 ["nshortparallel"] = 8742,
4376 ["nspar"] = 8742,
4377 ["and"] = 8743,
4378 ["wedge"] = 8743,
4379 ["or"] = 8744,
4380 ["vee"] = 8744,
4381 ["cap"] = 8745,
4382 ["caps"] = {8745, 65024},
4383 ["cup"] = 8746,
4384 ["cups"] = {8746, 65024},
4385 ["Integral"] = 8747,
4386 ["int"] = 8747,
4387 ["Int"] = 8748,
4388 ["iiint"] = 8749,
4389 ["tint"] = 8749,
4390 ["ContourIntegral"] = 8750,
4391 ["conint"] = 8750,
4392 ["oint"] = 8750,
4393 ["Conint"] = 8751,
4394 ["DoubleContourIntegral"] = 8751,
4395 ["Cconint"] = 8752,
4396 ["cwint"] = 8753,
4397 ["ClockwiseContourIntegral"] = 8754,
4398 ["cwconint"] = 8754,
4399 ["CounterClockwiseContourIntegral"] = 8755,
4400 ["awconint"] = 8755,
4401 ["Therefore"] = 8756,
4402 ["there4"] = 8756,
4403 ["therefore"] = 8756,
4404 ["Because"] = 8757,
4405 ["because"] = 8757,
4406 ["because"] = 8757,
4407 ["ratio"] = 8758,
4408 ["Colon"] = 8759,
4409 ["Proportion"] = 8759,
4410 ["dotminus"] = 8760,
4411 ["minusd"] = 8760,
4412 ["mDDot"] = 8762,
4413 ["homtht"] = 8763,
4414 ["Tilde"] = 8764,
4415 ["nvsim"] = {8764, 8402},
4416 ["sim"] = 8764,
4417 ["thicksim"] = 8764,
4418 ["thksim"] = 8764,
4419 ["backsim"] = 8765,

4420 ["bsim"] = 8765,
4421 ["race"] = {8765, 817},
4422 ["ac"] = 8766,
4423 ["acE"] = {8766, 819},
4424 ["mstpos"] = 8766,
4425 ["acd"] = 8767,
4426 ["VerticalTilde"] = 8768,
4427 ["wr"] = 8768,
4428 ["wreath"] = 8768,
4429 ["NotTilde"] = 8769,
4430 ["nsim"] = 8769,
4431 ["EqualTilde"] = 8770,
4432 ["NotEqualTilde"] = {8770, 824},
4433 ["eqsim"] = 8770,
4434 ["esim"] = 8770,
4435 ["nesim"] = {8770, 824},
4436 ["TildeEqual"] = 8771,
4437 ["sime"] = 8771,
4438 ["simeq"] = 8771,
4439 ["NotTildeEqual"] = 8772,
4440 ["nsime"] = 8772,
4441 ["nsimeq"] = 8772,
4442 ["TildeFullEqual"] = 8773,
4443 ["cong"] = 8773,
4444 ["simne"] = 8774,
4445 ["NotTildeFullEqual"] = 8775,
4446 ["ncong"] = 8775,
4447 ["TildeTilde"] = 8776,
4448 ["ap"] = 8776,
4449 ["approx"] = 8776,
4450 ["asymp"] = 8776,
4451 ["thickapprox"] = 8776,
4452 ["thkap"] = 8776,
4453 ["NotTildeTilde"] = 8777,
4454 ["nap"] = 8777,
4455 ["napprox"] = 8777,
4456 ["ape"] = 8778,
4457 ["approxpeq"] = 8778,
4458 ["apid"] = 8779,
4459 ["napid"] = {8779, 824},
4460 ["backcong"] = 8780,
4461 ["bcong"] = 8780,
4462 ["CupCap"] = 8781,
4463 ["asympeq"] = 8781,
4464 ["nvap"] = {8781, 8402},
4465 ["Bumpeq"] = 8782,
4466 ["HumpDownHump"] = 8782,

4467 ["NotHumpDownHump"] = {8782, 824},
4468 ["bump"] = 8782,
4469 ["nbump"] = {8782, 824},
4470 ["HumpEqual"] = 8783,
4471 ["NotHumpEqual"] = {8783, 824},
4472 ["bumpe"] = 8783,
4473 ["bumpeq"] = 8783,
4474 ["nbumpe"] = {8783, 824},
4475 ["DotEqual"] = 8784,
4476 ["doteq"] = 8784,
4477 ["esdot"] = 8784,
4478 ["nedot"] = {8784, 824},
4479 ["doteqdot"] = 8785,
4480 ["eDot"] = 8785,
4481 ["efDot"] = 8786,
4482 ["fallingdotseq"] = 8786,
4483 ["erDot"] = 8787,
4484 ["risingdotseq"] = 8787,
4485 ["Assign"] = 8788,
4486 ["colone"] = 8788,
4487 ["coloneq"] = 8788,
4488 ["ecolon"] = 8789,
4489 ["eqcolon"] = 8789,
4490 ["ecir"] = 8790,
4491 ["eqcirc"] = 8790,
4492 ["circeq"] = 8791,
4493 ["cire"] = 8791,
4494 ["wedgeq"] = 8793,
4495 ["veeeq"] = 8794,
4496 ["triangleq"] = 8796,
4497 ["trie"] = 8796,
4498 ["equest"] = 8799,
4499 ["questeq"] = 8799,
4500 ["NotEqual"] = 8800,
4501 ["ne"] = 8800,
4502 ["Congruent"] = 8801,
4503 ["bnequiv"] = {8801, 8421},
4504 ["equiv"] = 8801,
4505 ["NotCongruent"] = 8802,
4506 ["nequiv"] = 8802,
4507 ["le"] = 8804,
4508 ["leq"] = 8804,
4509 ["nvle"] = {8804, 8402},
4510 ["GreaterEqual"] = 8805,
4511 ["ge"] = 8805,
4512 ["geq"] = 8805,
4513 ["nvge"] = {8805, 8402},

4514 ["LessFullEqual"] = 8806,
4515 ["lE"] = 8806,
4516 ["leqq"] = 8806,
4517 ["nlE"] = {8806, 824},
4518 ["nleqq"] = {8806, 824},
4519 ["GreaterFullEqual"] = 8807,
4520 ["NotGreaterFullEqual"] = {8807, 824},
4521 ["gE"] = 8807,
4522 ["geqq"] = 8807,
4523 ["ngE"] = {8807, 824},
4524 ["ngeqq"] = {8807, 824},
4525 ["lnE"] = 8808,
4526 ["lneqq"] = 8808,
4527 ["lvertneqq"] = {8808, 65024},
4528 ["lvnE"] = {8808, 65024},
4529 ["gnE"] = 8809,
4530 ["gneqq"] = 8809,
4531 ["gvertneqq"] = {8809, 65024},
4532 ["gvnE"] = {8809, 65024},
4533 ["Lt"] = 8810,
4534 ["NestedLessLess"] = 8810,
4535 ["NotLessLess"] = {8810, 824},
4536 ["ll"] = 8810,
4537 ["nLt"] = {8810, 8402},
4538 ["nLtv"] = {8810, 824},
4539 ["Gt"] = 8811,
4540 ["NestedGreaterGreater"] = 8811,
4541 ["NotGreaterGreater"] = {8811, 824},
4542 ["gg"] = 8811,
4543 ["nGt"] = {8811, 8402},
4544 ["nGtv"] = {8811, 824},
4545 ["between"] = 8812,
4546 ["twixt"] = 8812,
4547 ["NotCupCap"] = 8813,
4548 ["NotLess"] = 8814,
4549 ["nless"] = 8814,
4550 ["nlt"] = 8814,
4551 ["NotGreater"] = 8815,
4552 ["ngt"] = 8815,
4553 ["ngtr"] = 8815,
4554 ["NotLessEqual"] = 8816,
4555 ["nle"] = 8816,
4556 ["nleq"] = 8816,
4557 ["NotGreaterEqual"] = 8817,
4558 ["nge"] = 8817,
4559 ["ngeq"] = 8817,
4560 ["LessTilde"] = 8818,

4561 ["lesssim"] = 8818,
4562 ["lsim"] = 8818,
4563 ["GreaterTilde"] = 8819,
4564 ["gsim"] = 8819,
4565 ["gtrsim"] = 8819,
4566 ["NotLessTilde"] = 8820,
4567 ["nlsim"] = 8820,
4568 ["NotGreaterTilde"] = 8821,
4569 ["ngsim"] = 8821,
4570 ["LessGreater"] = 8822,
4571 ["lessgtr"] = 8822,
4572 ["lg"] = 8822,
4573 ["GreaterLess"] = 8823,
4574 ["gl"] = 8823,
4575 ["gtrless"] = 8823,
4576 ["NotLessGreater"] = 8824,
4577 ["ntlg"] = 8824,
4578 ["NotGreaterLess"] = 8825,
4579 ["ntgl"] = 8825,
4580 ["Precedes"] = 8826,
4581 ["pr"] = 8826,
4582 ["prec"] = 8826,
4583 ["Succeeds"] = 8827,
4584 ["sc"] = 8827,
4585 ["succ"] = 8827,
4586 ["PrecedesSlantEqual"] = 8828,
4587 ["prcue"] = 8828,
4588 ["preccurlyeq"] = 8828,
4589 ["SucceedsSlantEqual"] = 8829,
4590 ["sccue"] = 8829,
4591 ["succcurlyeq"] = 8829,
4592 ["PrecedesTilde"] = 8830,
4593 ["precsim"] = 8830,
4594 ["prsim"] = 8830,
4595 ["NotSucceedsTilde"] = {8831, 824},
4596 ["SucceedsTilde"] = 8831,
4597 ["scsim"] = 8831,
4598 ["succsim"] = 8831,
4599 ["NotPrecedes"] = 8832,
4600 ["npr"] = 8832,
4601 ["nprec"] = 8832,
4602 ["NotSucceeds"] = 8833,
4603 ["nsc"] = 8833,
4604 ["nsucc"] = 8833,
4605 ["NotSubset"] = {8834, 8402},
4606 ["nsubset"] = {8834, 8402},
4607 ["sub"] = 8834,

```

4608 ["subset"] = 8834,
4609 ["vnsub"] = {8834, 8402},
4610 ["NotSuperset"] = {8835, 8402},
4611 ["Superset"] = 8835,
4612 ["nsupset"] = {8835, 8402},
4613 ["sup"] = 8835,
4614 ["supset"] = 8835,
4615 ["vnsup"] = {8835, 8402},
4616 ["nsub"] = 8836,
4617 ["nsup"] = 8837,
4618 ["SubsetEqual"] = 8838,
4619 ["sube"] = 8838,
4620 ["subseteq"] = 8838,
4621 ["SupersetEqual"] = 8839,
4622 ["supe"] = 8839,
4623 ["supseteq"] = 8839,
4624 ["NotSubsetEqual"] = 8840,
4625 ["nsube"] = 8840,
4626 ["nsubseteq"] = 8840,
4627 ["NotSupersetEqual"] = 8841,
4628 ["nsupe"] = 8841,
4629 ["nsupseteq"] = 8841,
4630 ["subne"] = 8842,
4631 ["subsetneq"] = 8842,
4632 ["varsubsetneq"] = {8842, 65024},
4633 ["vsubne"] = {8842, 65024},
4634 ["supne"] = 8843,
4635 ["supsetneq"] = 8843,
4636 ["varsupsetneq"] = {8843, 65024},
4637 ["vsupne"] = {8843, 65024},
4638 ["cupdot"] = 8845,
4639 ["UnionPlus"] = 8846,
4640 ["uplus"] = 8846,
4641 ["NotSquareSubset"] = {8847, 824},
4642 ["SquareSubset"] = 8847,
4643 ["sqsub"] = 8847,
4644 ["sqsubset"] = 8847,
4645 ["NotSquareSuperset"] = {8848, 824},
4646 ["SquareSuperset"] = 8848,
4647 ["sqsup"] = 8848,
4648 ["sqsupset"] = 8848,
4649 ["SquareSubsetEqual"] = 8849,
4650 ["sqsube"] = 8849,
4651 ["sqsubseteq"] = 8849,
4652 ["SquareSupersetEqual"] = 8850,
4653 ["sqsupe"] = 8850,
4654 ["sqsupseteq"] = 8850,

```

4655 ["SquareIntersection"] = 8851,
 4656 ["sqcap"] = 8851,
 4657 ["sqcaps"] = {8851, 65024},
 4658 ["SquareUnion"] = 8852,
 4659 ["sqcup"] = 8852,
 4660 ["sqcups"] = {8852, 65024},
 4661 ["CirclePlus"] = 8853,
 4662 ["oplus"] = 8853,
 4663 ["CircleMinus"] = 8854,
 4664 ["ominus"] = 8854,
 4665 ["CircleTimes"] = 8855,
 4666 ["otimes"] = 8855,
 4667 ["osol"] = 8856,
 4668 ["CircleDot"] = 8857,
 4669 ["odot"] = 8857,
 4670 ["circledcirc"] = 8858,
 4671 ["ocir"] = 8858,
 4672 ["circledast"] = 8859,
 4673 ["oast"] = 8859,
 4674 ["circleddash"] = 8861,
 4675 ["odash"] = 8861,
 4676 ["boxplus"] = 8862,
 4677 ["plusb"] = 8862,
 4678 ["boxminus"] = 8863,
 4679 ["minusb"] = 8863,
 4680 ["boxtimes"] = 8864,
 4681 ["timesb"] = 8864,
 4682 ["dotsquare"] = 8865,
 4683 ["sdotb"] = 8865,
 4684 ["RightTee"] = 8866,
 4685 ["vdash"] = 8866,
 4686 ["LeftTee"] = 8867,
 4687 ["dashv"] = 8867,
 4688 ["DownTee"] = 8868,
 4689 ["top"] = 8868,
 4690 ["UpTee"] = 8869,
 4691 ["bot"] = 8869,
 4692 ["bottom"] = 8869,
 4693 ["perp"] = 8869,
 4694 ["models"] = 8871,
 4695 ["DoubleRightTee"] = 8872,
 4696 ["vDash"] = 8872,
 4697 ["Vdash"] = 8873,
 4698 ["Vvdash"] = 8874,
 4699 ["VDash"] = 8875,
 4700 ["nvdash"] = 8876,
 4701 ["nvDash"] = 8877,

4702 ["nVdash"] = 8878,
4703 ["nVDash"] = 8879,
4704 ["prurel"] = 8880,
4705 ["LeftTriangle"] = 8882,
4706 ["vartriangleleft"] = 8882,
4707 ["vltri"] = 8882,
4708 ["RightTriangle"] = 8883,
4709 ["vartriangleright"] = 8883,
4710 ["vrtri"] = 8883,
4711 ["LeftTriangleEqual"] = 8884,
4712 ["ltrie"] = 8884,
4713 ["nvltrie"] = {8884, 8402},
4714 ["trianglelefteq"] = 8884,
4715 ["RightTriangleEqual"] = 8885,
4716 ["nvrtrie"] = {8885, 8402},
4717 ["rtrie"] = 8885,
4718 ["trianglerighteq"] = 8885,
4719 ["origof"] = 8886,
4720 ["imof"] = 8887,
4721 ["multimap"] = 8888,
4722 ["mumap"] = 8888,
4723 ["hercon"] = 8889,
4724 ["intcal"] = 8890,
4725 ["intercal"] = 8890,
4726 ["veebar"] = 8891,
4727 ["barvee"] = 8893,
4728 ["angrtvb"] = 8894,
4729 ["ltri"] = 8895,
4730 ["Wedge"] = 8896,
4731 ["bigwedge"] = 8896,
4732 ["xwedge"] = 8896,
4733 ["Vee"] = 8897,
4734 ["bigvee"] = 8897,
4735 ["xvee"] = 8897,
4736 ["Intersection"] = 8898,
4737 ["bigcap"] = 8898,
4738 ["xcap"] = 8898,
4739 ["Union"] = 8899,
4740 ["bigcup"] = 8899,
4741 ["xcup"] = 8899,
4742 ["Diamond"] = 8900,
4743 ["diam"] = 8900,
4744 ["diamond"] = 8900,
4745 ["sdot"] = 8901,
4746 ["Star"] = 8902,
4747 ["sstarf"] = 8902,
4748 ["divideontimes"] = 8903,

4749 ["divonx"] = 8903,
 4750 ["bowtie"] = 8904,
 4751 ["ltimes"] = 8905,
 4752 ["rtimes"] = 8906,
 4753 ["leftthreetimes"] = 8907,
 4754 ["lthree"] = 8907,
 4755 ["rightthreetimes"] = 8908,
 4756 ["rthree"] = 8908,
 4757 ["backsimeq"] = 8909,
 4758 ["bsime"] = 8909,
 4759 ["curlyvee"] = 8910,
 4760 ["cuvee"] = 8910,
 4761 ["curlywedge"] = 8911,
 4762 ["cuwed"] = 8911,
 4763 ["Sub"] = 8912,
 4764 ["Subset"] = 8912,
 4765 ["Sup"] = 8913,
 4766 ["Supset"] = 8913,
 4767 ["Cap"] = 8914,
 4768 ["Cup"] = 8915,
 4769 ["fork"] = 8916,
 4770 ["pitchfork"] = 8916,
 4771 ["epar"] = 8917,
 4772 ["lessdot"] = 8918,
 4773 ["ltdot"] = 8918,
 4774 ["gtdot"] = 8919,
 4775 ["gtrdot"] = 8919,
 4776 ["Ll"] = 8920,
 4777 ["nLl"] = {8920, 824},
 4778 ["Gg"] = 8921,
 4779 ["ggg"] = 8921,
 4780 ["nGg"] = {8921, 824},
 4781 ["LessEqualGreater"] = 8922,
 4782 ["leg"] = 8922,
 4783 ["lesg"] = {8922, 65024},
 4784 ["lesseqgtr"] = 8922,
 4785 ["GreaterEqualLess"] = 8923,
 4786 ["gel"] = 8923,
 4787 ["gesl"] = {8923, 65024},
 4788 ["gtreqless"] = 8923,
 4789 ["cuepr"] = 8926,
 4790 ["curlyeqprec"] = 8926,
 4791 ["cuesc"] = 8927,
 4792 ["curlyeqsucc"] = 8927,
 4793 ["NotPrecedesSlantEqual"] = 8928,
 4794 ["nprcue"] = 8928,
 4795 ["NotSucceedsSlantEqual"] = 8929,

4796 ["nsccue"] = 8929,
 4797 ["NotSquareSubsetEqual"] = 8930,
 4798 ["nsqsube"] = 8930,
 4799 ["NotSquareSupersetEqual"] = 8931,
 4800 ["nsqsupe"] = 8931,
 4801 ["lnsim"] = 8934,
 4802 ["gnsim"] = 8935,
 4803 ["precnsim"] = 8936,
 4804 ["prnsim"] = 8936,
 4805 ["scnsim"] = 8937,
 4806 ["succnsim"] = 8937,
 4807 ["NotLeftTriangle"] = 8938,
 4808 ["nltri"] = 8938,
 4809 ["ntriangleleft"] = 8938,
 4810 ["NotRightTriangle"] = 8939,
 4811 ["nrtri"] = 8939,
 4812 ["ntriangleright"] = 8939,
 4813 ["NotLeftTriangleEqual"] = 8940,
 4814 ["nltrie"] = 8940,
 4815 ["ntrianglelefteq"] = 8940,
 4816 ["NotRightTriangleEqual"] = 8941,
 4817 ["nrtrie"] = 8941,
 4818 ["ntrianglerighteq"] = 8941,
 4819 ["vellip"] = 8942,
 4820 ["ctdot"] = 8943,
 4821 ["utdot"] = 8944,
 4822 ["dtdot"] = 8945,
 4823 ["disin"] = 8946,
 4824 ["isinsv"] = 8947,
 4825 ["isins"] = 8948,
 4826 ["isindot"] = 8949,
 4827 ["notinvc"] = {8949, 824},
 4828 ["notinvc"] = 8950,
 4829 ["notinvb"] = 8951,
 4830 ["isinE"] = 8953,
 4831 ["notinE"] = {8953, 824},
 4832 ["nisd"] = 8954,
 4833 ["xnis"] = 8955,
 4834 ["nis"] = 8956,
 4835 ["notnivc"] = 8957,
 4836 ["notnivb"] = 8958,
 4837 ["barwed"] = 8965,
 4838 ["barwedge"] = 8965,
 4839 ["Barwed"] = 8966,
 4840 ["doublebarwedge"] = 8966,
 4841 ["LeftCeiling"] = 8968,
 4842 ["lceil"] = 8968,

4843 ["RightCeiling"] = 8969,
 4844 ["rceil"] = 8969,
 4845 ["LeftFloor"] = 8970,
 4846 ["lfloor"] = 8970,
 4847 ["RightFloor"] = 8971,
 4848 ["rfloor"] = 8971,
 4849 ["drcrop"] = 8972,
 4850 ["dlcrop"] = 8973,
 4851 ["urcrop"] = 8974,
 4852 ["ulcrop"] = 8975,
 4853 ["bnot"] = 8976,
 4854 ["proflines"] = 8978,
 4855 ["profsurf"] = 8979,
 4856 ["telrec"] = 8981,
 4857 ["target"] = 8982,
 4858 ["ulcorn"] = 8988,
 4859 ["ulcorner"] = 8988,
 4860 ["urcorn"] = 8989,
 4861 ["urcorner"] = 8989,
 4862 ["dlcorn"] = 8990,
 4863 ["llcorner"] = 8990,
 4864 ["drcorn"] = 8991,
 4865 ["lrcorn"] = 8991,
 4866 ["frown"] = 8994,
 4867 ["sfrown"] = 8994,
 4868 ["smile"] = 8995,
 4869 ["ssmile"] = 8995,
 4870 ["cylcty"] = 9005,
 4871 ["profalar"] = 9006,
 4872 ["topbot"] = 9014,
 4873 ["ovbar"] = 9021,
 4874 ["solbar"] = 9023,
 4875 ["angzarr"] = 9084,
 4876 ["lmoust"] = 9136,
 4877 ["lmoustache"] = 9136,
 4878 ["rmoust"] = 9137,
 4879 ["rmoustache"] = 9137,
 4880 ["OverBracket"] = 9140,
 4881 ["tbrk"] = 9140,
 4882 ["UnderBracket"] = 9141,
 4883 ["bbrk"] = 9141,
 4884 ["bbrktbrk"] = 9142,
 4885 ["OverParenthesis"] = 9180,
 4886 ["UnderParenthesis"] = 9181,
 4887 ["OverBrace"] = 9182,
 4888 ["UnderBrace"] = 9183,
 4889 ["trpezium"] = 9186,

4890 ["elinters"] = 9191,
4891 ["blank"] = 9251,
4892 ["circledS"] = 9416,
4893 ["oS"] = 9416,
4894 ["HorizontalLine"] = 9472,
4895 ["boxh"] = 9472,
4896 ["boxv"] = 9474,
4897 ["boxdr"] = 9484,
4898 ["boxdl"] = 9488,
4899 ["boxur"] = 9492,
4900 ["boxul"] = 9496,
4901 ["boxvr"] = 9500,
4902 ["boxvl"] = 9508,
4903 ["boxhd"] = 9516,
4904 ["boxhu"] = 9524,
4905 ["boxvh"] = 9532,
4906 ["boxH"] = 9552,
4907 ["boxV"] = 9553,
4908 ["boxdR"] = 9554,
4909 ["boxDr"] = 9555,
4910 ["boxDR"] = 9556,
4911 ["boxdL"] = 9557,
4912 ["boxDL"] = 9558,
4913 ["boxDL"] = 9559,
4914 ["boxuR"] = 9560,
4915 ["boxUr"] = 9561,
4916 ["boxUR"] = 9562,
4917 ["boxuL"] = 9563,
4918 ["boxUL"] = 9564,
4919 ["boxUL"] = 9565,
4920 ["boxvR"] = 9566,
4921 ["boxVr"] = 9567,
4922 ["boxVR"] = 9568,
4923 ["boxvL"] = 9569,
4924 ["boxVl"] = 9570,
4925 ["boxVL"] = 9571,
4926 ["boxHd"] = 9572,
4927 ["boxhD"] = 9573,
4928 ["boxHD"] = 9574,
4929 ["boxHu"] = 9575,
4930 ["boxhU"] = 9576,
4931 ["boxHU"] = 9577,
4932 ["boxvH"] = 9578,
4933 ["boxVh"] = 9579,
4934 ["boxVH"] = 9580,
4935 ["uhblk"] = 9600,
4936 ["lhblk"] = 9604,

4937 ["block"] = 9608,
4938 ["blk14"] = 9617,
4939 ["blk12"] = 9618,
4940 ["blk34"] = 9619,
4941 ["Square"] = 9633,
4942 ["squ"] = 9633,
4943 ["square"] = 9633,
4944 ["FilledVerySmallSquare"] = 9642,
4945 ["blacksquare"] = 9642,
4946 ["squarf"] = 9642,
4947 ["squf"] = 9642,
4948 ["EmptyVerySmallSquare"] = 9643,
4949 ["rect"] = 9645,
4950 ["marker"] = 9646,
4951 ["fltns"] = 9649,
4952 ["bigtriangleup"] = 9651,
4953 ["xutri"] = 9651,
4954 ["blacktriangle"] = 9652,
4955 ["utrif"] = 9652,
4956 ["triangle"] = 9653,
4957 ["utri"] = 9653,
4958 ["blacktriangleright"] = 9656,
4959 ["rtrif"] = 9656,
4960 ["rtri"] = 9657,
4961 ["triangleright"] = 9657,
4962 ["bigtriangledown"] = 9661,
4963 ["xdtri"] = 9661,
4964 ["blacktriangledown"] = 9662,
4965 ["dtrif"] = 9662,
4966 ["dtri"] = 9663,
4967 ["triangledown"] = 9663,
4968 ["blacktriangleleft"] = 9666,
4969 ["ltrif"] = 9666,
4970 ["ltri"] = 9667,
4971 ["triangleleft"] = 9667,
4972 ["loz"] = 9674,
4973 ["lozenge"] = 9674,
4974 ["cir"] = 9675,
4975 ["tridot"] = 9708,
4976 ["bigcirc"] = 9711,
4977 ["xcirc"] = 9711,
4978 ["ultri"] = 9720,
4979 ["urtri"] = 9721,
4980 ["lltri"] = 9722,
4981 ["EmptySmallSquare"] = 9723,
4982 ["FilledSmallSquare"] = 9724,
4983 ["bigstar"] = 9733,

4984 ["starf"] = 9733,
4985 ["star"] = 9734,
4986 ["phone"] = 9742,
4987 ["female"] = 9792,
4988 ["male"] = 9794,
4989 ["spades"] = 9824,
4990 ["spadesuit"] = 9824,
4991 ["clubs"] = 9827,
4992 ["clubsuit"] = 9827,
4993 ["hearts"] = 9829,
4994 ["heartsuit"] = 9829,
4995 ["diamondsuit"] = 9830,
4996 ["diams"] = 9830,
4997 ["sung"] = 9834,
4998 ["flat"] = 9837,
4999 ["natur"] = 9838,
5000 ["natural"] = 9838,
5001 ["sharp"] = 9839,
5002 ["check"] = 10003,
5003 ["checkmark"] = 10003,
5004 ["cross"] = 10007,
5005 ["malt"] = 10016,
5006 ["maltese"] = 10016,
5007 ["sext"] = 10038,
5008 ["VerticalSeparator"] = 10072,
5009 ["lbrk"] = 10098,
5010 ["rbrk"] = 10099,
5011 ["bsolhsub"] = 10184,
5012 ["suphsol"] = 10185,
5013 ["LeftDoubleBracket"] = 10214,
5014 ["lobrk"] = 10214,
5015 ["RightDoubleBracket"] = 10215,
5016 ["robrk"] = 10215,
5017 ["LeftAngleBracket"] = 10216,
5018 ["lang"] = 10216,
5019 ["langle"] = 10216,
5020 ["RightAngleBracket"] = 10217,
5021 ["rang"] = 10217,
5022 ["rangle"] = 10217,
5023 ["Lang"] = 10218,
5024 ["Rang"] = 10219,
5025 ["loang"] = 10220,
5026 ["roang"] = 10221,
5027 ["LongLeftArrow"] = 10229,
5028 ["longleftarrow"] = 10229,
5029 ["xlarr"] = 10229,
5030 ["LongRightArrow"] = 10230,

5031 ["longrightarrow"] = 10230,
 5032 ["xrarr"] = 10230,
 5033 ["LongLeftRightArrow"] = 10231,
 5034 ["longleftrightarrow"] = 10231,
 5035 ["xharr"] = 10231,
 5036 ["DoubleLongLeftArrow"] = 10232,
 5037 ["Longleftarrow"] = 10232,
 5038 ["xlArr"] = 10232,
 5039 ["DoubleLongRightArrow"] = 10233,
 5040 ["Longrightarrow"] = 10233,
 5041 ["xrArr"] = 10233,
 5042 ["DoubleLongLeftRightArrow"] = 10234,
 5043 ["Longlefttrirightarrow"] = 10234,
 5044 ["xhArr"] = 10234,
 5045 ["longmapsto"] = 10236,
 5046 ["xmap"] = 10236,
 5047 ["dzigrarr"] = 10239,
 5048 ["nvlArr"] = 10498,
 5049 ["nvrArr"] = 10499,
 5050 ["nvHarr"] = 10500,
 5051 ["Map"] = 10501,
 5052 ["lbarr"] = 10508,
 5053 ["bkarow"] = 10509,
 5054 ["rbarr"] = 10509,
 5055 ["lBarr"] = 10510,
 5056 ["dbkarow"] = 10511,
 5057 ["rBarr"] = 10511,
 5058 ["RBarr"] = 10512,
 5059 ["drbkarow"] = 10512,
 5060 ["DDotrahd"] = 10513,
 5061 ["UpArrowBar"] = 10514,
 5062 ["DownArrowBar"] = 10515,
 5063 ["Rarrtl"] = 10518,
 5064 ["latail"] = 10521,
 5065 ["ratail"] = 10522,
 5066 ["lAtail"] = 10523,
 5067 ["rAtail"] = 10524,
 5068 ["larrfs"] = 10525,
 5069 ["rarrfs"] = 10526,
 5070 ["larrbfs"] = 10527,
 5071 ["rarrbfs"] = 10528,
 5072 ["nwarhk"] = 10531,
 5073 ["nearhk"] = 10532,
 5074 ["hksearrow"] = 10533,
 5075 ["searhk"] = 10533,
 5076 ["hkswarrow"] = 10534,
 5077 ["swarhk"] = 10534,

5078 ["nwnear"] = 10535,
5079 ["nesear"] = 10536,
5080 ["toea"] = 10536,
5081 ["seswar"] = 10537,
5082 ["tosa"] = 10537,
5083 ["swnwar"] = 10538,
5084 ["nrarrc"] = {10547, 824},
5085 ["rarrc"] = 10547,
5086 ["cudarr"] = 10549,
5087 ["ldca"] = 10550,
5088 ["rdca"] = 10551,
5089 ["cudarrl"] = 10552,
5090 ["larrpl"] = 10553,
5091 ["curarrm"] = 10556,
5092 ["cularrp"] = 10557,
5093 ["rarrpl"] = 10565,
5094 ["harrcir"] = 10568,
5095 ["Uarrocir"] = 10569,
5096 ["lurdshar"] = 10570,
5097 ["ldrushar"] = 10571,
5098 ["LeftRightVector"] = 10574,
5099 ["RightUpDownVector"] = 10575,
5100 ["DownLeftRightVector"] = 10576,
5101 ["LeftUpDownVector"] = 10577,
5102 ["LeftVectorBar"] = 10578,
5103 ["RightVectorBar"] = 10579,
5104 ["RightUpVectorBar"] = 10580,
5105 ["RightDownVectorBar"] = 10581,
5106 ["DownLeftVectorBar"] = 10582,
5107 ["DownRightVectorBar"] = 10583,
5108 ["LeftUpVectorBar"] = 10584,
5109 ["LeftDownVectorBar"] = 10585,
5110 ["LeftTeeVector"] = 10586,
5111 ["RightTeeVector"] = 10587,
5112 ["RightUpTeeVector"] = 10588,
5113 ["RightDownTeeVector"] = 10589,
5114 ["DownLeftTeeVector"] = 10590,
5115 ["DownRightTeeVector"] = 10591,
5116 ["LeftUpTeeVector"] = 10592,
5117 ["LeftDownTeeVector"] = 10593,
5118 ["lHar"] = 10594,
5119 ["uHar"] = 10595,
5120 ["rHar"] = 10596,
5121 ["dHar"] = 10597,
5122 ["luruhar"] = 10598,
5123 ["ldrdhar"] = 10599,
5124 ["ruluhar"] = 10600,

5125 ["rdldhar"] = 10601,
5126 ["lharul"] = 10602,
5127 ["llhard"] = 10603,
5128 ["rharul"] = 10604,
5129 ["lrhard"] = 10605,
5130 ["UpEquilibrium"] = 10606,
5131 ["udhar"] = 10606,
5132 ["ReverseUpEquilibrium"] = 10607,
5133 ["duhar"] = 10607,
5134 ["RoundImplies"] = 10608,
5135 ["erarr"] = 10609,
5136 ["simrarr"] = 10610,
5137 ["larrsim"] = 10611,
5138 ["rarrsim"] = 10612,
5139 ["rarrap"] = 10613,
5140 ["ltlarr"] = 10614,
5141 ["gtrarr"] = 10616,
5142 ["subrarr"] = 10617,
5143 ["suplarr"] = 10619,
5144 ["lfisht"] = 10620,
5145 ["rfisht"] = 10621,
5146 ["ufisht"] = 10622,
5147 ["dfisht"] = 10623,
5148 ["lopar"] = 10629,
5149 ["ropar"] = 10630,
5150 ["lbrke"] = 10635,
5151 ["rbrke"] = 10636,
5152 ["lbrkslu"] = 10637,
5153 ["rbrksld"] = 10638,
5154 ["lbrksld"] = 10639,
5155 ["rbrkslu"] = 10640,
5156 ["langd"] = 10641,
5157 ["rangd"] = 10642,
5158 ["lparlt"] = 10643,
5159 ["rpargt"] = 10644,
5160 ["gtlPar"] = 10645,
5161 ["ltrPar"] = 10646,
5162 ["vzigzag"] = 10650,
5163 ["vangrt"] = 10652,
5164 ["angrtvbd"] = 10653,
5165 ["ange"] = 10660,
5166 ["range"] = 10661,
5167 ["dwangle"] = 10662,
5168 ["uwangle"] = 10663,
5169 ["angmsdaa"] = 10664,
5170 ["angmsdab"] = 10665,
5171 ["angmsdac"] = 10666,

5172 ["angmsdad"] = 10667,
5173 ["angmsdae"] = 10668,
5174 ["angmsdaf"] = 10669,
5175 ["angmsdag"] = 10670,
5176 ["angmsdah"] = 10671,
5177 ["bemptyv"] = 10672,
5178 ["demptyv"] = 10673,
5179 ["cemptyv"] = 10674,
5180 ["raemptyv"] = 10675,
5181 ["laemptyv"] = 10676,
5182 ["ohbar"] = 10677,
5183 ["omid"] = 10678,
5184 ["opar"] = 10679,
5185 ["operp"] = 10681,
5186 ["olcross"] = 10683,
5187 ["odsold"] = 10684,
5188 ["olcir"] = 10686,
5189 ["ofcir"] = 10687,
5190 ["olt"] = 10688,
5191 ["ogt"] = 10689,
5192 ["cirscir"] = 10690,
5193 ["cirE"] = 10691,
5194 ["solb"] = 10692,
5195 ["bsolb"] = 10693,
5196 ["boxbox"] = 10697,
5197 ["trish"] = 10701,
5198 ["rtriltri"] = 10702,
5199 ["LeftTriangleBar"] = 10703,
5200 ["NotLeftTriangleBar"] = {10703, 824},
5201 ["NotRightTriangleBar"] = {10704, 824},
5202 ["RightTriangleBar"] = 10704,
5203 ["iinfin"] = 10716,
5204 ["infintie"] = 10717,
5205 ["nvinfin"] = 10718,
5206 ["eparsl"] = 10723,
5207 ["smeparsl"] = 10724,
5208 ["eqvparsl"] = 10725,
5209 ["blacklozenge"] = 10731,
5210 ["lozf"] = 10731,
5211 ["RuleDelayed"] = 10740,
5212 ["dsol"] = 10742,
5213 ["bigodot"] = 10752,
5214 ["xodot"] = 10752,
5215 ["bigoplus"] = 10753,
5216 ["xoplus"] = 10753,
5217 ["bigotimes"] = 10754,
5218 ["xotime"] = 10754,

5219 ["biguplus"] = 10756,
5220 ["xuplus"] = 10756,
5221 ["bigsqcup"] = 10758,
5222 ["xscup"] = 10758,
5223 ["iiiint"] = 10764,
5224 ["qint"] = 10764,
5225 ["fpartint"] = 10765,
5226 ["cirfnint"] = 10768,
5227 ["awint"] = 10769,
5228 ["rppolint"] = 10770,
5229 ["scpolint"] = 10771,
5230 ["npolint"] = 10772,
5231 ["pointint"] = 10773,
5232 ["quatint"] = 10774,
5233 ["intlarhk"] = 10775,
5234 ["pluscir"] = 10786,
5235 ["plusacir"] = 10787,
5236 ["simplus"] = 10788,
5237 ["plusdu"] = 10789,
5238 ["plussim"] = 10790,
5239 ["plustwo"] = 10791,
5240 ["mcomma"] = 10793,
5241 ["minusdu"] = 10794,
5242 ["loplus"] = 10797,
5243 ["roplus"] = 10798,
5244 ["Cross"] = 10799,
5245 ["timesd"] = 10800,
5246 ["timesbar"] = 10801,
5247 ["smashp"] = 10803,
5248 ["lotimes"] = 10804,
5249 ["rotimes"] = 10805,
5250 ["otimesas"] = 10806,
5251 ["Otimes"] = 10807,
5252 ["odiv"] = 10808,
5253 ["triplus"] = 10809,
5254 ["triminus"] = 10810,
5255 ["tritime"] = 10811,
5256 ["intprod"] = 10812,
5257 ["iproduct"] = 10812,
5258 ["amalg"] = 10815,
5259 ["capdot"] = 10816,
5260 ["ncup"] = 10818,
5261 ["ncap"] = 10819,
5262 ["capand"] = 10820,
5263 ["cupor"] = 10821,
5264 ["cupcap"] = 10822,
5265 ["capcup"] = 10823,

5266 ["cupbrcap"] = 10824,
5267 ["capbrcup"] = 10825,
5268 ["cupcup"] = 10826,
5269 ["capcap"] = 10827,
5270 ["ccups"] = 10828,
5271 ["ccaps"] = 10829,
5272 ["ccupssm"] = 10832,
5273 ["And"] = 10835,
5274 ["Or"] = 10836,
5275 ["andand"] = 10837,
5276 ["oror"] = 10838,
5277 ["orslope"] = 10839,
5278 ["andslope"] = 10840,
5279 ["andv"] = 10842,
5280 ["orv"] = 10843,
5281 ["andd"] = 10844,
5282 ["ord"] = 10845,
5283 ["wedbar"] = 10847,
5284 ["sdote"] = 10854,
5285 ["simdot"] = 10858,
5286 ["congdote"] = 10861,
5287 ["ncongdote"] = {10861, 824},
5288 ["easter"] = 10862,
5289 ["apacir"] = 10863,
5290 ["apE"] = 10864,
5291 ["napE"] = {10864, 824},
5292 ["eplus"] = 10865,
5293 ["pluse"] = 10866,
5294 ["Esim"] = 10867,
5295 ["Colone"] = 10868,
5296 ["Equal"] = 10869,
5297 ["ddotseq"] = 10871,
5298 ["eDDot"] = 10871,
5299 ["equivDD"] = 10872,
5300 ["ltcir"] = 10873,
5301 ["gtcir"] = 10874,
5302 ["ltquest"] = 10875,
5303 ["gtquest"] = 10876,
5304 ["LessSlantEqual"] = 10877,
5305 ["NotLessSlantEqual"] = {10877, 824},
5306 ["leqslant"] = 10877,
5307 ["les"] = 10877,
5308 ["nleqslant"] = {10877, 824},
5309 ["nles"] = {10877, 824},
5310 ["GreaterSlantEqual"] = 10878,
5311 ["NotGreaterSlantEqual"] = {10878, 824},
5312 ["geqslant"] = 10878,

5313 ["ges"] = 10878,
5314 ["ngeqslant"] = {10878, 824},
5315 ["nges"] = {10878, 824},
5316 ["lesdot"] = 10879,
5317 ["gesdot"] = 10880,
5318 ["lesdoto"] = 10881,
5319 ["gesdoto"] = 10882,
5320 ["lesdotor"] = 10883,
5321 ["gesdoto1"] = 10884,
5322 ["lap"] = 10885,
5323 ["lessapprox"] = 10885,
5324 ["gap"] = 10886,
5325 ["gtrapprox"] = 10886,
5326 ["lne"] = 10887,
5327 ["lneq"] = 10887,
5328 ["gne"] = 10888,
5329 ["gneq"] = 10888,
5330 ["lnap"] = 10889,
5331 ["lnapprox"] = 10889,
5332 ["gnap"] = 10890,
5333 ["gnapprox"] = 10890,
5334 ["lEg"] = 10891,
5335 ["lesseqqgtr"] = 10891,
5336 ["gEl"] = 10892,
5337 ["gtreqqless"] = 10892,
5338 ["lsime"] = 10893,
5339 ["gsime"] = 10894,
5340 ["lsimg"] = 10895,
5341 ["gsiml"] = 10896,
5342 ["lgE"] = 10897,
5343 ["glE"] = 10898,
5344 ["lesges"] = 10899,
5345 ["gesles"] = 10900,
5346 ["els"] = 10901,
5347 ["eqslantless"] = 10901,
5348 ["egs"] = 10902,
5349 ["eqslantgtr"] = 10902,
5350 ["elsdot"] = 10903,
5351 ["egsdot"] = 10904,
5352 ["el"] = 10905,
5353 ["eg"] = 10906,
5354 ["siml"] = 10909,
5355 ["sing"] = 10910,
5356 ["simLE"] = 10911,
5357 ["singE"] = 10912,
5358 ["LessLess"] = 10913,
5359 ["NotNestedLessLess"] = {10913, 824},

```

5360 ["GreaterGreater"] = 10914,
5361 ["NotNestedGreaterGreater"] = {10914, 824},
5362 ["glj"] = 10916,
5363 ["gla"] = 10917,
5364 ["ltcc"] = 10918,
5365 ["gtcc"] = 10919,
5366 ["lescc"] = 10920,
5367 ["gescc"] = 10921,
5368 ["smt"] = 10922,
5369 ["lat"] = 10923,
5370 ["smte"] = 10924,
5371 ["smtes"] = {10924, 65024},
5372 ["late"] = 10925,
5373 ["lates"] = {10925, 65024},
5374 ["bumpE"] = 10926,
5375 ["NotPrecedesEqual"] = {10927, 824},
5376 ["PrecedesEqual"] = 10927,
5377 ["npre"] = {10927, 824},
5378 ["npreceq"] = {10927, 824},
5379 ["pre"] = 10927,
5380 ["preceq"] = 10927,
5381 ["NotSucceedsEqual"] = {10928, 824},
5382 ["SucceedsEqual"] = 10928,
5383 ["nsce"] = {10928, 824},
5384 ["nsucceq"] = {10928, 824},
5385 ["sce"] = 10928,
5386 ["succeq"] = 10928,
5387 ["prE"] = 10931,
5388 ["scE"] = 10932,
5389 ["precneqq"] = 10933,
5390 ["prnE"] = 10933,
5391 ["scnE"] = 10934,
5392 ["sucneqq"] = 10934,
5393 ["prap"] = 10935,
5394 ["precapprox"] = 10935,
5395 ["scap"] = 10936,
5396 ["succapprox"] = 10936,
5397 ["precnapprox"] = 10937,
5398 ["prnap"] = 10937,
5399 ["scnap"] = 10938,
5400 ["succnapprox"] = 10938,
5401 ["Pr"] = 10939,
5402 ["Sc"] = 10940,
5403 ["subdot"] = 10941,
5404 ["supdot"] = 10942,
5405 ["subplus"] = 10943,
5406 ["supplus"] = 10944,

```

5407 ["submult"] = 10945,
 5408 ["supmult"] = 10946,
 5409 ["subedot"] = 10947,
 5410 ["supedot"] = 10948,
 5411 ["nsubE"] = {10949, 824},
 5412 ["nsubseteqq"] = {10949, 824},
 5413 ["subE"] = 10949,
 5414 ["subseteqq"] = 10949,
 5415 ["nsupE"] = {10950, 824},
 5416 ["nsupseteqq"] = {10950, 824},
 5417 ["supE"] = 10950,
 5418 ["supseteqq"] = 10950,
 5419 ["subsim"] = 10951,
 5420 ["supsim"] = 10952,
 5421 ["subnE"] = 10955,
 5422 ["subsetneqq"] = 10955,
 5423 ["varsubsetneqq"] = {10955, 65024},
 5424 ["vsubnE"] = {10955, 65024},
 5425 ["supnE"] = 10956,
 5426 ["supsetneqq"] = 10956,
 5427 ["varsupsetneqq"] = {10956, 65024},
 5428 ["vsupnE"] = {10956, 65024},
 5429 ["csub"] = 10959,
 5430 ["csup"] = 10960,
 5431 ["csube"] = 10961,
 5432 ["csupe"] = 10962,
 5433 ["subsup"] = 10963,
 5434 ["supsub"] = 10964,
 5435 ["subsub"] = 10965,
 5436 ["supsup"] = 10966,
 5437 ["suphsub"] = 10967,
 5438 ["supdsub"] = 10968,
 5439 ["forkv"] = 10969,
 5440 ["topfork"] = 10970,
 5441 ["mlcp"] = 10971,
 5442 ["Dashv"] = 10980,
 5443 ["DoubleLeftTee"] = 10980,
 5444 ["Vdashl"] = 10982,
 5445 ["Barv"] = 10983,
 5446 ["vBar"] = 10984,
 5447 ["vBarv"] = 10985,
 5448 ["Vbar"] = 10987,
 5449 ["Not"] = 10988,
 5450 ["bNot"] = 10989,
 5451 ["rnmid"] = 10990,
 5452 ["cirmid"] = 10991,
 5453 ["midcir"] = 10992,

5454 ["topcir"] = 10993,
5455 ["nhpar"] = 10994,
5456 ["parsim"] = 10995,
5457 ["nparsl"] = {11005, 8421},
5458 ["parsl"] = 11005,
5459 ["fflig"] = 64256,
5460 ["filig"] = 64257,
5461 ["fllig"] = 64258,
5462 ["ffilig"] = 64259,
5463 ["ffllig"] = 64260,
5464 ["Ascr"] = 119964,
5465 ["Cscr"] = 119966,
5466 ["Dscr"] = 119967,
5467 ["Gscr"] = 119970,
5468 ["Jscr"] = 119973,
5469 ["Kscr"] = 119974,
5470 ["Nscr"] = 119977,
5471 ["Oscr"] = 119978,
5472 ["Pscr"] = 119979,
5473 ["Qscr"] = 119980,
5474 ["Sscr"] = 119982,
5475 ["Tscr"] = 119983,
5476 ["Uscr"] = 119984,
5477 ["Vscr"] = 119985,
5478 ["Wscr"] = 119986,
5479 ["Xscr"] = 119987,
5480 ["Yscr"] = 119988,
5481 ["Zscr"] = 119989,
5482 ["ascr"] = 119990,
5483 ["bscr"] = 119991,
5484 ["cscr"] = 119992,
5485 ["dscr"] = 119993,
5486 ["fscr"] = 119995,
5487 ["hscr"] = 119997,
5488 ["iscr"] = 119998,
5489 ["jscr"] = 119999,
5490 ["kscr"] = 120000,
5491 ["lscr"] = 120001,
5492 ["mscr"] = 120002,
5493 ["nscr"] = 120003,
5494 ["pscr"] = 120005,
5495 ["qscr"] = 120006,
5496 ["rscr"] = 120007,
5497 ["sscr"] = 120008,
5498 ["tscr"] = 120009,
5499 ["uscr"] = 120010,
5500 ["vscr"] = 120011,

5501 ["wscr"] = 120012,
5502 ["xscr"] = 120013,
5503 ["yscr"] = 120014,
5504 ["zscr"] = 120015,
5505 ["Afr"] = 120068,
5506 ["Bfr"] = 120069,
5507 ["Dfr"] = 120071,
5508 ["Efr"] = 120072,
5509 ["Ffr"] = 120073,
5510 ["Gfr"] = 120074,
5511 ["Jfr"] = 120077,
5512 ["Kfr"] = 120078,
5513 ["Lfr"] = 120079,
5514 ["Mfr"] = 120080,
5515 ["Nfr"] = 120081,
5516 ["Ofr"] = 120082,
5517 ["Pfr"] = 120083,
5518 ["Qfr"] = 120084,
5519 ["Sfr"] = 120086,
5520 ["Tfr"] = 120087,
5521 ["Ufr"] = 120088,
5522 ["Vfr"] = 120089,
5523 ["Wfr"] = 120090,
5524 ["Xfr"] = 120091,
5525 ["Yfr"] = 120092,
5526 ["afr"] = 120094,
5527 ["bfr"] = 120095,
5528 ["cfr"] = 120096,
5529 ["dfr"] = 120097,
5530 ["efr"] = 120098,
5531 ["ffr"] = 120099,
5532 ["gfr"] = 120100,
5533 ["hfr"] = 120101,
5534 ["ifr"] = 120102,
5535 ["jfr"] = 120103,
5536 ["kfr"] = 120104,
5537 ["lfr"] = 120105,
5538 ["mfr"] = 120106,
5539 ["nfr"] = 120107,
5540 ["ofr"] = 120108,
5541 ["pfr"] = 120109,
5542 ["qfr"] = 120110,
5543 ["rfr"] = 120111,
5544 ["sfr"] = 120112,
5545 ["tfr"] = 120113,
5546 ["ufr"] = 120114,
5547 ["vfr"] = 120115,

5548 ["wfr"] = 120116,
5549 ["xfr"] = 120117,
5550 ["yfr"] = 120118,
5551 ["zfr"] = 120119,
5552 ["Aopf"] = 120120,
5553 ["Bopf"] = 120121,
5554 ["Dopf"] = 120123,
5555 ["Eopf"] = 120124,
5556 ["Fopf"] = 120125,
5557 ["Gopf"] = 120126,
5558 ["Iopf"] = 120128,
5559 ["Jopf"] = 120129,
5560 ["Kopf"] = 120130,
5561 ["Lopf"] = 120131,
5562 ["Mopf"] = 120132,
5563 ["Oopf"] = 120134,
5564 ["Sopf"] = 120138,
5565 ["Topf"] = 120139,
5566 ["Uopf"] = 120140,
5567 ["Vopf"] = 120141,
5568 ["Wopf"] = 120142,
5569 ["Xopf"] = 120143,
5570 ["Yopf"] = 120144,
5571 ["aopf"] = 120146,
5572 ["bopf"] = 120147,
5573 ["copf"] = 120148,
5574 ["dopf"] = 120149,
5575 ["eopf"] = 120150,
5576 ["fopf"] = 120151,
5577 ["gopf"] = 120152,
5578 ["hopf"] = 120153,
5579 ["iopf"] = 120154,
5580 ["jopf"] = 120155,
5581 ["kopf"] = 120156,
5582 ["lopf"] = 120157,
5583 ["mopf"] = 120158,
5584 ["nopf"] = 120159,
5585 ["oopf"] = 120160,
5586 ["popf"] = 120161,
5587 ["qopf"] = 120162,
5588 ["ropf"] = 120163,
5589 ["sopf"] = 120164,
5590 ["topf"] = 120165,
5591 ["uopf"] = 120166,
5592 ["vopf"] = 120167,
5593 ["wopf"] = 120168,
5594 ["xopf"] = 120169,

```

5595 ["yopf"] = 120170,
5596 ["zopf"] = 120171,
5597 }

```

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

```

5598 function entities.dec_entity(s)
5599   local n = tonumber(s)
5600   if n == nil then
5601     return "&#" .. s .. ";" -- fallback for unknown entities
5602   end
5603   return unicode.utf8.char(n)
5604 end

```

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

```

5605 function entities.hex_entity(s)
5606   local n = tonumber("0x"..s)
5607   if n == nil then
5608     return "&#x" .. s .. ";" -- fallback for unknown entities
5609   end
5610   return unicode.utf8.char(n)
5611 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.

```

5612 function entities.hex_entity_with_x_char(x, s)
5613   local n = tonumber("0x"..s)
5614   if n == nil then
5615     return "&#" .. x .. s .. ";" -- fallback for unknown entities
5616   end
5617   return unicode.utf8.char(n)
5618 end

```

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

```

5619 function entities.char_entity(s)
5620   local code_points = character_entities[s]
5621   if code_points == nil then
5622     return "&" .. s .. ";"
5623   end
5624   if type(code_points) ~= 'table' then
5625     code_points = {code_points}
5626   end
5627   local char_table = {}
5628   for _, code_point in ipairs(code_points) do
5629     table.insert(char_table, unicode.utf8.char(code_point))

```

```

5630     end
5631     return table.concat(char_table)
5632 end

```

3.1.3 Plain T_EX Writer

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

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

```

5633 M.writer = {}

```

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

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

```

5634 function M.writer.new(options)
5635     local self = {}

```

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

```

5636     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.

```

5637     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.

```

5638     local slice_specifiers = {}
5639     for specifier in options.slice:gmatch("[^%s]+") do
5640         table.insert(slice_specifiers, specifier)
5641     end
5642
5643     if #slice_specifiers == 2 then
5644         self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5645         local slice_begin_type = self.slice_begin:sub(1, 1)

```



```

5646     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5647         self.slice_begin = "^" .. self.slice_begin
5648     end
5649     local slice_end_type = self.slice_end:sub(1, 1)
5650     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5651         self.slice_end = "$" .. self.slice_end
5652     end
5653     elseif #slice_specifiers == 1 then
5654         self.slice_begin = "^" .. slice_specifiers[1]
5655         self.slice_end = "$" .. slice_specifiers[1]
5656     end
5657
5658     self.slice_begin_type = self.slice_begin:sub(1, 1)
5659     self.slice_begin_identifier = self.slice_begin:sub(2) or ""
5660     self.slice_end_type = self.slice_end:sub(1, 1)
5661     self.slice_end_identifier = self.slice_end:sub(2) or ""
5662
5663     if self.slice_begin == "^" and self.slice_end ~= "^" then
5664         self.is_writing = true
5665     else
5666         self.is_writing = false
5667     end

```

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

```
5668     self.suffix = ".tex"
```

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

```
5669     self.space = " "
```

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

```
5670     self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5671     function self.plain(s)
5672         return s
5673     end

```

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

```
5674     function self.paragraph(s)
5675         if not self.is_writing then return "" end
5676         return s
5677     end

```

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

```
5678     function self.pack(name)
5679         return [[\input{]] .. name .. [[]\relax]]
5680     end

```

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

```
5681 self.interblocksep_text = "\\markdownRendererInterblockSeparator\n{"
5682 function self.interblocksep()
5683   if not self.is_writing then return "" end
5684   return self.interblocksep_text
5685 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.

```
5686 self.paragraphsep_text = "\\markdownRendererParagraphSeparator\n{"
5687 function self.paragraphsep()
5688   if not self.is_writing then return "" end
5689   return self.paragraphsep_text
5690 end
```

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

```
5691 self.undosep_text = "\\markdownRendererUndoSeparator\n{"
5692 function self.undosep()
5693   if not self.is_writing then return "" end
5694   return self.undosep_text
5695 end
```

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

```
5696 self.soft_line_break = function()
5697   if self.flatten_inlines then return "\n" end
5698   return "\\markdownRendererSoftLineBreak\n{"
5699 end
```

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

```
5700 self.hard_line_break = function()
5701   if self.flatten_inlines then return "\n" end
5702   return "\\markdownRendererHardLineBreak\n{"
5703 end
```

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

```
5704 self.ellipsis = "\\markdownRendererEllipsis{"
```

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

```
5705 function self.thematic_break()
5706   if not self.is_writing then return "" end
5707   return "\\markdownRendererThematicBreak{"
5708 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.

```

5709 self.escaped_uri_chars = {
5710     [{""] = "\\markdownRendererLeftBrace{}" ,
5711     ["}"] = "\\markdownRendererRightBrace{}" ,
5712     ["\\"] = "\\markdownRendererBackslash{}" ,
5713 }
5714 self.escaped_minimal_strings = {
5715     ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex " ,
5716     [{"☒"}] = "\\markdownRendererTickedBox{}" ,
5717     [{"◻"}] = "\\markdownRendererHalfTickedBox{}" ,
5718     [{"□"}] = "\\markdownRendererUntickedBox{}" ,
5719     [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{}" ,
5720 }

```

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

```

5721 self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
5722 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.

```

5723 self.escaped_chars = {
5724     [{""] = "\\markdownRendererLeftBrace{}" ,
5725     ["}"] = "\\markdownRendererRightBrace{}" ,
5726     [{"%"}] = "\\markdownRendererPercentSign{}" ,
5727     ["\\"] = "\\markdownRendererBackslash{}" ,
5728     [{"#"}] = "\\markdownRendererHash{}" ,
5729     [{"$"}] = "\\markdownRendererDollarSign{}" ,
5730     [{"&"}] = "\\markdownRendererAmpersand{}" ,
5731     [{"_"}] = "\\markdownRendererUnderscore{}" ,
5732     [{"^"}] = "\\markdownRendererCircumflex{}" ,
5733     [{"~"}] = "\\markdownRendererTilde{}" ,
5734     [{"|"}] = "\\markdownRendererPipe{}" ,
5735     [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{}" ,
5736 }

```

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

```

5737 local function create_escaper(char_escapes, string_escapes)
5738     local escape = util.escaper(char_escapes, string_escapes)
5739     return function(s)
5740         if self.flatten_inlines then return s end
5741         return escape(s)
5742     end
5743 end
5744 local escape_typographic_text = create_escaper(
5745     self.escaped_chars, self.escaped_strings)

```

```

5746 local escape_programmatic_text = create_escaper(
5747     self.escaped_uri_chars, self.escaped_minimal_strings)
5748 local escape_minimal = create_escaper(
5749     {}, 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.

```

5750 self.escape = escape_typographic_text
5751 self.math = escape_minimal
5752 if options.hybrid then
5753     self.identifier = escape_minimal
5754     self.string = escape_minimal
5755     self.uri = escape_minimal
5756     self.infostring = escape_minimal
5757 else
5758     self.identifier = escape_programmatic_text
5759     self.string = escape_typographic_text
5760     self.uri = escape_programmatic_text
5761     self.infostring = escape_programmatic_text
5762 end

```

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

```

5763 function self.code(s, attributes)
5764     if self.flatten_inlines then return s end
5765     local buf = {}
5766     if attributes ~= nil then
5767         table.insert(buf,
5768             "\\markdownRendererCodeSpanAttributeContextBegin\n")
5769         table.insert(buf, self.attributes(attributes))
5770     end
5771     table.insert(buf,
5772         {"\\markdownRendererCodeSpan{" , self.escape(s), "}")})
5773     if attributes ~= nil then
5774         table.insert(buf,
5775             "\\markdownRendererCodeSpanAttributeContextEnd{")
5776     end
5777     return buf
5778 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.

```

5779 function self.link(lab, src, tit, attributes)
5780   if self.flatten_inlines then return lab end
5781   local buf = {}
5782   if attributes ~= nil then
5783     table.insert(buf,
5784       "\\markdownRendererLinkAttributeContextBegin\n")
5785     table.insert(buf, self.attributes(attributes))
5786   end
5787   table.insert(buf, {"\\markdownRendererLink{",lab,"}",
5788     "{",self.escape(src),"",
5789     "{",self.uri(src),"",
5790     "{",self.string(tit or ""),"}}"})
5791   if attributes ~= nil then
5792     table.insert(buf,
5793       "\\markdownRendererLinkAttributeContextEnd{")
5794   end
5795   return buf
5796 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.

```

5797 function self.image(lab, src, tit, attributes)
5798   if self.flatten_inlines then return lab end
5799   local buf = {}
5800   if attributes ~= nil then
5801     table.insert(buf,
5802       "\\markdownRendererImageAttributeContextBegin\n")
5803     table.insert(buf, self.attributes(attributes))
5804   end
5805   table.insert(buf, {"\\markdownRendererImage{",lab,"}",
5806     "{",self.string(src),"",
5807     "{",self.uri(src),"",
5808     "{",self.string(tit or ""),"}}"})
5809   if attributes ~= nil then
5810     table.insert(buf,
5811       "\\markdownRendererImageAttributeContextEnd{")
5812   end
5813   return buf
5814 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.

```

5815 function self.bulletlist(items,tight)
5816   if not self.is_writing then return "" end
5817   local buffer = {}
5818   for _,item in ipairs(items) do
5819     if item ~= "" then
5820       buffer[#buffer + 1] = self.bulletitem(item)
5821     end
5822   end
5823   local contents = util.intersperse(buffer,"\n")
5824   if tight and options.tightLists then
5825     return {"\markdownRendererUlBeginTight\n",contents,
5826           "\n\markdownRendererUlEndTight "}
5827   else
5828     return {"\markdownRendererUlBegin\n",contents,
5829           "\n\markdownRendererUlEnd "}
5830   end
5831 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.

```

5832 function self.bulletitem(s)
5833   return {"\markdownRendererUlItem ",s,
5834         "\markdownRendererUlItemEnd "}
5835 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.

```

5836 function self.orderedlist(items,tight,startnum)
5837   if not self.is_writing then return "" end
5838   local buffer = {}
5839   local num = startnum
5840   for _,item in ipairs(items) do
5841     if item ~= "" then
5842       buffer[#buffer + 1] = self.ordereditem(item,num)
5843     end
5844     if num ~= nil and item ~= "" then
5845       num = num + 1
5846     end
5847   end
5848   local contents = util.intersperse(buffer,"\n")
5849   if tight and options.tightLists then
5850     return {"\markdownRendererOlBeginTight\n",contents,
5851           "\n\markdownRendererOlEndTight "}
5852   else
5853     return {"\markdownRendererOlBegin\n",contents,

```

```

5854             "\n\\markdownRendererOlEnd "}
5855     end
5856 end

```

Define `writer->orderitem` 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.

```

5857 function self.orderitem(s,num)
5858     if num ~= nil then
5859         return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
5860             "\\markdownRendererOlItemEnd "}
5861     else
5862         return {"\\markdownRendererOlItem ",s,
5863             "\\markdownRendererOlItemEnd "}
5864     end
5865 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.

```

5866 function self.inline_html_comment(contents)
5867     if self.flatten_inlines then return contents end
5868     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
5869 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.

```

5870 function self.inline_html_tag(contents)
5871     if self.flatten_inlines then return contents end
5872     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}
5873 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.

```

5874 function self.block_html_element(s)
5875     if not self.is_writing then return "" end
5876     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
5877     return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
5878 end

```

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

```

5879 function self.emphasis(s)
5880     if self.flatten_inlines then return s end
5881     return {"\\markdownRendererEmphasis{" ,s,"}"}
5882 end

```

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

```
5883 function self.checkbox(f)
5884   if f == 1.0 then
5885     return "☒ "
5886   elseif f == 0.0 then
5887     return "☐ "
5888   else
5889     return "◻ "
5890   end
5891 end
```

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

```
5892 function self.strong(s)
5893   if self.flatten_inlines then return s end
5894   return {"\\markdownRendererStrongEmphasis{" ,s,""} }
5895 end
```

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

```
5896 function self.blockquote(s)
5897   if not self.is_writing then return "" end
5898   return {"\\markdownRendererBlockQuoteBegin\\n",s,
5899     "\\markdownRendererBlockQuoteEnd "}
5900 end
```

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

```
5901 function self.verbatim(s)
5902   if not self.is_writing then return "" end
5903   s = s:gsub("\\n$", "")
5904   local name = util.cache_verbatim(options.cacheDir, s)
5905   return {"\\markdownRendererInputVerbatim{" ,name,""} }
5906 end
```

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

```
5907 function self.document(d)
5908   local buf = {"\\markdownRendererDocumentBegin\\n", d}
5909
5910   -- pop all attributes
5911   table.insert(buf, self.pop_attributes())
5912
5913   table.insert(buf, "\\markdownRendererDocumentEnd")
5914
5915   return buf
5916 end
```


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

```
5917 local seen_identifiers = {}
5918 local key_value_regex = "(^[^= ]+)%s*=%s*(.*)"
5919 local function normalize_attributes(attributes, auto_identifiers)
5920   -- normalize attributes
5921   local normalized_attributes = {}
5922   local has_explicit_identifiers = false
5923   local key, value
5924   for _, attribute in ipairs(attributes or {}) do
5925     if attribute:sub(1, 1) == "#" then
5926       table.insert(normalized_attributes, attribute)
5927       has_explicit_identifiers = true
5928       seen_identifiers[attribute:sub(2)] = true
5929     elseif attribute:sub(1, 1) == "." then
5930       table.insert(normalized_attributes, attribute)
5931     else
5932       key, value = attribute:match(key_value_regex)
5933       if key:lower() == "id" then
5934         table.insert(normalized_attributes, "#" .. value)
5935       elseif key:lower() == "class" then
5936         local classes = {}
5937         for class in value:gmatch("%S+") do
5938           table.insert(classes, class)
5939         end
5940         table.sort(classes)
5941         for _, class in ipairs(classes) do
5942           table.insert(normalized_attributes, "." .. class)
5943         end
5944       else
5945         table.insert(normalized_attributes, attribute)
5946       end
5947     end
5948   end
5949
5950   -- if no explicit identifiers exist, add auto identifiers
5951   if not has_explicit_identifiers and auto_identifiers ~= nil then
5952     local seen_auto_identifiers = {}
5953     for _, auto_identifier in ipairs(auto_identifiers) do
5954       if seen_auto_identifiers[auto_identifier] == nil then
5955         seen_auto_identifiers[auto_identifier] = true
5956         if seen_identifiers[auto_identifier] == nil then
5957           seen_identifiers[auto_identifier] = true
5958           table.insert(normalized_attributes,
5959             "#" .. auto_identifier)
5960         end
5961       end
5962     end
5963     local auto_identifier_number = 1
```

```

5962         while true do
5963             local numbered_auto_identifier = auto_identifier .. "-"
5964                 .. auto_identifier_number
5965             if seen_identifiers[numbered_auto_identifier] == nil then
5966                 seen_identifiers[numbered_auto_identifier] = true
5967                 table.insert(normalized_attributes,
5968                     "#" .. numbered_auto_identifier)
5969                 break
5970             end
5971             auto_identifier_number = auto_identifier_number + 1
5972         end
5973     end
5974 end
5975 end
5976 end
5977
5978 -- sort and deduplicate normalized attributes
5979 table.sort(normalized_attributes)
5980 local seen_normalized_attributes = {}
5981 local deduplicated_normalized_attributes = {}
5982 for _, attribute in ipairs(normalized_attributes) do
5983     if seen_normalized_attributes[attribute] == nil then
5984         seen_normalized_attributes[attribute] = true
5985         table.insert(deduplicated_normalized_attributes, attribute)
5986     end
5987 end
5988
5989 return deduplicated_normalized_attributes
5990 end
5991
5992 function self.attributes(attributes, should_normalize_attributes)
5993     local normalized_attributes
5994     if should_normalize_attributes == false then
5995         normalized_attributes = attributes
5996     else
5997         normalized_attributes = normalize_attributes(attributes)
5998     end
5999
6000     local buf = {}
6001     local key, value
6002     for _, attribute in ipairs(normalized_attributes) do
6003         if attribute:sub(1, 1) == "#" then
6004             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
6005                 attribute:sub(2), "}"})
6006         elseif attribute:sub(1, 1) == "." then
6007             table.insert(buf, {"\\markdownRendererAttributeName{" ,
6008                 attribute:sub(2), "}"})

```

```

6009     else
6010         key, value = attribute:match(key_value_regex)
6011         table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
6012             key, "}{" , value, "}"}))
6013     end
6014 end
6015
6016 return buf
6017 end

```

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

```

6018 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`.

```

6019 self.attribute_type_levels = {}
6020 setmetatable(self.attribute_type_levels,
6021     { __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`.

```

6022 local function apply_attributes()
6023     local buf = {}
6024     for i = 1, #self.active_attributes do
6025         local start_output = self.active_attributes[i][3]
6026         if start_output ~= nil then
6027             table.insert(buf, start_output)
6028         end
6029     end
6030     return buf
6031 end
6032
6033 local function tear_down_attributes()
6034     local buf = {}
6035     for i = #self.active_attributes, 1, -1 do
6036         local end_output = self.active_attributes[i][4]
6037         if end_output ~= nil then
6038             table.insert(buf, end_output)
6039         end
6040     end
6041     return buf
6042 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.

```
6043 function self.push_attributes(attribute_type, attributes,
6044                               start_output, end_output)
6045   local attribute_type_level = self.attribute_type_levels[attribute_type]
6046   self.attribute_type_levels[attribute_type] = attribute_type_level + 1
6047
6048   -- index attributes in a hash table for easy lookup
6049   attributes = attributes or {}
6050   for i = 1, #attributes do
6051     attributes[attributes[i]] = true
6052   end
6053
6054   local buf = {}
6055   -- handle slicing
6056   if attributes["#" .. self.slice_end_identifier] ~= nil and
6057     self.slice_end_type == "^" then
6058     if self.is_writing then
6059       table.insert(buf, self.undosep())
6060       table.insert(buf, tear_down_attributes())
6061     end
6062     self.is_writing = false
6063   end
6064   if attributes["#" .. self.slice_begin_identifier] ~= nil and
6065     self.slice_begin_type == "^" then
6066     table.insert(buf, apply_attributes())
6067     self.is_writing = true
6068   end
6069   if self.is_writing and start_output ~= nil then
6070     table.insert(buf, start_output)
6071   end
6072   table.insert(self.active_attributes,
6073               {attribute_type, attributes,
6074                start_output, end_output})
6075   return buf
6076 end
6077
```

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](#)).

```
6078 function self.pop_attributes(attribute_type)
6079   local buf = {}
6080   -- pop attributes until we find attributes of correct type
```

```

6081 -- or until no attributes remain
6082 local current_attribute_type = false
6083 while current_attribute_type ~= attribute_type and
6084     #self.active_attributes > 0 do
6085     local attributes, _, end_output
6086     current_attribute_type, attributes, _, end_output = table.unpack(
6087         self.active_attributes[#self.active_attributes])
6088     local attribute_type_level = self.attribute_type_levels[current_attribute_type]
6089     self.attribute_type_levels[current_attribute_type] = attribute_type_level - 1
6090     if self.is_writing and end_output ~= nil then
6091         table.insert(buf, end_output)
6092     end
6093     table.remove(self.active_attributes, #self.active_attributes)
6094     -- handle slicing
6095     if attributes["#" .. self.slice_end_identifier] ~= nil
6096         and self.slice_end_type == "$" then
6097         if self.is_writing then
6098             table.insert(buf, self.undosep())
6099             table.insert(buf, tear_down_attributes())
6100         end
6101         self.is_writing = false
6102     end
6103     if attributes["#" .. self.slice_begin_identifier] ~= nil and
6104         self.slice_begin_type == "$" then
6105         self.is_writing = true
6106         table.insert(buf, apply_attributes())
6107     end
6108 end
6109 return buf
6110 end

```

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

```

6111 local function create_auto_identifier(s)
6112     local buffer = {}
6113     local prev_space = false
6114     local letter_found = false
6115     local normalized_s = s
6116     if not options.unicodeNormalization or options.unicodeNormalizationForm ~= "nfc"
6117         normalized_s = uni_algos.normalize.NFC(normalized_s)
6118     end
6119
6120     for _, code in utf8.codes(normalized_s) do
6121         local char = utf8.char(code)
6122
6123         -- Remove everything up to the first letter.
6124         if not letter_found then
6125             local is_letter = unicode.utf8.match(char, "%a")
6126             if is_letter then

```

```

6127         letter_found = true
6128     else
6129         goto continue
6130     end
6131 end
6132
6133 -- Remove all non-alphanumeric characters, except underscores, hyphens, and per
6134 if not unicode.utf8.match(char, "[%w_-%.%s]") then
6135     goto continue
6136 end
6137
6138 -- Replace all spaces and newlines with hyphens.
6139 if unicode.utf8.match(char, "[%s\n]") then
6140     char = "-"
6141     if prev_space then
6142         goto continue
6143     else
6144         prev_space = true
6145     end
6146 else
6147     -- Convert all alphabetic characters to lowercase.
6148     char = unicode.utf8.lower(char)
6149     prev_space = false
6150 end
6151
6152 table.insert(buffer, char)
6153
6154 ::continue::
6155 end
6156
6157 if prev_space then
6158     table.remove(buffer)
6159 end
6160
6161 local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6162 return identifier
6163 end

```

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

```

6164 local function create_gfm_auto_identifier(s)
6165     local buffer = {}
6166     local prev_space = false
6167     local letter_found = false
6168     local normalized_s = s
6169     if not options.unicodeNormalization or options.unicodeNormalizationForm ~= "nfc"
6170         normalized_s = uni_algos.normalize.NFC(normalized_s)

```

```

6171     end
6172
6173     for _, code in utf8.codes(normalized_s) do
6174         local char = utf8.char(code)
6175
6176         -- Remove everything up to the first non-space.
6177         if not letter_found then
6178             local is_letter = unicode.utf8.match(char, "%S")
6179             if is_letter then
6180                 letter_found = true
6181             else
6182                 goto continue
6183             end
6184         end
6185
6186         -- Remove all non-alphanumeric characters, except underscores and hyphens.
6187         if not unicode.utf8.match(char, "[%w_%-s]") then
6188             prev_space = false
6189             goto continue
6190         end
6191
6192         -- Replace all spaces and newlines with hyphens.
6193         if unicode.utf8.match(char, "[%s\n]") then
6194             char = "-"
6195             if prev_space then
6196                 goto continue
6197             else
6198                 prev_space = true
6199             end
6200         else
6201             -- Convert all alphabetic characters to lowercase.
6202             char = unicode.utf8.lower(char)
6203             prev_space = false
6204         end
6205
6206         table.insert(buffer, char)
6207
6208         ::continue::
6209     end
6210
6211     if prev_space then
6212         table.remove(buffer)
6213     end
6214
6215     local identifier = #buffer == 0 and "section" or table.concat(buffer, "")
6216     return identifier
6217 end

```

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

```
6218 self.secbegin_text = "\\markdownRendererSectionBegin\n"
6219 self.secend_text = "\n\\markdownRendererSectionEnd "
6220 function self.heading(s, level, attributes)
6221   local buf = {}
6222   local flat_text, inlines = table.unpack(s)
6223
6224   -- push empty attributes for implied sections
6225   while self.attribute_type_levels["heading"] < level - 1 do
6226     table.insert(buf,
6227                 self.push_attributes("heading",
6228                                     nil,
6229                                     self.secbegin_text,
6230                                     self.secend_text))
6231   end
6232
6233   -- pop attributes for sections that have ended
6234   while self.attribute_type_levels["heading"] >= level do
6235     table.insert(buf, self.pop_attributes("heading"))
6236   end
6237
6238   -- construct attributes for the new section
6239   local auto_identifiers = {}
6240   if self.options.autoIdentifiers then
6241     table.insert(auto_identifiers, create_auto_identifier(flat_text))
6242   end
6243   if self.options.gfmAutoIdentifiers then
6244     table.insert(auto_identifiers, create_gfm_auto_identifier(flat_text))
6245   end
6246   local normalized_attributes = normalize_attributes(attributes, auto_identifiers)
6247
6248   -- push attributes for the new section
6249   local start_output = {}
6250   local end_output = {}
6251   table.insert(start_output, self.secbegin_text)
6252   table.insert(end_output, self.secend_text)
6253
6254   table.insert(buf, self.push_attributes("heading",
6255                                         normalized_attributes,
6256                                         start_output,
6257                                         end_output))
6258   assert(self.attribute_type_levels["heading"] == level)
6259
6260   -- render the heading and its attributes
6261   if self.is_writing and #normalized_attributes > 0 then
6262     table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin\n")
```



```

6263     table.insert(buf, self.attributes(normalized_attributes, false))
6264 end
6265
6266 local cmd
6267 level = level + options.shiftHeadings
6268 if level <= 1 then
6269     cmd = "\\markdownRendererHeadingOne"
6270 elseif level == 2 then
6271     cmd = "\\markdownRendererHeadingTwo"
6272 elseif level == 3 then
6273     cmd = "\\markdownRendererHeadingThree"
6274 elseif level == 4 then
6275     cmd = "\\markdownRendererHeadingFour"
6276 elseif level == 5 then
6277     cmd = "\\markdownRendererHeadingFive"
6278 elseif level >= 6 then
6279     cmd = "\\markdownRendererHeadingSix"
6280 else
6281     cmd = ""
6282 end
6283 if self.is_writing then
6284     table.insert(buf, {cmd, "{", inlines, "}"})
6285 end
6286
6287 if self.is_writing and #normalized_attributes > 0 then
6288     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd{")
6289 end
6290
6291 return buf
6292 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.

```

6293 function self.get_state()
6294     return {
6295         is_writing=self.is_writing,
6296         flatten_inlines=self.flatten_inlines,
6297         active_attributes={table.unpack(self.active_attributes)},
6298     }
6299 end

```

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

```

6300 function self.set_state(s)
6301     local previous_state = self.get_state()
6302     for key, value in pairs(s) do
6303         self[key] = value
6304     end

```

```

6305     return previous_state
6306 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`.

```

6307 function self.defer_call(f)
6308     local previous_state = self.get_state()
6309     return function(...)
6310         local state = self.set_state(previous_state)
6311         local return_value = f(...)
6312         self.set_state(state)
6313         return return_value
6314     end
6315 end
6316
6317 return self
6318 end

```

3.1.4 Parsers

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

```

6319 local parsers = {}

```

3.1.4.1 Basic Parsers

```

6320 parsers.percent = P("%")
6321 parsers.at = P("@")
6322 parsers.comma = P(",")
6323 parsers.asterisk = P("*")
6324 parsers.dash = P("-")
6325 parsers.plus = P("+")
6326 parsers.underscore = P("_")
6327 parsers.period = P(".")
6328 parsers.hash = P("#")
6329 parsers.dollar = P("$")
6330 parsers.ampersand = P("&")
6331 parsers.backtick = P("`")
6332 parsers.less = P("<")
6333 parsers.more = P(">")
6334 parsers.space = P(" ")
6335 parsers.squote = P("'")
6336 parsers.dquote = P('"')
6337 parsers.lparent = P("(")
6338 parsers.rparent = P(")")
6339 parsers.lbracket = P("[")

```

```

6340 parsers.rbracket           = P("]")
6341 parsers.lbrace             = P("{")
6342 parsers.rbrace             = P("}")
6343 parsers.circumflex         = P("^")
6344 parsers.slash              = P("/")
6345 parsers.equal              = P("=")
6346 parsers.colon              = P(":")
6347 parsers.semicolon         = P(";")
6348 parsers.exclamation        = P("!")
6349 parsers.pipe               = P("|")
6350 parsers.tilde              = P("~")
6351 parsers.backslash          = P("\\")
6352 parsers.tab                 = P("\t")
6353 parsers.newline            = P("\n")
6354
6355 parsers.digit               = R("09")
6356 parsers.hexdigit           = R("09","af","AF")
6357 parsers.letter             = R("AZ","az")
6358 parsers.alphanumeric       = R("AZ","az","09")
6359 parsers.keyword           = parsers.letter
6360                            * (parsers.alphanumeric + parsers.dash)^0
6361
6362 parsers.doubleasterisks     = P("**")
6363 parsers.doubleunderscores   = P("__")
6364 parsers.doubletildes       = P("~~")
6365 parsers.fourspace          = P("    ")
6366
6367 parsers.any                 = P(1)
6368 parsers.succeed            = P(true)
6369 parsers.fail                = P(false)
6370
6371 parsers.internal_punctuation = S(";,.,?")
6372 parsers.ascii_punctuation  = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")

```

3.1.5 Unicode punctuation

This section documents the Unicode punctuation³³ recognized by the markdown reader. The punctuation is organized in the `parsers.punctuation` table according to the number of bytes occupied after conversion to UTF8.

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

```

6373 parsers.punctuation       = {}
6374 (function()

```

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

```

6375 local pathname = kpse.lookup("UnicodeData.txt")
6376 local file = assert(io.open(pathname, "r"),
6377 [[Could not open file "UnicodeData.txt"]])
6378 for line in file:lines() do
6379     local codepoint, major_category = line:match("^(%x+);[^\;]*;(%a)")
6380     if major_category == "P" or major_category == "S" then
6381         local code = unicode.utf8.char(tonumber(codepoint, 16))
6382         if parsers.punctuation[#code] == nil then
6383             parsers.punctuation[#code] = parsers.fail
6384         end
6385         local code_parser = parsers.succeed
6386         for i = 1, #code do
6387             local byte = code:sub(i, i)
6388             local byte_parser = S(byte)
6389             code_parser = code_parser
6390                 * byte_parser
6391         end
6392         parsers.punctuation[#code] = parsers.punctuation[#code]
6393             + code_parser
6394     end
6395 end
6396 assert(file:close())
6397 end)()
6398
6399 parsers.escapable           = parsers.ascii_punctuation
6400 parsers.anyescaped         = parsers.backslash / "" * parsers.escapable
6401                             + parsers.any
6402
6403 parsers.spacechar          = S("\t ")
6404 parsers.spacing            = S(" \n\r\t")
6405 parsers.nonspacechar       = parsers.any - parsers.spacing
6406 parsers.optionalspace     = parsers.spacechar^0
6407
6408 parsers.normalchar         = parsers.any - (V("SpecialChar")
6409                                             + parsers.spacing)
6410 parsers.eof                 = -parsers.any
6411 parsers.nonindentSPACE     = parsers.space^-3 * - parsers.spacechar
6412 parsers.indent             = parsers.space^-3 * parsers.tab
6413                             + parsers.fourspaces / ""
6414 parsers.linechar           = P(1 - parsers.newline)
6415
6416 parsers.blankline          = parsers.optionalspace
6417                             * parsers.newline / "\n"
6418 parsers.blanklines         = parsers.blankline^0
6419 parsers.skipblanklines     = (parsers.optionalspace * parsers.newline)^0
6420 parsers.indentedline       = parsers.indent / ""

```

```

6421                                     * C(parsers.linechar^1 * parsers.newline^-
    1)
6422 parsers.optionallyindentedline = parsers.indent^-1 /""
6423                                     * C(parsers.linechar^1 * parsers.newline^-
    1)
6424 parsers.sp                          = parsers.spacing^0
6425 parsers.spnl                        = parsers.optionalspace
6426                                     * (parsers.newline * parsers.optionalspace)^-
    1
6427 parsers.line                        = parsers.linechar^0 * parsers.newline
6428 parsers.nonemptyline                = parsers.line - parsers.blankline

```

3.1.5.1 Parsers Used for Indentation

```

6429
6430 parsers.leader      = parsers.space^-3
6431

```

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

```

6432 local function has_trail(indent_table)
6433   return indent_table ~= nil and
6434     indent_table.trail ~= nil and
6435     next(indent_table.trail) ~= nil
6436 end
6437

```

Check if indent table `indent_table` has any indents.

```

6438 local function has_indents(indent_table)
6439   return indent_table ~= nil and
6440     indent_table.indents ~= nil and
6441     next(indent_table.indents) ~= nil
6442 end
6443

```

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

```

6444 local function add_trail(indent_table, trail_info)
6445   indent_table.trail = trail_info
6446   return indent_table
6447 end
6448

```

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

```

6449 local function remove_trail(indent_table)
6450   indent_table.trail = nil
6451   return indent_table
6452 end
6453

```

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

```

6454 local function update_indent_table(indent_table, new_indent, add)
6455     indent_table = remove_trail(indent_table)
6456
6457     if not has_indents(indent_table) then
6458         indent_table.indents = {}
6459     end
6460
6461
6462     if add then
6463         indent_table.indents[#indent_table.indents + 1] = new_indent
6464     else
6465         if indent_table.indents[#indent_table.indents].name == new_indent.name then
6466             indent_table.indents[#indent_table.indents] = nil
6467         end
6468     end
6469
6470     return indent_table
6471 end
6472

```

Remove an indent by its name `name`.

```

6473 local function remove_indent(name)
6474     local function remove_indent_level(s, i, indent_table) -- luacheck: ignore s i
6475         indent_table = update_indent_table(indent_table, {name=name}, false)
6476         return true, indent_table
6477     end
6478
6479     return Cg(Cmt(Cb("indent_info"), remove_indent_level), "indent_info")
6480 end
6481

```

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`.

```

6482 local function process_starter_spacing(indent, spacing, minimum, left_strip_length)
6483     left_strip_length = left_strip_length or 0
6484
6485     local count = 0
6486     local tab_value = 4 - (indent) % 4
6487
6488     local code_started, minimum_found = false, false
6489     local code_start, minimum_remainder = "", ""
6490
6491     local left_total_stripped = 0
6492     local full_remainder = ""

```

```

6493
6494 if spacing ~= nil then
6495     for i = 1, #spacing do
6496         local character = spacing:sub(i, i)
6497
6498         if character == "\t" then
6499             count = count + tab_value
6500             tab_value = 4
6501         elseif character == " " then
6502             count = count + 1
6503             tab_value = 4 - (1 - tab_value) % 4
6504         end
6505
6506         if (left_strip_length ~= 0) then
6507             local possible_to_strip = math.min(count, left_strip_length)
6508             count = count - possible_to_strip
6509             left_strip_length = left_strip_length - possible_to_strip
6510             left_total_stripped = left_total_stripped + possible_to_strip
6511         else
6512             full_remainder = full_remainder .. character
6513         end
6514
6515         if (minimum_found) then
6516             minimum_remainder = minimum_remainder .. character
6517         elseif (count >= minimum) then
6518             minimum_found = true
6519             minimum_remainder = minimum_remainder .. string.rep(" ", count - minimum)
6520         end
6521
6522         if (code_started) then
6523             code_start = code_start .. character
6524         elseif (count >= minimum + 4) then
6525             code_started = true
6526             code_start = code_start .. string.rep(" ", count - (minimum + 4))
6527         end
6528     end
6529 end
6530
6531 local remainder
6532 if (code_started) then
6533     remainder = code_start
6534 else
6535     remainder = string.rep(" ", count - minimum)
6536 end
6537
6538 local is_minimum = count >= minimum
6539 return {

```

```

6540     is_code = code_started,
6541     remainder = remainder,
6542     left_total_stripped = left_total_stripped,
6543     is_minimum = is_minimum,
6544     minimum_remainder = minimum_remainder,
6545     total_length = count,
6546     full_remainder = full_remainder
6547 }
6548 end
6549

```

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

```

6550 local function count_indent_tab_level(indent_table)
6551     local count = 0
6552     if not has_indents(indent_table) then
6553         return count
6554     end
6555
6556     for i=1, #indent_table.indents do
6557         count = count + indent_table.indents[i].length
6558     end
6559     return count
6560 end
6561

```

Count the total width of a delimiter `delimiter`.

```

6562 local function total_delimiter_length(delimiter)
6563     local count = 0
6564     if type(delimiter) == "string" then return #delimiter end
6565     for _, value in pairs(delimiter) do
6566         count = count + total_delimiter_length(value)
6567     end
6568     return count
6569 end
6570

```

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`.

```

6571 local function process_starter_indent(_, _, indent_table, starter, is_blank, indent_t
6572     local last_trail = starter[1]
6573     local delimiter = starter[2]
6574     local raw_new_trail = starter[3]
6575
6576     if indent_type == "bq" and not breakable then
6577         indent_table.ignore_blockquote_blank = true
6578     end
6579
6580     if has_trail(indent_table) then

```



```

6581     local trail = indent_table.trail
6582     if trail.is_code then
6583         return false
6584     end
6585     last_trail = trail.remainder
6586 else
6587     local sp = process_starter_spacing(0, last_trail, 0, 0)
6588
6589     if sp.is_code then
6590         return false
6591     end
6592     last_trail = sp.remainder
6593 end
6594
6595 local preceding_indentation = count_indent_tab_level(indent_table) % 4
6596 local last_trail_length = #last_trail
6597 local delimiter_length = total_delimiter_length(delimiter)
6598
6599 local total_indent_level = preceding_indentation + last_trail_length + delimiter_le
6600
6601 local sp = {}
6602 if not is_blank then
6603     sp = process_starter_spacing(total_indent_level, raw_new_trail, 0, 1)
6604 end
6605
6606 local del_trail_length = sp.left_total_stripped
6607 if is_blank then
6608     del_trail_length = 1
6609 elseif not sp.is_code then
6610     del_trail_length = del_trail_length + #sp.remainder
6611 end
6612
6613 local indent_length = last_trail_length + delimiter_length + del_trail_length
6614 local new_indent_info = {name=indent_type, length=indent_length}
6615
6616 indent_table = update_indent_table(indent_table, new_indent_info, true)
6617 indent_table = add_trail(indent_table, {is_code=sp.is_code, remainder=sp.remainder,
6618                                     full_remainder=sp.full_remainder})
6619
6620 return true, indent_table
6621 end
6622

```

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

```

6623 local function decode_pattern(name)
6624     local delimiter = parsers.succeed
6625     if name == "bq" then
6626         delimiter = parsers.more

```

```

6627 end
6628
6629 return C(parsers.optionalspace) * C(delimiter) * C(parsers.optionalspace) * Cp()
6630 end
6631

```

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

```

6632 local function left_blank_starter(indent_table)
6633   local blank_starter_index
6634
6635   if not has_indents(indent_table) then
6636     return
6637   end
6638
6639   for i = #indent_table.indents,1,-1 do
6640     local value = indent_table.indents[i]
6641     if value.name == "li" then
6642       blank_starter_index = i
6643     else
6644       break
6645     end
6646   end
6647
6648   return blank_starter_index
6649 end
6650

```

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.

```

6651 local function traverse_indent(s, i, indent_table, is_optional, is_blank, current_line_indents)
6652   local new_index = i
6653
6654   local preceding_indentation = 0
6655   local current_trail = {}
6656
6657   local blank_starter = left_blank_starter(indent_table)
6658
6659   if current_line_indents == nil then
6660     current_line_indents = {}
6661   end
6662
6663   for index = 1,#indent_table.indents do
6664     local value = indent_table.indents[index]

```

```

6665     local pattern = decode_pattern(value.name)
6666
6667     -- match decoded pattern
6668     local new_indent_info = lpeg.match(Ct(pattern), s, new_index)
6669     if new_indent_info == nil then
6670         local blankline_end = lpeg.match(Ct(parsers.blankline * Cg(Cp(), "pos")), s, ne
6671         if is_optional or not indent_table.ignore_blockquote_blank or not blankline_end
6672             return is_optional, new_index, current_trail, current_line_indents
6673         end
6674
6675         return traverse_indent(s, tonumber(blankline_end.pos), indent_table, is_optiona
6676     end
6677
6678     local raw_last_trail = new_indent_info[1]
6679     local delimiter = new_indent_info[2]
6680     local raw_new_trail = new_indent_info[3]
6681     local next_index = new_indent_info[4]
6682
6683     local space_only = delimiter == ""
6684
6685     -- check previous trail
6686     if not space_only and next(current_trail) == nil then
6687         local sp = process_starter_spacing(0, raw_last_trail, 0, 0)
6688         current_trail = {is_code=sp.is_code, remainder=sp.remainder, total_length=sp.t
6689             full_remainder=sp.full_remainder}
6690     end
6691
6692     if next(current_trail) ~= nil then
6693         if not space_only and current_trail.is_code then
6694             return is_optional, new_index, current_trail, current_line_indents
6695         end
6696         if current_trail.internal_remainder ~= nil then
6697             raw_last_trail = current_trail.internal_remainder
6698         end
6699     end
6700
6701     local raw_last_trail_length = 0
6702     local delimiter_length = 0
6703
6704     if not space_only then
6705         delimiter_length = #delimiter
6706         raw_last_trail_length = #raw_last_trail
6707     end
6708
6709     local total_indent_level = preceding_indentation + raw_last_trail_length + delimi
6710
6711     local spacing_to_process

```

```

6712     local minimum = 0
6713     local left_strip_length = 0
6714
6715     if not space_only then
6716         spacing_to_process = raw_new_trail
6717         left_strip_length = 1
6718     else
6719         spacing_to_process = raw_last_trail
6720         minimum = value.length
6721     end
6722
6723     local sp = process_starter_spacing(total_indent_level, spacing_to_process, minimum)
6724
6725     if space_only and not sp.is_minimum then
6726         return is_optional or (is_blank and blank_starter <= index), new_index, current_trail
6727     end
6728
6729     local indent_length = raw_last_trail_length + delimiter_length + sp.left_total_strip_length
6730
6731     -- update info for the next pattern
6732     if not space_only then
6733         preceding_indentation = preceding_indentation + indent_length
6734     else
6735         preceding_indentation = preceding_indentation + value.length
6736     end
6737
6738     current_trail = {is_code=sp.is_code, remainder=sp.remainder, internal_remainder=sp.internal_remainder,
6739                    total_length=sp.total_length, full_remainder=sp.full_remainder}
6740
6741     current_line_indents[#current_line_indents + 1] = new_indent_info
6742     new_index = next_index
6743 end
6744
6745 return true, new_index, current_trail, current_line_indents
6746 end
6747

```

Check if a code trail is expected.

```

6748 local function check_trail(expect_code, is_code)
6749     return (expect_code and is_code) or (not expect_code and not is_code)
6750 end
6751

```

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`.

```

6752 local function check_trail_joined(s, i, indent_table, spacing, expect_code, omit_remainder)
6753     local is_code

```

```

6754 local remainder
6755
6756 if has_trail(indent_table) then
6757     local trail = indent_table.trail
6758     is_code = trail.is_code
6759     if is_code then
6760         remainder = trail.remainder
6761     else
6762         remainder = trail.full_remainder
6763     end
6764 else
6765     local sp = process_starter_spacing(0, spacing, 0, 0)
6766     is_code = sp.is_code
6767     if is_code then
6768         remainder = sp.remainder
6769     else
6770         remainder = sp.full_remainder
6771     end
6772 end
6773
6774 local result = check_trail(expect_code, is_code)
6775 if omit_remainder then
6776     return result
6777 end
6778 return result, remainder
6779 end
6780

```

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

```

6781 local function check_trail_length(s, i, indent_table, spacing, min, max) -- luacheck:
6782     local trail
6783
6784     if has_trail(indent_table) then
6785         trail = indent_table.trail
6786     else
6787         trail = process_starter_spacing(0, spacing, 0, 0)
6788     end
6789
6790     local total_length = trail.total_length
6791     if total_length == nil then
6792         return false
6793     end
6794
6795     return min <= total_length and total_length <= max
6796 end
6797

```

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

```
6798 local function check_continuation_indentation(s, i, indent_table, is_optional, is_bla
6799   if not has_indents(indent_table) then
6800     return true
6801   end
6802
6803   local passes, new_index, current_trail, current_line_indents =
6804     traverse_indent(s, i, indent_table, is_optional, is_blank)
6805
6806   if passes then
6807     indent_table.current_line_indents = current_line_indents
6808     indent_table = add_trail(indent_table, current_trail)
6809     return new_index, indent_table
6810   end
6811   return false
6812 end
6813
```

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

```
6814 local function get_last_indent_name(indent_table)
6815   if has_indents(indent_table) then
6816     return indent_table.indents[#indent_table.indents].name
6817   end
6818 end
6819
```

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

```
6820 local function remove_remainder_if_blank(indent_table, remainder)
6821   if get_last_indent_name(indent_table) == "li" then
6822     return ""
6823   end
6824   return remainder
6825 end
6826
```

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.

```
6827 local function check_trail_type(s, i, trail, spacing, trail_type) -- luacheck: ignore
6828   if trail == nil then
6829     trail = process_starter_spacing(0, spacing, 0, 0)
6830   end
6831
6832   if trail_type == "non-code" then
6833     return check_trail(false, trail.is_code)
6834   end
6835   if trail_type == "code" then
```

```

6836     return check_trail(true, trail.is_code)
6837 end
6838 if trail_type == "full-code" then
6839     if (trail.is_code) then
6840         return i, trail.remainder
6841     end
6842     return i, ""
6843 end
6844 if trail_type == "full-any" then
6845     return i, trail.internal_remainder
6846 end
6847 end
6848

```

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

```

6849 local function trail_freezing(s, i, indent_table, is_freezing) -- luacheck: ignore s
6850     if is_freezing then
6851         if indent_table.is_trail_frozen then
6852             indent_table.trail = indent_table.frozen_trail
6853         else
6854             indent_table.frozen_trail = indent_table.trail
6855             indent_table.is_trail_frozen = true
6856         end
6857     else
6858         indent_table.frozen_trail = nil
6859         indent_table.is_trail_frozen = false
6860     end
6861     return true, indent_table
6862 end
6863

```

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`.

```

6864 local function check_continuation_indentation_and_trail(s, i, indent_table, is_option
6865                                                                                   reset_rem, omit_remainder)
6866     if not has_indents(indent_table) then
6867         local spacing, new_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s, i)
6868         local result, remainder = check_trail_type(s, i, indent_table.trail, spacing, tra
6869         if remainder == nil then
6870             if result then
6871                 return new_index
6872             end
6873             return false
6874         end
6875         if result then
6876             return new_index, remainder
6877         end

```

```

6878     return false
6879 end
6880
6881 local passes, new_index, current_trail = traverse_indent(s, i, indent_table, is_opt
6882
6883 if passes then
6884     local spacing
6885     if current_trail == nil then
6886         local newer_spacing, newer_index = lpeg.match(C(parsers.spacechar^0) * Cp(), s,
6887             current_trail = process_starter_spacing(0, newer_spacing, 0, 0)
6888             new_index = newer_index
6889             spacing = newer_spacing
6890     else
6891         spacing = current_trail.remainder
6892     end
6893     local result, remainder = check_trail_type(s, new_index, current_trail, spacing,
6894     if remainder == nil or omit_remainder then
6895         if result then
6896             return new_index
6897         end
6898         return false
6899     end
6900
6901     if is_blank and reset_rem then
6902         remainder = remove_remainder_if_blank(indent_table, remainder)
6903     end
6904     if result then
6905         return new_index, remainder
6906     end
6907     return false
6908 end
6909 return false
6910 end
6911

```

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

```

6912 parsers.check_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(false), che
6913
6914 parsers.check_trail_no_rem = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(fals
6915
6916 parsers.check_code_trail = Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(true)
6917
6918 parsers.check_trail_length_range = function(min, max)
6919     return Cmt(Cb("indent_info") * C(parsers.spacechar^0) * Cc(min) * Cc(max), check_tr
6920 end
6921
6922 parsers.check_trail_length = function(n)
6923     return parsers.check_trail_length_range(n, n)

```



```
6924 end
6925
```

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

```
6926 parsers.freeze_trail = Cg(Cmt(Cb("indent_info") * Cc(true), trail_freezing), "indent_
6927
6928 parsers.unfreeze_trail = Cg(Cmt(Cb("indent_info") * Cc(false), trail_freezing), "inde
6929
```

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

```
6930 parsers.check_minimal_indent = Cmt(Cb("indent_info") * Cc(false), check_continuation_
6931
6932 parsers.check_optional_indent = Cmt(Cb("indent_info") * Cc(true), check_continuation_
6933
6934 parsers.check_minimal_blank_indent = Cmt(Cb("indent_info") * Cc(false) * Cc(true), ch
6935
```

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

```
6936
6937 parsers.check_minimal_indent_and_trail = Cmt( Cb("indent_info")
6938           * Cc(false) * Cc(false) * Cc("non-
           code") * Cc(true),
6939           check_continuation_indentation_and_trail)
6940
6941 parsers.check_minimal_indent_and_code_trail = Cmt( Cb("indent_info")
6942           * Cc(false) * Cc(false) * Cc("code")
6943           check_continuation_indentation_and_t
6944
6945 parsers.check_minimal_blank_indent_and_full_code_trail = Cmt( Cb("indent_info")
6946           * Cc(false) * Cc(true) *
           code") * Cc(true),
6947           check_continuation_indentation_and_full_code_trail)
6948
6949 parsers.check_minimal_indent_and_any_trail = Cmt( Cb("indent_info")
6950           * Cc(false) * Cc(false) * Cc("full-
           any") * Cc(true) * Cc(false),
6951           check_continuation_indentation_and_any_trail)
6952
6953 parsers.check_minimal_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6954           * Cc(false) * Cc(true) * Cc("fu
           any") * Cc(true) * Cc(false),
6955           check_continuation_indentation_and_any_trail)
6956
6957 parsers.check_minimal_blank_indent_and_any_trail_no_rem = Cmt( Cb("indent_info")
```

```

6958                                     * Cc(false) * Cc(true) * Cc("
any") * Cc(true) * Cc(true),
6959                                     check_continuation_indentation
6960
6961 parsers.check_optional_indent_and_any_trail = Cmt( Cb("indent_info")
6962                                     * Cc(true) * Cc(false) * Cc("full-
any") * Cc(true) * Cc(false),
6963                                     check_continuation_indentation_and_tr
6964
6965 parsers.check_optional_blank_indent_and_any_trail = Cmt( Cb("indent_info")
6966                                     * Cc(true) * Cc(true) * Cc("ful
any") * Cc(true) * Cc(false),
6967                                     check_continuation_indentation
6968

```

The following patterns specify behaviour around newlines.

```

6969
6970 parsers.spnlc_noexc = parsers.optionalspace
6971                                     * (parsers.newline * parsers.check_minimal_indent_and_any_trail)^
1
6972
6973 parsers.spnlc = parsers.optionalspace
6974                                     * (V("EndlineNoSub"))^-1
6975
6976 parsers.spnlc_sep = parsers.optionalspace * V("EndlineNoSub")
6977                                     + parsers.spacechar^1
6978
6979 parsers.only_blank = parsers.spacechar^0 * (parsers.newline + parsers.eof)
6980
6981 % \end{macrocode}
6982 % \begin{figure}
6983 % \hspace*{-0.1\textwidth}
6984 % \begin{minipage}{1.2\textwidth}
6985 % \centering
6986 % \begin{tikzpicture}[shorten >=1pt, line width=0.1mm, >={Stealth[length=2mm]}, node
6987 % \node[state, initial by diamond, accepting] (noop) {initial};
6988 % \node[state] (odd_backslash) [above right=of noop] {odd backslash};
6989 % \node[state] (even_backslash) [below right=of odd_backslash] {even backslash};
6990 % \node[state] (comment) [below=of noop] {comment};
6991 % \node[state] (leading_spaces) [below=of even_backslash, align=center] {leading tabs};
6992 % \node[state] (blank_line) [below right=of comment] {blank line};
6993 % \path[->]
6994 % (noop) edge [in=150, out=180, loop] node [align=center, yshift=-0.75cm] {match [$^
6995 % edge [bend right=10] node [below right=-0.2cm] {match \textbackslash} (odd_b
6996 % edge [bend left=30] node [left, align=center] {match \%\\capture \textbacksl
6997 % (comment) edge [in=305, out=325, loop] node [xshift=-1.2cm] {match [$^\wedge$$\drsh$
6998 % edge [bend left=10] node {match $\drsh$} (leading_spaces)
6999 % (leading_spaces) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$

```

```

7000 %             edge [bend right=90] node [right] {match \textbackslash} (odd_back
7001 %             edge [bend left=10] node {match \%} (comment)
7002 %             edge [bend right=10] node {${\epsilon}} (blank_line)
7003 %             edge [bend left=10] node [align=center, right=0.3cm] {match [${\w}
7004 % (blank_line) edge [loop below] node {match [\textvisiblespace$\rightleftarrows$]} (
7005 %             edge [bend left=90] node [align=center, below=1.2cm] {match $\drsh$\
7006 % (odd_backslash) edge [bend right=10] node [align=center, xshift=-0.3cm, yshift=0.2c
7007 %             edge [bend right=10] node [align=center, above left=-
0.3cm, xshift=0.1cm] {match [${\wedge}$\textbackslash]\for \%, capture \textbackslash
7008 % (even_backslash) edge [bend left=10] node {${\epsilon}} (noop);
7009 % \end{tikzpicture}
7010 % \caption{A pushdown automaton that recognizes \TeX{ } comments}
7011 % \label{fig:commented_line}
7012 % \end{minipage}
7013 % \end{figure}
7014 % \begin{markdown}
7015 %
7016 % The \luamdef{parsers.commented_line}^1 parser recognizes the regular
7017 % language of \TeX{ } comments, see an equivalent finite automaton in Figure
7018 % <#fig:commented_line>.
7019 %
7020 % \end{markdown}
7021 % \begin{macrocode}
7022 parsers.commented_line_letter = parsers.linechar
7023                               + parsers.newline
7024                               - parsers.backslash
7025                               - parsers.percent
7026 parsers.commented_line        = Cg(Cc(""), "backslashes")
7027                               * ((#(parsers.commented_line_letter
7028                                   - parsers.newline)
7029                                   * Cb("backslashes")
7030                                   * Cs(parsers.commented_line_letter
7031                                       - parsers.newline)^1 -- initial
7032                                   * Cg(Cc(""), "backslashes"))
7033                               + #(parsers.backslash * parsers.backslash)
7034                               * Cg((parsers.backslash -- even backslash
7035                                   * parsers.backslash)^1, "backslashes")
7036                               + (parsers.backslash
7037                                   * (#parsers.percent
7038                                       * Cb("backslashes")
7039                                       / function(backslashes)
7040                                           return string.rep("\\", #backslashes / 2)
7041                                       end
7042                                       * C(parsers.percent)
7043                                       + #parsers.commented_line_letter
7044                                       * Cb("backslashes")
7045                                       * Cc("\\"))

```

```

7046             * C(parsers.commented_line_letter))
7047             * Cg(Cc(""), "backslashes"))^0
7048 * (#parsers.percent
7049 * Cb("backslashes")
7050 / function(backslashes)
7051     return string.rep("\\", #backslashes / 2)
7052 end
7053 * ((parsers.percent -- comment
7054   * parsers.line
7055   * #parsers.blankline) -- blank line
7056 / "\n"
7057 + parsers.percent -- comment
7058   * parsers.line
7059   * parsers.optionalspace) -- leading tabs and space
7060 + #(parsers.newline)
7061 * Cb("backslashes")
7062 * C(parsers.newline))
7063
7064 parsers.chunk = parsers.line * (parsers.optionallyindentedline
7065                             - parsers.blankline)^0
7066
7067 parsers.attribute_key_char = parsers.alphanumeric + S("-_:.")
7068 parsers.attribute_raw_char = parsers.alphanumeric + S("-_")
7069 parsers.attribute_key = (parsers.attribute_key_char
7070                       - parsers.dash - parsers.digit)
7071 * parsers.attribute_key_char^0
7072 parsers.attribute_value = ( (parsers.dquote / "\"")
7073                           * (parsers.anyescaped - parsers.dquote)^0
7074                           * (parsers.dquote / "\""))
7075 + ( (parsers.squote / "\"")
7076   * (parsers.anyescaped - parsers.squote)^0
7077   * (parsers.squote / "\""))
7078 + ( parsers.anyescaped - parsers.dquote - parsers.rbracket
7079   - parsers.space)^0
7080 parsers.attribute_identifier = parsers.attribute_key_char^1
7081 parsers.attribute_classname = parsers.letter
7082 * parsers.attribute_key_char^0
7083 parsers.attribute_raw = parsers.attribute_raw_char^1
7084
7085 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
7086 + C( parsers.hash
7087     * parsers.attribute_identifier)
7088 + C( parsers.period
7089     * parsers.attribute_classname)
7090 + Cs( parsers.attribute_key
7091     * parsers.optionalspace * parsers.equal * parsers.optionalspace
7092     * parsers.attribute_value)

```

```

7093 parsers.attributes = parsers.lbrace
7094         * parsers.optionalspace
7095         * parsers.attribute
7096         * (parsers.spacechar^1
7097           * parsers.attribute)^0
7098         * parsers.optionalspace
7099         * parsers.rbrace
7100
7101
7102 parsers.raw_attribute = parsers.lbrace
7103         * parsers.optionalspace
7104         * parsers.equal
7105         * C(parsers.attribute_raw)
7106         * parsers.optionalspace
7107         * parsers.rbrace
7108
7109 -- block followed by 0 or more optionally
7110 -- indented blocks with first line indented.
7111 parsers.indented_blocks = function(bl)
7112   return Cs( bl
7113             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
7114             * (parsers.blankline^1 + parsers.eof) )
7115 end

```

3.1.5.2 Parsers Used for HTML Entities

```

7116 local function repeat_between(pattern, min, max)
7117   return -pattern^(max + 1) * pattern^min
7118 end
7119
7120 parsers.hexentity = parsers.ampersand * parsers.hash * C(S("Xx"))
7121                 * C(repeat_between(parsers.hexdigit, 1, 6)) * parsers.semicolon
7122 parsers.decentity = parsers.ampersand * parsers.hash
7123                 * C(repeat_between(parsers.digit, 1, 7)) * parsers.semicolon
7124 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
7125                 * parsers.semicolon
7126
7127 parsers.html_entities = parsers.hexentity / entities.hex_entity_with_x_char
7128                     + parsers.decentity / entities.dec_entity
7129                     + parsers.tagentity / entities.char_entity

```

3.1.5.3 Parsers Used for Markdown Lists

```

7130 parsers.bullet = function(bullet_char, interrupting)
7131   local allowed_end
7132   if interrupting then
7133     allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7134   else

```

```

7135     allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7136 end
7137 return parsers.check_trail
7138     * Ct(C(bullet_char) * Cc(""))
7139     * allowed_end
7140 end
7141
7142 local function tickbox(interior)
7143     return parsers.optionalspace * parsers.lbracket
7144         * interior * parsers.rbracket * parsers.spacechar^1
7145 end
7146
7147 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
7148 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
7149 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
7150

```

3.1.5.4 Parsers Used for Markdown Code Spans

```

7151 parsers.openticks = Cg(parsers.backtick^1, "ticks")
7152
7153 local function captures_equal_length(_,i,a,b)
7154     return #a == #b and i
7155 end
7156
7157 parsers.closeticks = Cmt(C(parsers.backtick^1)
7158     * Cb("ticks"), captures_equal_length)
7159
7160 parsers.intickschar = (parsers.any - S("\n\r`"))
7161     + V("NoSoftLineBreakEndline")
7162     + (parsers.backtick^1 - parsers.closeticks)
7163
7164 local function process_inticks(s)
7165     s = s:gsub("\n", " ")
7166     s = s:gsub("^ (.*) $", "%1")
7167     return s
7168 end
7169
7170 parsers.inticks = parsers.openticks
7171     * C(parsers.space^0)
7172     * parsers.closeticks
7173     + parsers.openticks
7174     * Cs(Cs(parsers.intickschar^0) / process_inticks)
7175     * parsers.closeticks
7176

```

3.1.5.5 Parsers Used for HTML

```

7177 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
7178 parsers.keyword_exact = function(s)
7179     local parser = P(0)
7180     for i=1,#s do
7181         local c = s:sub(i,i)
7182         local m = c .. upper(c)
7183         parser = parser * S(m)
7184     end
7185     return parser
7186 end
7187
7188 parsers.special_block_keyword =
7189     parsers.keyword_exact("pre") +
7190     parsers.keyword_exact("script") +
7191     parsers.keyword_exact("style") +
7192     parsers.keyword_exact("textarea")
7193
7194 parsers.block_keyword =
7195     parsers.keyword_exact("address") +
7196     parsers.keyword_exact("article") +
7197     parsers.keyword_exact("aside") +
7198     parsers.keyword_exact("base") +
7199     parsers.keyword_exact("basefont") +
7200     parsers.keyword_exact("blockquote") +
7201     parsers.keyword_exact("body") +
7202     parsers.keyword_exact("caption") +
7203     parsers.keyword_exact("center") +
7204     parsers.keyword_exact("col") +
7205     parsers.keyword_exact("colgroup") +
7206     parsers.keyword_exact("dd") +
7207     parsers.keyword_exact("details") +
7208     parsers.keyword_exact("dialog") +
7209     parsers.keyword_exact("dir") +
7210     parsers.keyword_exact("div") +
7211     parsers.keyword_exact("dl") +
7212     parsers.keyword_exact("dt") +
7213     parsers.keyword_exact("fieldset") +
7214     parsers.keyword_exact("figcaption") +
7215     parsers.keyword_exact("figure") +
7216     parsers.keyword_exact("footer") +
7217     parsers.keyword_exact("form") +
7218     parsers.keyword_exact("frame") +
7219     parsers.keyword_exact("frameset") +
7220     parsers.keyword_exact("h1") +
7221     parsers.keyword_exact("h2") +
7222     parsers.keyword_exact("h3") +
7223     parsers.keyword_exact("h4") +

```

```

7224     parsers.keyword_exact("h5") +
7225     parsers.keyword_exact("h6") +
7226     parsers.keyword_exact("head") +
7227     parsers.keyword_exact("header") +
7228     parsers.keyword_exact("hr") +
7229     parsers.keyword_exact("html") +
7230     parsers.keyword_exact("iframe") +
7231     parsers.keyword_exact("legend") +
7232     parsers.keyword_exact("li") +
7233     parsers.keyword_exact("link") +
7234     parsers.keyword_exact("main") +
7235     parsers.keyword_exact("menu") +
7236     parsers.keyword_exact("menuitem") +
7237     parsers.keyword_exact("nav") +
7238     parsers.keyword_exact("noframes") +
7239     parsers.keyword_exact("ol") +
7240     parsers.keyword_exact("optgroup") +
7241     parsers.keyword_exact("option") +
7242     parsers.keyword_exact("p") +
7243     parsers.keyword_exact("param") +
7244     parsers.keyword_exact("section") +
7245     parsers.keyword_exact("source") +
7246     parsers.keyword_exact("summary") +
7247     parsers.keyword_exact("table") +
7248     parsers.keyword_exact("tbody") +
7249     parsers.keyword_exact("td") +
7250     parsers.keyword_exact("tfoot") +
7251     parsers.keyword_exact("th") +
7252     parsers.keyword_exact("thead") +
7253     parsers.keyword_exact("title") +
7254     parsers.keyword_exact("tr") +
7255     parsers.keyword_exact("track") +
7256     parsers.keyword_exact("ul")
7257
7258 -- end conditions
7259 parsers.html_blankline_end_condition = parsers.linechar^0
7260     * ( parsers.newline
7261     * (parsers.check_minimal_blank_indent_and_any
7262     * #parsers.blankline
7263     + parsers.check_minimal_indent_and_any_trai
7264     * parsers.linechar^1)^0
7265     * (parsers.newline^-1 / "")
7266
7267 local function remove_trailing_blank_lines(s)
7268     return s:gsub("[\n\r]+%s*$", "")
7269 end
7270

```



```

7271 parsers.html_until_end = function(end_marker)
7272   return Cs(Cs((parsers.newline
7273     * (parsers.check_minimal_blank_indent_and_any_trail
7274     * #parsers.blankline
7275     + parsers.check_minimal_indent_and_any_trail)
7276     + parsers.linechar - end_marker)^0
7277     * parsers.linechar^0 * parsers.newline^-1)
7278     / remove_trailing_blank_lines)
7279 end
7280
7281 -- attributes
7282 parsers.html_attribute_spacing = parsers.optionalspace
7283     * V("NoSoftLineBreakEndline")
7284     * parsers.optionalspace
7285     + parsers.spacechar^1
7286
7287 parsers.html_attribute_name = (parsers.letter + parsers.colon + parsers.underscore)
7288     * (parsers.alphanumeric + parsers.colon + parsers.underscore
7289     + parsers.period + parsers.dash)^0
7290
7291 parsers.html_attribute_value = parsers.squote
7292     * (parsers.linechar - parsers.squote)^0
7293     * parsers.squote
7294     + parsers.dquote
7295     * (parsers.linechar - parsers.dquote)^0
7296     * parsers.dquote
7297     + ( parsers.any - parsers.spacechar - parsers.newline
7298     - parsers.dquote - parsers.squote - parsers.backtick
7299     - parsers.equal - parsers.less - parsers.more)^1
7300
7301 parsers.html_inline_attribute_value = parsers.squote
7302     * (V("NoSoftLineBreakEndline")
7303     + parsers.any
7304     - parsers.blankline^2
7305     - parsers.squote)^0
7306     * parsers.squote
7307     + parsers.dquote
7308     * (V("NoSoftLineBreakEndline")
7309     + parsers.any
7310     - parsers.blankline^2
7311     - parsers.dquote)^0
7312     * parsers.dquote
7313     + (parsers.any - parsers.spacechar - parsers.newline
7314     - parsers.dquote - parsers.squote - parsers.backtick
7315     - parsers.equal - parsers.less - parsers.more)^1
7316
7317 parsers.html_attribute_value_specification = parsers.optionalspace

```

```

7318                                     * parsers.equal
7319                                     * parsers.optionalspace
7320                                     * parsers.html_attribute_value
7321
7322 parsers.html_spnl = parsers.optionalspace
7323                                     * (V("NoSoftLineBreakEndline") * parsers.optionalspace)^-
7324                                     1
7325 parsers.html_inline_attribute_value_specification = parsers.html_spnl
7326                                                         * parsers.equal
7327                                                         * parsers.html_spnl
7328                                                         * parsers.html_inline_attribute_val
7329
7330 parsers.html_attribute = parsers.html_attribute_spacing
7331                                     * parsers.html_attribute_name
7332                                     * parsers.html_inline_attribute_value_specification^-
7333                                     1
7334 parsers.html_non_newline_attribute = parsers.spacechar^1
7335                                     * parsers.html_attribute_name
7336                                     * parsers.html_attribute_value_specification^-
7337                                     1
7338 parsers.nested_breaking_blank = parsers.newline
7339                                     * parsers.check_minimal_blank_indent
7340                                     * parsers.blankline
7341
7342 parsers.html_comment_start = P("<!--")
7343
7344 parsers.html_comment_end = P("-->")
7345
7346 parsers.html_comment = Cs( parsers.html_comment_start
7347                             * parsers.html_until_end(parsers.html_comment_end))
7348
7349 parsers.html_inline_comment = (parsers.html_comment_start / "")
7350                             * -P(">") * -P("-->")
7351                             * Cs((V("NoSoftLineBreakEndline") + parsers.any
7352                                   - parsers.nested_breaking_blank - parsers.html_commen
7353                             * (parsers.html_comment_end / ""))
7354
7355 parsers.html_cdatasection_start = P("<![CDATA[")
7356
7357 parsers.html_cdatasection_end = P("]]>")
7358
7359 parsers.html_cdatasection = Cs( parsers.html_cdatasection_start
7360                                 * parsers.html_until_end(parsers.html_cdatasection_end))
7361

```

```

7362 parsers.html_inline_cdatasection = parsers.html_cdatasection_start
7363                                     * Cs(V("NoSoftLineBreakEndline") + parsers.any
7364                                         - parsers.nested_breaking_blank - parsers.html_
7365                                         * parsers.html_cdatasection_end
7366
7367 parsers.html_declaration_start = P("<!") * parsers.letter
7368
7369 parsers.html_declaration_end = P(">")
7370
7371 parsers.html_declaration = Cs( parsers.html_declaration_start
7372                               * parsers.html_until_end(parsers.html_declaration_end))
7373
7374 parsers.html_inline_declaration = parsers.html_declaration_start
7375                                 * Cs(V("NoSoftLineBreakEndline") + parsers.any
7376                                     - parsers.nested_breaking_blank - parsers.html_de
7377                                     * parsers.html_declaration_end
7378
7379 parsers.html_instruction_start = P("<?")
7380
7381 parsers.html_instruction_end = P("?>")
7382
7383 parsers.html_instruction = Cs( parsers.html_instruction_start
7384                               * parsers.html_until_end(parsers.html_instruction_end))
7385
7386 parsers.html_inline_instruction = parsers.html_instruction_start
7387                                 * Cs(V("NoSoftLineBreakEndline") + parsers.any
7388                                     - parsers.nested_breaking_blank - parsers.html_in
7389                                     * parsers.html_instruction_end
7390
7391 parsers.html_blankline = parsers.newline
7392                         * parsers.optionalspace
7393                         * parsers.newline
7394
7395 parsers.html_tag_start = parsers.less
7396
7397 parsers.html_tag_closing_start = parsers.less
7398                               * parsers.slash
7399
7400 parsers.html_tag_end = parsers.html_spnl
7401                      * parsers.more
7402
7403 parsers.html_empty_tag_end = parsers.html_spnl
7404                            * parsers.slash
7405                            * parsers.more
7406
7407 -- opening tags
7408 parsers.html_any_open_inline_tag = parsers.html_tag_start

```

```

7409             * parsers.keyword
7410             * parsers.html_attribute^0
7411             * parsers.html_tag_end
7412
7413 parsers.html_any_open_tag = parsers.html_tag_start
7414             * parsers.keyword
7415             * parsers.html_non_newline_attribute^0
7416             * parsers.html_tag_end
7417
7418 parsers.html_open_tag = parsers.html_tag_start
7419             * parsers.block_keyword
7420             * parsers.html_attribute^0
7421             * parsers.html_tag_end
7422
7423 parsers.html_open_special_tag = parsers.html_tag_start
7424             * parsers.special_block_keyword
7425             * parsers.html_attribute^0
7426             * parsers.html_tag_end
7427
7428 -- incomplete tags
7429 parsers.incomplete_tag_following = parsers.spacechar
7430             + parsers.more
7431             + parsers.slash * parsers.more
7432             + #(parsers.newline + parsers.eof)
7433
7434 parsers.incomplete_special_tag_following = parsers.spacechar
7435             + parsers.more
7436             + #(parsers.newline + parsers.eof)
7437
7438 parsers.html_incomplete_open_tag = parsers.html_tag_start
7439             * parsers.block_keyword
7440             * parsers.incomplete_tag_following
7441
7442 parsers.html_incomplete_open_special_tag = parsers.html_tag_start
7443             * parsers.special_block_keyword
7444             * parsers.incomplete_special_tag_following
7445
7446 parsers.html_incomplete_close_tag = parsers.html_tag_closing_start
7447             * parsers.block_keyword
7448             * parsers.incomplete_tag_following
7449
7450 parsers.html_incomplete_close_special_tag = parsers.html_tag_closing_start
7451             * parsers.special_block_keyword
7452             * parsers.incomplete_tag_following
7453
7454 -- closing tags
7455 parsers.html_close_tag = parsers.html_tag_closing_start

```

```

7456             * parsers.block_keyword
7457             * parsers.html_tag_end
7458
7459 parsers.html_any_close_tag = parsers.html_tag_closing_start
7460             * parsers.keyword
7461             * parsers.html_tag_end
7462
7463 parsers.html_close_special_tag = parsers.html_tag_closing_start
7464             * parsers.special_block_keyword
7465             * parsers.html_tag_end
7466
7467 -- empty tags
7468 parsers.html_any_empty_inline_tag = parsers.html_tag_start
7469             * parsers.keyword
7470             * parsers.html_attribute^0
7471             * parsers.html_empty_tag_end
7472
7473 parsers.html_any_empty_tag = parsers.html_tag_start
7474             * parsers.keyword
7475             * parsers.html_non_newline_attribute^0
7476             * parsers.optionalspace
7477             * parsers.slash
7478             * parsers.more
7479
7480 parsers.html_empty_tag = parsers.html_tag_start
7481             * parsers.block_keyword
7482             * parsers.html_attribute^0
7483             * parsers.html_empty_tag_end
7484
7485 parsers.html_empty_special_tag = parsers.html_tag_start
7486             * parsers.special_block_keyword
7487             * parsers.html_attribute^0
7488             * parsers.html_empty_tag_end
7489
7490 parsers.html_incomplete_blocks = parsers.html_incomplete_open_tag
7491             + parsers.html_incomplete_open_special_tag
7492             + parsers.html_incomplete_close_tag
7493
7494 -- parse special html blocks
7495 parsers.html_blankline_ending_special_block_opening = (parsers.html_close_special_tag
7496             + parsers.html_empty_special_tag
7497             * #(parsers.optionalspace
7498             * (parsers.newline + parsers.e
7499
7500 parsers.html_blankline_ending_special_block = parsers.html_blankline_ending_special_b
7501             * parsers.html_blankline_end_condition
7502

```

```

7503 parsers.html_special_block_opening = parsers.html_incomplete_open_special_tag
7504                                     - parsers.html_empty_special_tag
7505
7506 parsers.html_closing_special_block = parsers.html_special_block_opening
7507                                     * parsers.html_until_end(parsers.html_close_speci
7508
7509 parsers.html_special_block = parsers.html_blankline_ending_special_block
7510                             + parsers.html_closing_special_block
7511
7512 -- parse html blocks
7513 parsers.html_block_opening = parsers.html_incomplete_open_tag
7514                             + parsers.html_incomplete_close_tag
7515
7516 parsers.html_block = parsers.html_block_opening
7517                       * parsers.html_blankline_end_condition
7518
7519 -- parse any html blocks
7520 parsers.html_any_block_opening = (parsers.html_any_open_tag
7521                                   + parsers.html_any_close_tag
7522                                   + parsers.html_any_empty_tag)
7523                                   * #(parsers.optionalspace * (parsers.newline + parser
7524
7525 parsers.html_any_block = parsers.html_any_block_opening
7526                         * parsers.html_blankline_end_condition
7527
7528 parsers.html_inline_comment_full = parsers.html_comment_start
7529                                   * -P(">") * -P("->")
7530                                   * Cs((V("NoSoftLineBreakEndline") + parsers.any - P
7531                                     ")
7532                                     - parsers.nested_breaking_blank - parsers.html_
7533                                     * parsers.html_comment_end
7534
7534 parsers.html_inline_tags = parsers.html_inline_comment_full
7535                           + parsers.html_any_empty_inline_tag
7536                           + parsers.html_inline_instruction
7537                           + parsers.html_inline_cdatasection
7538                           + parsers.html_inline_declaration
7539                           + parsers.html_any_open_inline_tag
7540                           + parsers.html_any_close_tag
7541

```

3.1.5.6 Parsers Used for Markdown Tags and Links

```

7542 parsers.urlchar = parsers.anyescaped
7543                 - parsers.newline
7544                 - parsers.more
7545

```

```

7546 parsers.auto_link_scheme_part = parsers.alphanumeric
7547                                 + parsers.plus
7548                                 + parsers.period
7549                                 + parsers.dash
7550
7551 parsers.auto_link_scheme = parsers.letter
7552                             * parsers.auto_link_scheme_part
7553                             * parsers.auto_link_scheme_part^-30
7554
7555 parsers.absolute_uri = parsers.auto_link_scheme * parsers.colon
7556                       * (parsers.any - parsers.spacing - parsers.less - parsers.more)
7557
7558 parsers.printable_characters = S(" !#$%&'*/+/?^_`{|}~-")
7559
7560 parsers.email_address_local_part_char = parsers.alphanumeric
7561                                         + parsers.printable_characters
7562
7563 parsers.email_address_local_part = parsers.email_address_local_part_char^1
7564
7565 parsers.email_address_dns_label = parsers.alphanumeric
7566                                  * (parsers.alphanumeric + parsers.dash)^-
7567
7568                                  * B(parsers.alphanumeric)
7569
7570 parsers.email_address_domain = parsers.email_address_dns_label
7571                               * (parsers.period * parsers.email_address_dns_label)^0
7572
7573 parsers.email_address = parsers.email_address_local_part
7574                       * parsers.at
7575                       * parsers.email_address_domain
7576
7576 parsers.auto_link_url = parsers.less
7577                       * C(parsers.absolute_uri)
7578                       * parsers.more
7579
7580 parsers.auto_link_email = parsers.less
7581                         * C(parsers.email_address)
7582                         * parsers.more
7583
7584 parsers.auto_link_relative_reference = parsers.less
7585                                       * C(parsers.urlchar^1)
7586                                       * parsers.more
7587
7588 parsers.autolink = parsers.auto_link_url
7589                 + parsers.auto_link_email
7590
7591 -- content in balanced brackets, parentheses, or quotes:

```

```

7592 parsers.bracketed = P{ parsers.lbracket
7593     * (( parsers.backslash / "\"" * parsers.rbracket
7594         + parsers.any - (parsers.lbracket
7595                         + parsers.rbracket
7596                         + parsers.blankline^2)
7597         ) + V(1))^0
7598     * parsers.rbracket }
7599
7600 parsers.inparens = P{ parsers.lparent
7601     * ((parsers.anyescaped - (parsers.lparent
7602                             + parsers.rparent
7603                             + parsers.blankline^2)
7604     ) + V(1))^0
7605     * parsers.rparent }
7606
7607 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
7608     * ((parsers.anyescaped - (parsers.squote
7609                             + parsers.blankline^2)
7610     ) + V(1))^0
7611     * parsers.squote }
7612
7613 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
7614     * ((parsers.anyescaped - (parsers.dquote
7615                             + parsers.blankline^2)
7616     ) + V(1))^0
7617     * parsers.dquote }
7618
7619 parsers.link_text = parsers.lbracket
7620     * Cs((parsers.alphanumeric^1
7621         + parsers.bracketed
7622         + parsers.inticks
7623         + parsers.autolink
7624         + V("InlineHtml")
7625         + ( parsers.backslash * parsers.backslash)
7626         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7627         + V("NoSoftLineBreakSpace")
7628         + V("NoSoftLineBreakEndline")
7629         + (parsers.any
7630           - (parsers.newline + parsers.lbracket + parsers.rbracket)
7631         * parsers.rbracket
7632
7633 parsers.link_label_body = -(parsers.sp * parsers.rbracket)
7634     * #((parsers.any - parsers.rbracket)^-999 * parsers.rbracket)
7635     * Cs((parsers.alphanumeric^1
7636         + parsers.inticks
7637         + parsers.autolink
7638         + V("InlineHtml")

```



```

7639         + ( parsers.backslash * parsers.backslash)
7640         + ( parsers.backslash * (parsers.lbracket + parsers.rbracket)
7641         + V("NoSoftLineBreakSpace")
7642         + V("NoSoftLineBreakEndline")
7643         + (parsers.any
7644         - (parsers.newline + parsers.lbracket + parsers.rbracket)
7645
7646 parsers.link_label = parsers.lbracket
7647                   * parsers.link_label_body
7648                   * parsers.rbracket
7649
7650 parsers.inparens_url = P{ parsers.lparent
7651                       * ((parsers.anyescaped - (parsers.lparent
7652                       + parsers.rparent
7653                       + parsers.spacing)
7654                       ) + V(1))^0
7655                       * parsers.rparent }
7656
7657 -- url for markdown links, allowing nested brackets:
7658 parsers.url       = parsers.less * Cs((parsers.anyescaped
7659                                     - parsers.newline
7660                                     - parsers.less
7661                                     - parsers.more)^0)
7662                                     * parsers.more
7663 + -parsers.less
7664 * Cs((parsers.inparens_url + (parsers.anyescaped
7665                                     - parsers.spacing
7666                                     - parsers.lparent
7667                                     - parsers.rparent))^1)
7668
7669 -- quoted text:
7670 parsers.title_s   = parsers.squote
7671                   * Cs((parsers.html_entities
7672                   + V("NoSoftLineBreakSpace")
7673                   + V("NoSoftLineBreakEndline")
7674                   + (parsers.anyescaped - parsers.newline - parsers.squote - p
7675                   * parsers.squote
7676
7677 parsers.title_d   = parsers.dquote
7678                   * Cs((parsers.html_entities
7679                   + V("NoSoftLineBreakSpace")
7680                   + V("NoSoftLineBreakEndline")
7681                   + (parsers.anyescaped - parsers.newline - parsers.dquote - p
7682                   * parsers.dquote
7683
7684 parsers.title_p   = parsers.lparent
7685                   * Cs((parsers.html_entities

```

```

7686             + V("NoSoftLineBreakSpace")
7687             + V("NoSoftLineBreakEndline")
7688             + (parsers.anyescaped - parsers.newline - parsers.lparent -
7689               - parsers.blankline^2))^0)
7690         * parsers.rparent
7691
7692 parsers.title      = parsers.title_d + parsers.title_s + parsers.title_p
7693
7694 parsers.optionaltitle
7695             = parsers.spnlc * parsers.title * parsers.spacechar^0
7696             + Cc("")
7697

```

3.1.5.7 Helpers for Links and Link Reference Definitions

```

7698 -- parse a reference definition: [foo]: /bar "title"
7699 parsers.define_reference_parser = (parsers.check_trail / "") * parsers.link_label * p
7700                                 * parsers.spnlc * parsers.url
7701                                 * ( parsers.spnlc_sep * parsers.title * parsers.only_
7702                                   + Cc("") * parsers.only_blank)

```

3.1.5.8 Inline Elements

```

7703 parsers.Inline      = V("Inline")
7704
7705 -- parse many p between starter and ender
7706 parsers.between = function(p, starter, ender)
7707     local ender2 = B(parsers.nonspacechar) * ender
7708     return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
7709 end
7710

```

3.1.5.9 Block Elements

```

7711 parsers.lineof = function(c)
7712     return (parsers.check_trail_no_rem * (P(c) * parsers.optionalspace)^3
7713           * (parsers.newline + parsers.eof))
7714 end
7715
7716 parsers.thematic_break_lines = parsers.lineof(parsers.asterisk)
7717                               + parsers.lineof(parsers.dash)
7718                               + parsers.lineof(parsers.underscore)

```

3.1.5.10 Headings

```

7719 -- parse Atx heading start and return level
7720 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
7721                       * -parsers.hash / length
7722

```

```

7723 -- parse setext header ending and return level
7724 parsers.heading_level = parsers.nonindentspace * parsers.equal^1 * parsers.optionalsp
7725                        + parsers.nonindentspace * parsers.dash^1 * parsers.optionalspa
7726
7727 local function strip_atx_end(s)
7728   return s:gsub("%s+#+%s*\n$", "")
7729 end
7730
7731 parsers.atx_heading = parsers.check_trail_no_rem
7732                    * Cg(parsers.heading_start, "level")
7733                    * (C( parsers.optionalspace
7734                        * parsers.hash^0
7735                        * parsers.optionalspace
7736                        * parsers.newline)
7737                      + parsers.spacechar^1
7738                      * 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`.

```

7739 M.reader = {}
7740 function M.reader.new(writer, options)
7741   local self = {}

```

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

```

7742   self.writer = writer
7743   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.

```

7744   self.parsers = {}
7745   (function(parsers)
7746     setmetatable(self.parsers, {
7747       __index = function (_, key)
7748         return parsers[key]
7749       end

```

```

7750     })
7751 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.

```

7752 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.

```

7753 function self.normalize_tag(tag)
7754     tag = util.rope_to_string(tag)
7755     tag = tag:gsub("[ \n\r\t]+", " ")
7756     tag = tag:gsub("^ ", ""):gsub(" $", "")
7757     tag = uni_algos.case.casefold(tag, true, false)
7758     return tag
7759 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.

```

7760 local function iterlines(s, f)
7761     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
7762     return util.rope_to_string(rope)
7763 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.

```

7764 if options.preserveTabs then
7765     self.expandtabs = function(s) return s end
7766 else
7767     self.expandtabs = function(s)
7768         if s:find("\t") then
7769             return iterlines(s, util.expand_tabs_in_line)
7770         else
7771             return s
7772         end
7773     end
7774 end

```

3.1.6.2 High-Level Parser Functions

Create a `reader->parser_functions` hash table that stores high-level parser functions. Define `reader->create_parser` as a function that will create a high-level parser function `reader->parser_functions.name`, that matches input using

grammar `grammar`. If `toplevel` is true, the input is expected to come straight from the user, not from a recursive call, and will be preprocessed.

```
7775 self.parser_functions = {}
7776 self.create_parser = function(name, grammar, toplevel)
7777     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.

```
7778     if toplevel and options.stripIndent then
7779         local min_prefix_length, min_prefix = nil, ''
7780         str = iterlines(str, function(line)
7781             if lpeg.match(parsers.nonemptyline, line) == nil then
7782                 return line
7783             end
7784             line = util.expand_tabs_in_line(line)
7785             local prefix = lpeg.match(C(parsers.optionalspace), line)
7786             local prefix_length = #prefix
7787             local is_shorter = min_prefix_length == nil
7788             is_shorter = is_shorter or prefix_length < min_prefix_length
7789             if is_shorter then
7790                 min_prefix_length, min_prefix = prefix_length, prefix
7791             end
7792             return line
7793         end)
7794         str = str:gsub('^' .. min_prefix, '')
7795     end
```

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

```
7796     if toplevel and (options.texComments or options.hybrid) then
7797         str = lpeg.match(Ct(parsers.commented_line^1), str)
7798         str = util.rope_to_string(str)
7799     end
7800     local res = lpeg.match(grammar(), str)
7801     if res == nil then
7802         error(format("%s failed on:\n%s", name, str:sub(1,20)))
7803     else
7804         return res
7805     end
7806 end
7807 end
7808
7809 self.create_parser("parse_blocks",
7810     function()
```

```

7811         return parsers.blocks
7812     end, true)
7813
7814     self.create_parser("parse_blocks_nested",
7815         function()
7816             return parsers.blocks_nested
7817         end, false)
7818
7819     self.create_parser("parse_inlines",
7820         function()
7821             return parsers.inlines
7822         end, false)
7823
7824     self.create_parser("parse_inlines_no_inline_note",
7825         function()
7826             return parsers.inlines_no_inline_note
7827         end, false)
7828
7829     self.create_parser("parse_inlines_no_html",
7830         function()
7831             return parsers.inlines_no_html
7832         end, false)
7833
7834     self.create_parser("parse_inlines_nbsp",
7835         function()
7836             return parsers.inlines_nbsp
7837         end, false)
7838     self.create_parser("parse_inlines_no_link_or_emphasis",
7839         function()
7840             return parsers.inlines_no_link_or_emphasis
7841         end, false)

```

3.1.6.3 Parsers Used for Indentation (local)

The following patterns represent basic building blocks of indented content.

```

7842     parsers.minimally_indented_blankline = parsers.check_minimal_indent * (parsers.blankline)
7843
7844     parsers.minimally_indented_block = parsers.check_minimal_indent * V("Block")
7845
7846     parsers.minimally_indented_block_or_paragraph = parsers.check_minimal_indent * V("Block|Paragraph")
7847
7848     parsers.minimally_indented_paragraph = parsers.check_minimal_indent * V("Paragraph")
7849
7850     parsers.minimally_indented_plain = parsers.check_minimal_indent * V("Plain")
7851
7852     parsers.minimally_indented_par_or_plain = parsers.minimally_indented_paragraph
7853         + parsers.minimally_indented_plain

```

```

7854
7855 parsers.minimally_indented_par_or_plain_no_blank = parsers.minimally_indented_par_
7856                                                    - parsers.minimally_indented_blan
7857
7858 parsers.minimally_indented_ref = parsers.check_minimal_indent * V("Reference")
7859
7860 parsers.minimally_indented_blank = parsers.check_minimal_indent * V("Blank")
7861
7862 parsers.conditionally_indented_blankline = parsers.check_minimal_blank_indent * (pa
7863
7864 parsers.minimally_indented_ref_or_block = parsers.minimally_indented_ref
7865                                           + parsers.minimally_indented_block
7866                                           - parsers.minimally_indented_blankline
7867
7868 parsers.minimally_indented_ref_or_block_or_par = parsers.minimally_indented_ref
7869                                                  + parsers.minimally_indented_block_
7870                                                  - parsers.minimally_indented_blan
7871

```

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

```

7872
7873 parsers.separator_loop = function(separated_block, paragraph, block_separator, para
7874     return separated_block
7875         + block_separator
7876         * paragraph
7877         * separated_block
7878         + paragraph_separator
7879         * paragraph
7880 end
7881
7882 parsers.create_loop_body_pair = function(separated_block, paragraph, block_separato
7883     return {
7884         block = parsers.separator_loop(separated_block, paragraph, block_separator, blo
7885         par = parsers.separator_loop(separated_block, paragraph, block_separator, para
7886     }
7887 end
7888
7889 parsers.block_sep_group = function(blank)
7890     return blank^0 * parsers.eof
7891         + ( blank^2 / writer.paragraphsep
7892           + blank^0 / writer.interblocksep
7893         )
7894 end
7895
7896 parsers.par_sep_group = function(blank)
7897     return blank^0 * parsers.eof

```

```

7898         + blank^0 / writer.paragraphsep
7899     end
7900
7901     parsers.sep_group_no_output = function(blank)
7902         return blank^0 * parsers.eof
7903             + blank^0
7904     end
7905
7906     parsers.content_blank = parsers.minimally_indented_blankline
7907
7908     parsers.ref_or_block_separated = parsers.sep_group_no_output(parsers.content_blank
7909         * ( parsers.minimally_indented_ref
7910           - parsers.content_blank)
7911         + parsers.block_sep_group(parsers.content_blank)
7912         * ( parsers.minimally_indented_block
7913           - parsers.content_blank)
7914
7915     parsers.loop_body_pair =
7916         parsers.create_loop_body_pair(parsers.ref_or_block_separated,
7917             parsers.minimally_indented_par_or_plain_no_blank,
7918             parsers.block_sep_group(parsers.content_blank),
7919             parsers.par_sep_group(parsers.content_blank))
7920
7921     parsers.content_loop = ( V("Block")
7922         * parsers.loop_body_pair.block^0
7923         + (V("Paragraph") + V("Plain"))
7924         * parsers.ref_or_block_separated
7925         * parsers.loop_body_pair.block^0
7926         + (V("Paragraph") + V("Plain"))
7927         * parsers.loop_body_pair.par^0)
7928         * parsers.content_blank^0
7929
7930     parsers.indented_content = function()
7931         return Ct( (V("Reference") + (parsers.blankline / ""))
7932             * parsers.content_blank^0
7933             * parsers.check_minimal_indent
7934             * parsers.content_loop
7935             + (V("Reference") + (parsers.blankline / ""))
7936             * parsers.content_blank^0
7937             + parsers.content_loop)
7938     end
7939
7940     parsers.add_indent = function(pattern, name, breakable)
7941         return Cg(Cmt( Cb("indent_info")
7942             * Ct(pattern)
7943             * (#parsers.linechar * Cc(false) + Cc(true)) -- check if starter is
7944             * Cc(name)

```



```

7945             * Cc(breakable),
7946             process_starter_indent), "indent_info")
7947     end
7948

```

3.1.6.4 Parsers Used for Markdown Lists (local)

```

7949     if options.hashEnumerators then
7950         parsers.dig = parsers.digit + parsers.hash
7951     else
7952         parsers.dig = parsers.digit
7953     end
7954
7955     parsers.enumerator = function(delimiter_type, interrupting)
7956         local delimiter_range
7957         local allowed_end
7958         if interrupting then
7959             delimiter_range = P("1")
7960             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
7961         else
7962             delimiter_range = parsers.dig * parsers.dig^-8
7963             allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
7964         end
7965
7966         return parsers.check_trail
7967             * Ct(C(delimiter_range) * C(delimiter_type))
7968             * allowed_end
7969     end
7970
7971     parsers.starter = parsers.bullet(parsers.dash)
7972         + parsers.bullet(parsers.asterisk)
7973         + parsers.bullet(parsers.plus)
7974         + parsers.enumerator(parsers.period)
7975         + parsers.enumerator(parsers.rparent)
7976

```

3.1.6.5 Parsers Used for Blockquotes (local)

```

7977     parsers.blockquote_start = parsers.check_trail * C(parsers.more) * C(parsers.space)
7978
7979     parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", true)
7980         * parsers.indented_content()
7981         * remove_indent("bq")
7982
7983     if not options.breakableBlockquotes then
7984         parsers.blockquote_body = parsers.add_indent(parsers.blockquote_start, "bq", false)
7985             * parsers.indented_content()
7986             * remove_indent("bq")

```

```
7987 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.

```
7988 local function parse_content_part(content_part)
7989   local rope = util.rope_to_string(content_part)
7990   local parsed = self.parser_functions.parse_inlines_no_link_or_emphasis(rope)
7991   parsed.indent_info = nil
7992   return parsed
7993 end
7994
```

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

```
7995 local function collect_emphasis_content(t, opening_index, closing_index)
7996   local content = {}
7997
7998   local content_part = {}
7999   for i = opening_index, closing_index do
8000     local value = t[i]
8001
8002     if value.rendered ~= nil then
8003       content[#content + 1] = parse_content_part(content_part)
8004       content_part = {}
8005       content[#content + 1] = value.rendered
8006       value.rendered = nil
8007     else
8008       if value.type == "delimiter" and value.element == "emphasis" then
8009         if value.is_active then
8010           content_part[#content_part + 1] = string.rep(value.character, value.curren
8011         end
8012       else
8013         content_part[#content_part + 1] = value.content
8014       end
8015       value.content = ''
8016       value.is_active = false
8017     end
8018   end
8019
8020   if next(content_part) ~= nil then
8021     content[#content + 1] = parse_content_part(content_part)
8022   end
8023
8024   return content
8025 end
8026
```

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

```
8027 local function fill_emph(t, opening_index, closing_index)
8028     local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
8029     t[opening_index + 1].is_active = true
8030     t[opening_index + 1].rendered = writer.emphasis(content)
8031 end
8032
```

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

```
8033 local function fill_strong(t, opening_index, closing_index)
8034     local content = collect_emphasis_content(t, opening_index + 1, closing_index - 1)
8035     t[opening_index + 1].is_active = true
8036     t[opening_index + 1].rendered = writer.strong(content)
8037 end
8038
```

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

```
8039 local function breaks_three_rule(opening_delimiter, closing_delimiter)
8040     return (opening_delimiter.is_closing or closing_delimiter.is_opening) and
8041         ((opening_delimiter.original_count + closing_delimiter.original_count) % 3 == 0)
8042         (opening_delimiter.original_count % 3 ~= 0 or closing_delimiter.original_count % 3 == 0)
8043 end
8044
```

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`.

```
8045 local function find_emphasis_opener(t, bottom_index, latest_index, character, closing_delimiter)
8046     for i = latest_index, bottom_index, -1 do
8047         local value = t[i]
8048         if value.is_active and
8049             value.is_opening and
8050             value.type == "delimiter" and
8051             value.element == "emphasis" and
8052             (value.character == character) and
8053             (value.current_count > 0) then
8054             if not breaks_three_rule(value, closing_delimiter) then
8055                 return i
8056             end
8057         end
8058     end
8059 end
8060
```

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

```
8061 local function process_emphasis(t, opening_index, closing_index)
8062   for i = opening_index, closing_index do
8063     local value = t[i]
8064     if value.type == "delimiter" and value.element == "emphasis" then
8065       local delimiter_length = string.len(value.content)
8066       value.character = string.sub(value.content, 1, 1)
8067       value.current_count = delimiter_length
8068       value.original_count = delimiter_length
8069     end
8070   end
8071
8072   local openers_bottom = {
8073     ['*'] = {
8074       [true] = {opening_index, opening_index, opening_index},
8075       [false] = {opening_index, opening_index, opening_index}
8076     },
8077     ['_'] = {
8078       [true] = {opening_index, opening_index, opening_index},
8079       [false] = {opening_index, opening_index, opening_index}
8080     }
8081   }
8082
8083   local current_position = opening_index
8084   local max_position = closing_index
8085
8086   while current_position <= max_position do
8087     local value = t[current_position]
8088
8089     if value.type ~= "delimiter" or
8090        value.element ~= "emphasis" or
8091        not value.is_active or
8092        not value.is_closing or
8093        (value.current_count <= 0) then
8094       current_position = current_position + 1
8095       goto continue
8096     end
8097
8098     local character = value.character
8099     local is_opening = value.is_opening
8100     local closing_length_modulo_three = value.original_count % 3
8101
8102     local current_openers_bottom = openers_bottom[character][is_opening][closing_length_modulo_three]
8103
8104     local opener_position = find_emphasis_opener(t, current_openers_bottom, current_position)
8105
```

```

8106     if (opener_position == nil) then
8107         openers_bottom[character][is_opening][closing_length_modulo_three + 1] = current_position
8108         current_position = current_position + 1
8109         goto continue
8110     end
8111
8112     local opening_delimiter = t[opener_position]
8113
8114     local current_opening_count = opening_delimiter.current_count
8115     local current_closing_count = t[current_position].current_count
8116
8117     if (current_opening_count >= 2) and (current_closing_count >= 2) then
8118         opening_delimiter.current_count = current_opening_count - 2
8119         t[current_position].current_count = current_closing_count - 2
8120         fill_strong(t, opener_position, current_position)
8121     else
8122         opening_delimiter.current_count = current_opening_count - 1
8123         t[current_position].current_count = current_closing_count - 1
8124         fill_emph(t, opener_position, current_position)
8125     end
8126
8127     ::continue::
8128 end
8129 end
8130
8131 local cont = lpeg.R("\128\191") -- continuation byte
8132

```

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

```

8133 local function utf8_by_byte_count(n)
8134     if (n == 1) then
8135         return lpeg.R("\0\127")
8136     end
8137     if (n == 2) then
8138         return lpeg.R("\194\223") * cont
8139     end
8140     if (n == 3) then
8141         return lpeg.R("\224\239") * cont * cont
8142     end
8143     if (n == 4) then
8144         return lpeg.R("\240\244") * cont * cont * cont
8145     end
8146 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`.

```

8147 local function check_unicode_type(s, i, start_pos, end_pos, chartype)
8148     local c

```

```

8149     local char_length
8150     for pos = start_pos, end_pos, 1 do
8151         if (start_pos < 0) then
8152             char_length = -pos
8153         else
8154             char_length = pos + 1
8155         end
8156
8157         if (chartype == "punctuation") then
8158             if lpeg.match(parsers.punctuation[char_length], s, i+pos) then
8159                 return i
8160             end
8161         else
8162             c = lpeg.match({ C(utf8_by_byte_count(char_length)) },s,i+pos)
8163             if (c ~= nil) and (unicode.utf8.match(c, chartype)) then
8164                 return i
8165             end
8166         end
8167     end
8168 end
8169
8170 local function check_preceding_unicode_punctuation(s, i)
8171     return check_unicode_type(s, i, -4, -1, "punctuation")
8172 end
8173
8174 local function check_preceding_unicode_whitespace(s, i)
8175     return check_unicode_type(s, i, -4, -1, "%s")
8176 end
8177
8178 local function check_following_unicode_punctuation(s, i)
8179     return check_unicode_type(s, i, 0, 3, "punctuation")
8180 end
8181
8182 local function check_following_unicode_whitespace(s, i)
8183     return check_unicode_type(s, i, 0, 3, "%s")
8184 end
8185
8186 parsers.unicode_preceding_punctuation = B(parsers.escapable)
8187                                     + Cmt(parsers.succeed, check_preceding_unicode)
8188
8189 parsers.unicode_preceding_whitespace = Cmt(parsers.succeed, check_preceding_unicode)
8190
8191 parsers.unicode_following_punctuation = #parsers.escapable
8192                                     + Cmt(parsers.succeed, check_following_unicode)
8193
8194 parsers.unicode_following_whitespace = Cmt(parsers.succeed, check_following_unicode)
8195

```

```

8196 parsers.delimiter_run = function(character)
8197     return (B(parsers.backslash * character) + -B(character))
8198         * character^1
8199         * -#character
8200 end
8201
8202 parsers.left_flanking_delimiter_run = function(character)
8203     return (B( parsers.any)
8204         * (parsers.unicode_preceding_punctuation + parsers.unicode_preceding_wh
8205         + -B(parsers.any))
8206         * parsers.delimiter_run(character)
8207         * parsers.unicode_following_punctuation
8208         + parsers.delimiter_run(character)
8209         * -(parsers.unicode_following_punctuation + parsers.unicode_following_wh
8210         + parsers.eof)
8211 end
8212
8213 parsers.right_flanking_delimiter_run = function(character)
8214     return parsers.unicode_preceding_punctuation
8215         * parsers.delimiter_run(character)
8216         * (parsers.unicode_following_punctuation + parsers.unicode_following_whites
8217         + parsers.eof)
8218         + (B(parsers.any)
8219         * -(parsers.unicode_preceding_punctuation + parsers.unicode_preceding_whi
8220         * parsers.delimiter_run(character)
8221 end
8222
8223 if options.underscores then
8224     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8225         + (-#parsers.right_flanking_delimiter_run(parsers.underscore)
8226         + (parsers.unicode_preceding_punctuation
8227         * #parsers.right_flanking_delimiter_run(parsers.underscor
8228         * parsers.left_flanking_delimiter_run(parsers.underscore)
8229
8230     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8231         + (-#parsers.left_flanking_delimiter_run(parsers.underscore)
8232         + #(parsers.left_flanking_delimiter_run(parsers.underscore)
8233         * parsers.unicode_following_punctuation))
8234         * parsers.right_flanking_delimiter_run(parsers.underscore)
8235 else
8236     parsers.emph_start = parsers.left_flanking_delimiter_run(parsers.asterisk)
8237
8238     parsers.emph_end = parsers.right_flanking_delimiter_run(parsers.asterisk)
8239 end
8240
8241 parsers.emph_capturing_open_and_close = #parsers.emph_start * #parsers.emph_end
8242         * Ct( Cg(Cc("delimiter"), "type")

```

```

8243             * Cg(Cc("emphasis"), "element")
8244             * Cg(C(parsers.emph_start), "content")
8245             * Cg(Cc(true), "is_opening")
8246             * Cg(Cc(true), "is_closing")
8247
8248 parsers.emph_capturing_open = Ct( Cg(Cc("delimiter"), "type")
8249             * Cg(Cc("emphasis"), "element")
8250             * Cg(C(parsers.emph_start), "content")
8251             * Cg(Cc(true), "is_opening")
8252             * Cg(Cc(false), "is_closing"))
8253
8254 parsers.emph_capturing_close = Ct( Cg(Cc("delimiter"), "type")
8255             * Cg(Cc("emphasis"), "element")
8256             * Cg(C(parsers.emph_end), "content")
8257             * Cg(Cc(false), "is_opening")
8258             * Cg(Cc(true), "is_closing"))
8259
8260 parsers.emph_open_or_close = parsers.emph_capturing_open_and_close
8261             + parsers.emph_capturing_open
8262             + parsers.emph_capturing_close
8263
8264 parsers.emph_open = parsers.emph_capturing_open_and_close
8265             + parsers.emph_capturing_open
8266
8267 parsers.emph_close = parsers.emph_capturing_open_and_close
8268             + parsers.emph_capturing_close
8269

```

3.1.6.7 Helpers for Links and Link Reference Definitions (local)

```

8270 -- List of references defined in the document
8271 local references
8272
8273 -- List of note references defined in the document
8274 parsers.rawnotes = {}
8275

```

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.

```

8276 function self.register_link(_, tag, url, title,
8277                             attributes)
8278     local normalized_tag = self.normalize_tag(tag)
8279     if references[normalized_tag] == nil then
8280         references[normalized_tag] = {
8281             url = url,
8282             title = title,
8283             attributes = attributes

```



```

8284     }
8285     end
8286     return ""
8287 end
8288

```

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

```

8289 function self.lookup_reference(tag)
8290     return references[self.normalize_tag(tag)]
8291 end
8292

```

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

```

8293 function self.lookup_note_reference(tag)
8294     return parsers.rawnotes[self.normalize_tag(tag)]
8295 end
8296
8297 parsers.title_s_direct_ref = parsers.squote
8298                             * Cs((parsers.html_entities
8299                                 + (parsers.anyescaped - parsers.squote - parsers.bl
8300                                 * parsers.squote
8301
8302 parsers.title_d_direct_ref = parsers.dquote
8303                             * Cs((parsers.html_entities
8304                                 + (parsers.anyescaped - parsers.dquote - parsers.bl
8305                                 * parsers.dquote
8306
8307 parsers.title_p_direct_ref = parsers.lparent
8308                             * Cs((parsers.html_entities
8309                                 + (parsers.anyescaped - parsers.lparent - parsers.r
8310                                 * parsers.rparent
8311
8312 parsers.title_direct_ref = parsers.title_s_direct_ref
8313                          + parsers.title_d_direct_ref
8314                          + parsers.title_p_direct_ref
8315
8316 parsers.inline_direct_ref_inside = parsers.lparent * parsers.spnl
8317                                   * Cg(parsers.url + Cc(""), "url")
8318                                   * parsers.spnl
8319                                   * Cg(parsers.title_direct_ref + Cc(""), "title")
8320                                   * parsers.spnl * parsers.rparent
8321
8322 parsers.inline_direct_ref = parsers.lparent * parsers.spnlc
8323                           * Cg(parsers.url + Cc(""), "url")
8324                           * parsers.spnlc
8325                           * Cg(parsers.title + Cc(""), "title")
8326                           * parsers.spnlc * parsers.rparent

```

```

8327
8328 parsers.empty_link = parsers.lbracket
8329                      * parsers.rbracket
8330
8331 parsers.inline_link = parsers.link_text
8332                      * parsers.inline_direct_ref
8333
8334 parsers.full_link = parsers.link_text
8335                      * parsers.link_label
8336
8337 parsers.shortcut_link = parsers.link_label
8338                       * -(parsers.empty_link + parsers.link_label)
8339
8340 parsers.collapsed_link = parsers.link_label
8341                       * parsers.empty_link
8342
8343 parsers.image_opening = #(parsers.exclamation * parsers.inline_link)
8344                       * Cg(Cc("inline"), "link_type")
8345                       + #(parsers.exclamation * parsers.full_link)
8346                       * Cg(Cc("full"), "link_type")
8347                       + #(parsers.exclamation * parsers.collapsed_link)
8348                       * Cg(Cc("collapsed"), "link_type")
8349                       + #(parsers.exclamation * parsers.shortcut_link)
8350                       * Cg(Cc("shortcut"), "link_type")
8351                       + #(parsers.exclamation * parsers.empty_link)
8352                       * Cg(Cc("empty"), "link_type")
8353
8354 parsers.link_opening = #parsers.inline_link
8355                       * Cg(Cc("inline"), "link_type")
8356                       + #parsers.full_link
8357                       * Cg(Cc("full"), "link_type")
8358                       + #parsers.collapsed_link
8359                       * Cg(Cc("collapsed"), "link_type")
8360                       + #parsers.shortcut_link
8361                       * Cg(Cc("shortcut"), "link_type")
8362                       + #parsers.empty_link
8363                       * Cg(Cc("empty_link"), "link_type")
8364                       + #parsers.link_text
8365                       * Cg(Cc("link_text"), "link_type")
8366
8367 parsers.note_opening = #(parsers.circumflex * parsers.link_text)
8368                       * Cg(Cc("note_inline"), "link_type")
8369
8370 parsers.raw_note_opening = #( parsers.lbracket
8371                             * parsers.circumflex
8372                             * parsers.link_label_body
8373                             * parsers.rbracket)

```

```

8374             * Cg(Cc("raw_note"), "link_type")
8375
8376 local inline_note_element = Cg(Cc("note"), "element")
8377             * parsers.note_opening
8378             * Cg(parsers.circumflex * parsers.lbracket, "content")
8379
8380 local image_element = Cg(Cc("image"), "element")
8381             * parsers.image_opening
8382             * Cg(parsers.exclamation * parsers.lbracket, "content")
8383
8384 local note_element = Cg(Cc("note"), "element")
8385             * parsers.raw_note_opening
8386             * Cg(parsers.lbracket * parsers.circumflex, "content")
8387
8388 local link_element = Cg(Cc("link"), "element")
8389             * parsers.link_opening
8390             * Cg(parsers.lbracket, "content")
8391
8392 local opening_elements = parsers.fail
8393
8394 if options.inlineNotes then
8395     opening_elements = opening_elements + inline_note_element
8396 end
8397
8398 opening_elements = opening_elements + image_element
8399
8400 if options.notes then
8401     opening_elements = opening_elements + note_element
8402 end
8403
8404 opening_elements = opening_elements + link_element
8405
8406 parsers.link_image_opening = Ct( Cg(Cc("delimiter"), "type")
8407                               * Cg(Cc(true), "is_opening")
8408                               * Cg(Cc(false), "is_closing")
8409                               * opening_elements)
8410
8411 parsers.link_image_closing = Ct( Cg(Cc("delimiter"), "type")
8412                               * Cg(Cc("link"), "element")
8413                               * Cg(Cc(false), "is_opening")
8414                               * Cg(Cc(true), "is_closing")
8415                               * ( Cg(Cc(true), "is_direct")
8416                               * Cg(parsers.rbracket * #parsers.inline_direct_re
8417                               + Cg(Cc(false), "is_direct")
8418                               * Cg(parsers.rbracket, "content"))))
8419
8420 parsers.link_image_open_or_close = parsers.link_image_opening

```

```

8421             + parsers.link_image_closing
8422
8423     if options.html then
8424         parsers.link_emph_precedence = parsers.inticks
8425             + parsers.autolink
8426             + parsers.html_inline_tags
8427     else
8428         parsers.link_emph_precedence = parsers.inticks
8429             + parsers.autolink
8430     end
8431
8432     parsers.link_and_emph_endline = parsers.newline
8433         * ((parsers.check_minimal_indent
8434             * -V("EndlineExceptions")
8435             + parsers.check_optional_indent
8436             * -V("EndlineExceptions")
8437             * -parsers.starter) / "")
8438         * parsers.spacechar^0 / "\n"
8439
8440     parsers.link_and_emph_content = Ct( Cg(Cc("content"), "type")
8441         * Cg(Cs(( parsers.link_emph_precedence
8442             + parsers.backslash * parsers.any
8443             + parsers.link_and_emph_endline
8444             + (parsers.linechar
8445                 - parsers.blankline^2
8446                 - parsers.link_image_open_or_close
8447                 - parsers.emph_open_or_close))^0), "con
8448
8449     parsers.link_and_emph_table = (parsers.link_image_opening + parsers.emph_open)
8450         * parsers.link_and_emph_content
8451         * ((parsers.link_image_open_or_close + parsers.emph_ope
8452             * parsers.link_and_emph_content)^1
8453

```

Collect the content between the [opening_index](#) and [closing_index](#) in the delimiter table `t`.

```

8454     local function collect_link_content(t, opening_index, closing_index)
8455         local content = {}
8456         for i = opening_index, closing_index do
8457             content[#content + 1] = t[i].content
8458         end
8459         return util.ropetostring(content)
8460     end
8461

```

Look for the closest potential link opener in the delimiter table `t` in the range from [bottom_index](#) to [latest_index](#).

```

8462     local function find_link_opener(t, bottom_index, latest_index)

```

```

8463     for i = latest_index, bottom_index, -1 do
8464         local value = t[i]
8465         if value.type == "delimiter" and
8466             value.is_opening and
8467             (value.element == "link" or value.element == "image" or value.element == "no")
8468             and not value.removed then
8469             if value.is_active then
8470                 return i
8471             end
8472             value.removed = true
8473             return nil
8474         end
8475     end
8476 end
8477

```

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

```

8478 local function find_next_link_closing_index(t, latest_index)
8479     for i = latest_index, #t do
8480         local value = t[i]
8481         if value.is_closing and
8482             value.element == "link" and
8483             not value.removed then
8484             return i
8485         end
8486     end
8487 end
8488

```

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.

```

8489 local function disable_previous_link_openers(t, opening_index)
8490     if t[opening_index].element == "image" then
8491         return
8492     end
8493
8494     for i = opening_index, 1, -1 do
8495         local value = t[i]
8496         if value.is_active and
8497             value.type == "delimiter" and
8498             value.is_opening and
8499             value.element == "link" then
8500             value.is_active = false
8501         end
8502     end
8503 end
8504

```

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

```
8505 local function disable_range(t, opening_index, closing_index)
8506   for i = opening_index, closing_index do
8507     local value = t[i]
8508     if value.is_active then
8509       value.is_active = false
8510       if value.type == "delimiter" then
8511         value.removed = true
8512       end
8513     end
8514   end
8515 end
8516
```

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

```
8517 local function delete_parsed_content_in_range(t, opening_index, closing_index)
8518   for i = opening_index, closing_index do
8519     t[i].rendered = nil
8520   end
8521 end
8522
```

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

```
8523 local function empty_content_in_range(t, opening_index, closing_index)
8524   for i = opening_index, closing_index do
8525     t[i].content = ''
8526   end
8527 end
8528
```

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

```
8529 local function join_attributes(reference_attributes, own_attributes)
8530   local merged_attributes = {}
8531   for _, attribute in ipairs(reference_attributes or {}) do
8532     table.insert(merged_attributes, attribute)
8533   end
8534   for _, attribute in ipairs(own_attributes or {}) do
8535     table.insert(merged_attributes, attribute)
8536   end
8537   if next(merged_attributes) == nil then
8538     merged_attributes = nil
8539   end
8540   return merged_attributes
8541 end
```

8542

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

```
8543 local function render_link_or_image(t, opening_index, closing_index, content_end_index)
8544     process_emphasis(t, opening_index, content_end_index)
8545     local mapped = collect_emphasis_content(t, opening_index + 1, content_end_index)
8546
8547     local rendered = {}
8548     if (t[opening_index].element == "link") then
8549         rendered = writer.link(mapped, reference.url, reference.title, reference.attributes)
8550     end
8551
8552     if (t[opening_index].element == "image") then
8553         rendered = writer.image(mapped, reference.url, reference.title, reference.attributes)
8554     end
8555
8556     if (t[opening_index].element == "note") then
8557         if (t[opening_index].link_type == "note_inline") then
8558             rendered = writer.note(mapped)
8559         end
8560         if (t[opening_index].link_type == "raw_note") then
8561             rendered = writer.note(reference)
8562         end
8563     end
8564
8565     t[opening_index].rendered = rendered
8566     delete_parsed_content_in_range(t, opening_index + 1, closing_index)
8567     empty_content_in_range(t, opening_index, closing_index)
8568     disable_previous_link_openers(t, opening_index)
8569     disable_range(t, opening_index, closing_index)
8570 end
8571
```

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.

```
8572 local function resolve_inline_following_content(t, closing_index, match_reference,
8573     local content = ""
8574     for i = closing_index + 1, #t do
8575         content = content .. t[i].content
8576     end
8577
8578     local matching_content = parsers.succeed
8579
8580     if match_reference then
8581         matching_content = matching_content * parsers.inline_direct_ref_inside
8582     end
end
```

```

8583
8584     if match_link_attributes then
8585         matching_content = matching_content * Cg(Ct(parsers.attributes^-
1), "attributes")
8586     end
8587
8588     local matched = lpeg.match(Ct(matching_content * Cg(Cp(), "end_position")), conte
8589
8590     local matched_count = matched.end_position - 1
8591     for i = closing_index + 1, #t do
8592         local value = t[i]
8593
8594         local chars_left = matched_count
8595         matched_count = matched_count - #value.content
8596
8597         if matched_count <= 0 then
8598             value.content = value.content:sub(chars_left + 1)
8599             break
8600         end
8601
8602         value.content = ''
8603         value.is_active = false
8604     end
8605
8606     local attributes = matched.attributes
8607     if attributes == nil or next(attributes) == nil then
8608         attributes = nil
8609     end
8610
8611     return {
8612         url = matched.url or "",
8613         title = matched.title or "",
8614         attributes = attributes
8615     }
8616 end
8617

```

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.

```

8618     local function resolve_inline_link(t, opening_index, closing_index)
8619         local inline_content = resolve_inline_following_content(t, closing_index, true, t
8620         render_link_or_image(t, opening_index, closing_index, closing_index, inline_conte
8621     end
8622

```


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

```
8623 local function resolve_note_inline_link(t, opening_index, closing_index)
8624     local inline_content = resolve_inline_following_content(t, closing_index, false,
8625     render_link_or_image(t, opening_index, closing_index, closing_index, inline_content)
8626 end
8627
```

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.

```
8628 local function resolve_shortcut_link(t, opening_index, closing_index)
8629     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8630     local r = self.lookup_reference(content)
8631
8632     if r then
8633         local inline_content = resolve_inline_following_content(t, closing_index, false,
8634         r.attributes = join_attributes(r.attributes, inline_content.attributes)
8635         render_link_or_image(t, opening_index, closing_index, closing_index, r)
8636     end
8637 end
8638
```

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.

```
8639 local function resolve_raw_note_link(t, opening_index, closing_index)
8640     local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8641     local r = self.lookup_note_reference(content)
8642
8643     if r then
8644         local parsed_ref = self.parser_functions.parse_blocks_nested(r)
8645         render_link_or_image(t, opening_index, closing_index, closing_index, parsed_ref)
8646     end
8647 end
8648
```

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.

```
8649 local function resolve_full_link(t, opening_index, closing_index)
8650     local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8651     local next_link_content = collect_link_content(t, closing_index + 3, next_link_closing_index)
8652     local r = self.lookup_reference(next_link_content)
8653
8654     if r then
8655         local inline_content = resolve_inline_following_content(t, next_link_closing_index,
8656         t.match_link_attributes
8657         r.attributes = join_attributes(r.attributes, inline_content.attributes)

```

```

8658     render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8659     end
8660     end
8661

```

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.

```

8662     local function resolve_collapsed_link(t, opening_index, closing_index)
8663         local next_link_closing_index = find_next_link_closing_index(t, closing_index + 4)
8664         local content = collect_link_content(t, opening_index + 1, closing_index - 1)
8665         local r = self.lookup_reference(content)
8666
8667         if r then
8668             local inline_content = resolve_inline_following_content(t, closing_index, false)
8669             r.attributes = join_attributes(r.attributes, inline_content.attributes)
8670             render_link_or_image(t, opening_index, next_link_closing_index, closing_index,
8671             end
8672         end
8673

```

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.

```

8674     local function process_links_and_emphasis(t)
8675         for _,value in ipairs(t) do
8676             value.is_active = true
8677         end
8678
8679         for i,value in ipairs(t) do
8680             if not value.is_closing
8681                 or value.type ~= "delimiter"
8682                 or not (value.element == "link" or value.element == "image" or value.element == "strong")
8683                 or value.removed then
8684                 goto continue
8685             end
8686
8687             local opener_position = find_link_opener(t, 1, i - 1)
8688             if (opener_position == nil) then
8689                 goto continue
8690             end
8691
8692             local opening_delimiter = t[opener_position]
8693             opening_delimiter.removed = true
8694
8695             local link_type = opening_delimiter.link_type

```

```

8696
8697     if (link_type == "inline") then
8698         resolve_inline_link(t, opener_position, i)
8699     end
8700     if (link_type == "shortcut") then
8701         resolve_shortcut_link(t, opener_position, i)
8702     end
8703     if (link_type == "full") then
8704         resolve_full_link(t, opener_position, i)
8705     end
8706     if (link_type == "collapsed") then
8707         resolve_collapsed_link(t, opener_position, i)
8708     end
8709     if (link_type == "note_inline") then
8710         resolve_note_inline_link(t, opener_position, i)
8711     end
8712     if (link_type == "raw_note") then
8713         resolve_raw_note_link(t, opener_position, i)
8714     end
8715
8716     ::continue::
8717 end
8718
8719 t[#t].content = t[#t].content:gsub("%s*$", "")
8720
8721 process_emphasis(t, 1, #t)
8722 local final_result = collect_emphasis_content(t, 1, #t)
8723 return final_result
8724 end
8725
8726 function self.defer_link_and_emphasis_processing(delimiter_table)
8727     return writer.defer_call(function()
8728         return process_links_and_emphasis(delimiter_table)
8729     end)
8730 end
8731

```

3.1.6.8 Inline Elements (local)

```

8732 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
8733                 / writer.string
8734
8735 parsers.Symbol   = (parsers.backtick^1 + V("SpecialChar"))
8736                 / writer.string
8737
8738 parsers.Ellipsis = P("...") / writer.ellipsis
8739

```

```

8740 parsers.Smart      = parsers.Ellipsis
8741
8742 parsers.Code        = parsers.inticks / writer.code
8743
8744 if options.blankBeforeBlockquote then
8745     parsers.bqstart = parsers.fail
8746 else
8747     parsers.bqstart = parsers.blockquote_start
8748 end
8749
8750 if options.blankBeforeHeading then
8751     parsers.headerstart = parsers.fail
8752 else
8753     parsers.headerstart = parsers.atx_heading
8754 end
8755
8756 if options.blankBeforeList then
8757     parsers.interrupting_bullets = parsers.fail
8758     parsers.interrupting_enumerators = parsers.fail
8759 else
8760     parsers.interrupting_bullets = parsers.bullet(parsers.dash, true)
8761                                   + parsers.bullet(parsers.asterisk, true)
8762                                   + parsers.bullet(parsers.plus, true)
8763
8764     parsers.interrupting_enumerators = parsers.enumerator(parsers.period, true)
8765                                       + parsers.enumerator(parsers.rparent, true)
8766 end
8767
8768 if options.html then
8769     parsers.html_interrupting = parsers.check_trail
8770                                   * ( parsers.html_incomplete_open_tag
8771                                       + parsers.html_incomplete_close_tag
8772                                       + parsers.html_incomplete_open_special_tag
8773                                       + parsers.html_comment_start
8774                                       + parsers.html_cdatasection_start
8775                                       + parsers.html_declaration_start
8776                                       + parsers.html_instruction_start
8777                                       - parsers.html_close_special_tag
8778                                       - parsers.html_empty_special_tag)
8779 else
8780     parsers.html_interrupting = parsers.fail
8781 end
8782
8783 parsers.EndlineExceptions
8784     = parsers.blankline -- paragraph break
8785     + parsers.eof       -- end of document
8786     + parsers.bqstart

```

```

8787         + parsers.thematic_break_lines
8788         + parsers.interrupting_bullets
8789         + parsers.interrupting_enumerators
8790         + parsers.headerstart
8791         + parsers.html_interrupting
8792
8793 parsers.NoSoftLineBreakEndlineExceptions = parsers.EndlineExceptions
8794
8795 parsers.endline = parsers.newline
8796         * (parsers.check_minimal_indent
8797         * -V("EndlineExceptions")
8798         + parsers.check_optional_indent
8799         * -V("EndlineExceptions")
8800         * -parsers.starter)
8801         * parsers.spacechar^0
8802
8803 parsers.Endline = parsers.endline
8804         / writer.soft_line_break
8805
8806 parsers.EndlineNoSub = parsers.endline
8807
8808 parsers.NoSoftLineBreakEndline
8809         = parsers.newline
8810         * (parsers.check_minimal_indent
8811         * -V("NoSoftLineBreakEndlineExceptions")
8812         + parsers.check_optional_indent
8813         * -V("NoSoftLineBreakEndlineExceptions")
8814         * -parsers.starter)
8815         * parsers.spacechar^0
8816         / writer.space
8817
8818 parsers.EndlineBreak = parsers.backslash * parsers.Endline
8819         / writer.hard_line_break
8820
8821 parsers.OptionalIndent
8822         = parsers.spacechar^1 / writer.space
8823
8824 parsers.Space = parsers.spacechar^2 * parsers.Endline
8825         / writer.hard_line_break
8826         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8827         + parsers.spacechar^1 * parsers.Endline
8828         / writer.soft_line_break
8829         + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8830
8831 parsers.NoSoftLineBreakSpace
8832         = parsers.spacechar^2 * parsers.Endline
8833         / writer.hard_line_break

```

```

8834         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / self.
8835         + parsers.spacechar^1 * parsers.Endline
8836                                 / writer.soft_line_break
8837         + parsers.spacechar^1 * -parsers.newline / self.expandtabs
8838
8839 parsers.NonbreakingEndline
8840         = parsers.endline
8841         / writer.soft_line_break
8842
8843 parsers.NonbreakingSpace
8844         = parsers.spacechar^2 * parsers.Endline
8845                                 / writer.hard_line_break
8846         + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
8847         + parsers.spacechar^1 * parsers.Endline
8848                                 * parsers.optionalspace
8849                                 / writer.soft_line_break
8850         + parsers.spacechar^1 * parsers.optionalspace
8851                                 / writer.nbsp
8852

```

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.

```

8853 function self.auto_link_url(url, attributes)
8854   return writer.link(writer.escape(url),
8855                     url, nil, attributes)
8856 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.

```

8857 function self.auto_link_email(email, attributes)
8858   return writer.link(writer.escape(email),
8859                     "mailto:".email,
8860                     nil, attributes)
8861 end
8862
8863 parsers.AutoLinkUrl = parsers.auto_link_url
8864                     / self.auto_link_url
8865
8866 parsers.AutoLinkEmail
8867         = parsers.auto_link_email
8868         / self.auto_link_email
8869
8870 parsers.AutoLinkRelativeReference
8871         = parsers.auto_link_relative_reference
8872         / self.auto_link_url

```

```

8873
8874 parsers.LinkAndEmph = Ct(parsers.link_and_emph_table)
8875                       / self.defer_link_and_emphasis_processing
8876
8877 parsers.EscapedChar  = parsers.backslash * C(parsers.escapable) / writer.string
8878
8879 parsers.InlineHtml   = Cs(parsers.html_inline_comment) / writer.inline_html_comment
8880                       + Cs(parsers.html_any_empty_inline_tag
8881                             + parsers.html_inline_instruction
8882                             + parsers.html_inline_cdatasection
8883                             + parsers.html_inline_declaration
8884                             + parsers.html_any_open_inline_tag
8885                             + parsers.html_any_close_tag)
8886                       / writer.inline_html_tag
8887
8888 parsers.HtmlEntity   = parsers.html_entities / writer.string

```

3.1.6.9 Block Elements (local)

```

8889 parsers.DisplayHtml = Cs(parsers.check_trail
8890                          * ( parsers.html_comment
8891                              + parsers.html_special_block
8892                              + parsers.html_block
8893                              + parsers.html_any_block
8894                              + parsers.html_instruction
8895                              + parsers.html_cdatasection
8896                              + parsers.html_declaration))
8897                          / writer.block_html_element
8898
8899 parsers.indented_non_blank_line = parsers.indentedline - parsers.blankline
8900
8901 parsers.Verbatim = Cs(
8902     parsers.check_code_trail
8903     * (parsers.line - parsers.blankline)
8904     * ((parsers.check_minimal_blank_indent_and_full_code_trail * pa
8905         * ((parsers.check_minimal_indent / "") * parsers.check_code_t
8906           * (parsers.line - parsers.blankline))^1)^0
8907     ) / self.expandtabs / writer.verbatim
8908
8909 parsers.Blockquote = parsers.blockquote_body
8910                    / writer.blockquote
8911
8912 parsers.ThematicBreak = parsers.thematic_break_lines
8913                       / writer.thematic_break
8914
8915 parsers.Reference = parsers.define_reference_parser
8916                   / self.register_link

```

```

8917
8918 parsers.Paragraph = parsers.freeze_trail
8919                   * (Ct((parsers.Inline)^1)
8920                   * (parsers.newline + parsers.eof)
8921                   * parsers.unfreeze_trail
8922                   / writer.paragraph)
8923
8924 parsers.Plain      = parsers.nonindentspace * Ct(parsers.Inline^1)
8925                   / writer.plain

```

3.1.6.10 Lists (local)

```

8926
8927 if options.taskLists then
8928   parsers.tickbox = ( parsers.ticked_box
8929                     + parsers.halfticked_box
8930                     + parsers.unticked_box
8931                     ) / writer.tickbox
8932 else
8933   parsers.tickbox = parsers.fail
8934 end
8935
8936 parsers.list_blank = parsers.conditionally_indented_blankline
8937
8938 parsers.ref_or_block_list_separated = parsers.sep_group_no_output(parsers.list_blank
8939                           * parsers.minimally_indented_ref
8940                           + parsers.block_sep_group(parsers.list_blank)
8941                           * parsers.minimally_indented_block)
8942
8943 parsers.ref_or_block_non_separated = parsers.minimally_indented_ref
8944                                     + (parsers.succeed / writer.interblocksep)
8945                                     * parsers.minimally_indented_block
8946                                     - parsers.minimally_indented_blankline
8947
8948 parsers.tight_list_loop_body_pair =
8949   parsers.create_loop_body_pair(parsers.ref_or_block_non_separated,
8950                                 parsers.minimally_indented_par_or_plain_no_blank,
8951                                 (parsers.succeed / writer.interblocksep),
8952                                 (parsers.succeed / writer.paragraphsep))
8953
8954 parsers.loose_list_loop_body_pair =
8955   parsers.create_loop_body_pair(parsers.ref_or_block_list_separated,
8956                                 parsers.minimally_indented_par_or_plain,
8957                                 parsers.block_sep_group(parsers.list_blank),
8958                                 parsers.par_sep_group(parsers.list_blank))
8959
8960 parsers.tight_list_content_loop = V("Block")

```



```

8961         * parsers.tight_list_loop_body_pair.block^0
8962         + (V("Paragraph") + V("Plain"))
8963         * parsers.ref_or_block_non_separated
8964         * parsers.tight_list_loop_body_pair.block^0
8965         + (V("Paragraph") + V("Plain"))
8966         * parsers.tight_list_loop_body_pair.par^0
8967
8968 parsers.loose_list_content_loop = V("Block")
8969         * parsers.loose_list_loop_body_pair.block^0
8970         + (V("Paragraph") + V("Plain"))
8971         * parsers.ref_or_block_list_separated
8972         * parsers.loose_list_loop_body_pair.block^0
8973         + (V("Paragraph") + V("Plain"))
8974         * parsers.loose_list_loop_body_pair.par^0
8975
8976 parsers.list_item_tightness_condition = -( parsers.list_blank^0
8977         * parsers.minimally_indented_ref_or_block
8978         * remove_indent("li")
8979         + remove_indent("li")
8980         * parsers.fail
8981
8982 parsers.indented_content_tight = Ct( (parsers.blankline / "")
8983         * #parsers.list_blank
8984         * remove_indent("li")
8985         + ( (V("Reference") + (parsers.blankline / ""))
8986         * parsers.check_minimal_indent
8987         * parsers.tight_list_content_loop
8988         + (V("Reference") + (parsers.blankline / ""))
8989         + (parsers.checkbox^-1 / writer.escape)
8990         * parsers.tight_list_content_loop
8991         )
8992         * parsers.list_item_tightness_condition
8993         )
8994
8995 parsers.indented_content_loose = Ct( (parsers.blankline / "")
8996         * #parsers.list_blank
8997         + ( (V("Reference") + (parsers.blankline / ""))
8998         * parsers.check_minimal_indent
8999         * parsers.loose_list_content_loop
9000         + (V("Reference") + (parsers.blankline / ""))
9001         + (parsers.checkbox^-1 / writer.escape)
9002         * parsers.loose_list_content_loop
9003         )
9004         )
9005
9006 parsers.TightListItem = function(starter)
9007     return -parsers.ThematicBreak

```

```

9008         * parsers.add_indent(starter, "li")
9009         * parsers.indented_content_tight
9010     end
9011
9012     parsers.LooseListItem = function(starter)
9013         return -parsers.ThematicBreak
9014         * parsers.add_indent(starter, "li")
9015         * parsers.indented_content_loose
9016         * remove_indent("li")
9017     end
9018
9019     parsers.BulletListOfType = function(bullet_type)
9020         local bullet = parsers.bullet(bullet_type)
9021         return ( Ct( parsers.TightListItem(bullet)
9022             * ( (parsers.check_minimal_indent / "")
9023                 * parsers.TightListItem(bullet)
9024             )^0
9025         )
9026         * Cc(true)
9027         * -#( (parsers.list_blank^0 / "")
9028             * parsers.check_minimal_indent
9029             * (bullet - parsers.ThematicBreak)
9030         )
9031         + Ct( parsers.LooseListItem(bullet)
9032             * ( (parsers.list_blank^0 / "")
9033                 * (parsers.check_minimal_indent / "")
9034                 * parsers.LooseListItem(bullet)
9035             )^0
9036         )
9037         * Cc(false)
9038         ) / writer.bulletlist
9039     end
9040
9041     parsers.BulletList = parsers.BulletListOfType(parsers.dash)
9042         + parsers.BulletListOfType(parsers.asterisk)
9043         + parsers.BulletListOfType(parsers.plus)
9044
9045     local function ordered_list(items,tight,starter)
9046         local startnum = starter[2][1]
9047         if options.startNumber then
9048             startnum = tonumber(startnum) or 1 -- fallback for '#'
9049             if startnum ~= nil then
9050                 startnum = math.floor(startnum)
9051             end
9052         else
9053             startnum = nil
9054         end

```

```

9055     return writer.orderedlist(items,tight,startnum)
9056 end
9057
9058 parsers.OrderedListOfType = function(delimiter_type)
9059     local enumerator = parsers.enumerator(delimiter_type)
9060     return Cg(enumerator, "listtype")
9061         * (Ct( parsers.TightListItem(Cb("listtype"))
9062             * ((parsers.check_minimal_indent / "") * parsers.TightListItem(enumerat
9063             * Cc(true)
9064             * -#((parsers.list_blank^0 / "")
9065                 * parsers.check_minimal_indent * enumerator)
9066         + Ct( parsers.LooseListItem(Cb("listtype"))
9067             * ((parsers.list_blank^0 / "")
9068                 * (parsers.check_minimal_indent / "") * parsers.LooseListItem(enumerat
9069             * Cc(false)
9070         ) * Ct(Cb("listtype")) / ordered_list
9071 end
9072
9073 parsers.OrderedList = parsers.OrderedListOfType(parsers.period)
9074     + parsers.OrderedListOfType(parsers.rparent)

```

3.1.6.11 Blank (local)

```

9075 parsers.Blank           = parsers.blankline / ""
9076                       + V("Reference")

```

3.1.6.12 Headings (local)

```

9077 function parsers.parse_heading_text(s)
9078     local inlines = self.parser_functions.parse_inlines(s)
9079     local flatten_inlines = self.writer.flatten_inlines
9080     self.writer.flatten_inlines = true
9081     local flat_text = self.parser_functions.parse_inlines(s)
9082     flat_text = util.ropetostring(flat_text)
9083     self.writer.flatten_inlines = flatten_inlines
9084     return {flat_text, inlines}
9085 end
9086
9087 -- parse atx header
9088 parsers.AtxHeading = parsers.check_trail_no_rem
9089                     * Cg(parsers.heading_start, "level")
9090                     * ((C( parsers.optionalspace
9091                         * parsers.hash^0
9092                         * parsers.optionalspace
9093                         * parsers.newline)
9094                     + parsers.spacechar^1
9095                     * C(parsers.line))
9096                     / strip_atx_end

```

```

9097             / parsers.parse_heading_text)
9098         * Cb("level")
9099         / writer.heading
9100
9101     parsers.heading_line = parsers.linechar^1
9102                         - parsers.thematic_break_lines
9103
9104     parsers.heading_text = parsers.heading_line
9105                         * ((V("Endline") / "\n") * (parsers.heading_line - parsers.heading_line)
9106                         * parsers.newline^-1
9107
9108     parsers.SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
9109                         * #(parsers.heading_text
9110                         * parsers.check_minimal_indent * parsers.check_trail * parsers.heading_line)
9111                         * Cs(parsers.heading_text)
9112                         / parsers.parse_heading_text
9113                         * parsers.check_minimal_indent_and_trail * parsers.heading_line
9114                         * parsers.newline
9115                         * parsers.unfreeze_trail
9116                         / writer.heading
9117
9118     parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

3.1.6.13 Syntax Specification

Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain $\text{T}_{\text{E}}\text{X}$ output.

```

9119     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.

```

9120     local walkable_syntax = (function(global_walkable_syntax)
9121         local local_walkable_syntax = {}
9122         for lhs, rule in pairs(global_walkable_syntax) do
9123             local_walkable_syntax[lhs] = util.table_copy(rule)
9124         end
9125         return local_walkable_syntax
9126     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.

```

9127     local current_extension_name = nil

```

```

9128 self.insert_pattern = function(selector, pattern, pattern_name)
9129   assert(pattern_name == nil or type(pattern_name) == "string")
9130   local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
9131   assert(lhs ~= nil,
9132     [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
9133     .. selector .. ["])
9134   assert(walkable_syntax[lhs] ~= nil,
9135     [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
9136   assert(pos == "before" or pos == "after" or pos == "instead of",
9137     [[Expected positional specifier "before", "after", or "instead of", not "]]
9138     .. pos .. ["])
9139   local rule = walkable_syntax[lhs]
9140   local index = nil
9141   for current_index, current_rhs in ipairs(rule) do
9142     if type(current_rhs) == "string" and current_rhs == rhs then
9143       index = current_index
9144       if pos == "after" then
9145         index = index + 1
9146       end
9147       break
9148     end
9149   end
9150   assert(index ~= nil,
9151     [[Rule ]] .. lhs .. [[ -> ]] .. rhs
9152     .. [[ does not exist in markdown grammar]])
9153   local accountable_pattern
9154   if current_extension_name then
9155     accountable_pattern = { pattern, current_extension_name, pattern_name }
9156   else
9157     assert(type(pattern) == "string",
9158       [[reader->insert_pattern() was called outside an extension with ]]
9159       .. [[a PEG pattern instead of a rule name]])
9160     accountable_pattern = pattern
9161   end
9162   if pos == "instead of" then
9163     rule[index] = accountable_pattern
9164   else
9165     table.insert(rule, index, accountable_pattern)
9166   end
9167 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.

```

9168 local syntax =
9169   { "Blocks",
9170     Blocks = V("InitializeState")

```

```

9172 * ( V("ExpectedJekyllData")
9173 * ( V("Blank")^0 / writer.interblocksep)
9174 )^-1
9175 * 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.

```

9176 * ( V("Block")
9177 * ( V("Blank")^0 * parsers.eof
9178 + ( V("Blank")^2 / writer.paragraphsep
9179 + V("Blank")^0 / writer.interblocksep
9180 )
9181 )
9182 + ( V("Paragraph") + V("Plain") )
9183 * ( V("Blank")^0 * parsers.eof
9184 + ( V("Blank")^2 / writer.paragraphsep
9185 + V("Blank")^0 / writer.interblocksep
9186 )
9187 )
9188 * V("Block")
9189 * ( V("Blank")^0 * parsers.eof
9190 + ( V("Blank")^2 / writer.paragraphsep
9191 + V("Blank")^0 / writer.interblocksep
9192 )
9193 )
9194 + ( V("Paragraph") + V("Plain") )
9195 * ( V("Blank")^0 * parsers.eof
9196 + V("Blank")^0 / writer.paragraphsep
9197 )
9198 )^0,
9199
9200 ExpectedJekyllData = parsers.fail,
9201
9202 Blank = parsers.Blank,
9203 Reference = parsers.Reference,
9204
9205 Blockquote = parsers.Blockquote,
9206 Verbatim = parsers.Verbatim,
9207 ThematicBreak = parsers.ThematicBreak,
9208 BulletList = parsers.BulletList,
9209 OrderedList = parsers.OrderedList,
9210 DisplayHtml = parsers.DisplayHtml,
9211 Heading = parsers.Heading,
9212 Paragraph = parsers.Paragraph,
9213 Plain = parsers.Plain,
9214

```

```

9215     EndlineExceptions      = parsers.EndlineExceptions,
9216     NoSoftLineBreakEndlineExceptions
9217                               = parsers.NoSoftLineBreakEndlineExceptions,
9218
9219     Str                      = parsers.Str,
9220     Space                    = parsers.Space,
9221     NoSoftLineBreakSpace    = parsers.NoSoftLineBreakSpace,
9222     OptionalIndent          = parsers.OptionalIndent,
9223     Endline                  = parsers.Endline,
9224     EndlineNoSub            = parsers.EndlineNoSub,
9225     NoSoftLineBreakEndline
9226                               = parsers.NoSoftLineBreakEndline,
9227     EndlineBreak            = parsers.EndlineBreak,
9228     LinkAndEmph             = parsers.LinkAndEmph,
9229     Code                     = parsers.Code,
9230     AutoLinkUrl             = parsers.AutoLinkUrl,
9231     AutoLinkEmail           = parsers.AutoLinkEmail,
9232     AutoLinkRelativeReference
9233                               = parsers.AutoLinkRelativeReference,
9234     InlineHtml              = parsers.InlineHtml,
9235     HtmlEntity              = parsers.HtmlEntity,
9236     EscapedChar             = parsers.EscapedChar,
9237     Smart                    = parsers.Smart,
9238     Symbol                   = parsers.Symbol,
9239     SpecialChar             = parsers.fail,
9240     InitializeState         = parsers.succeed,
9241 }

```

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]`.

```

9242     self.update_rule = function(rule_name, get_pattern)
9243         assert(current_extension_name ~= nil)
9244         assert(syntax[rule_name] ~= nil,
9245             [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
9246         local previous_pattern
9247         local extension_name
9248         if walkable_syntax[rule_name] then
9249             local previous_accountable_pattern = walkable_syntax[rule_name][1]
9250             previous_pattern = previous_accountable_pattern[1]
9251             extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
9252         else
9253             previous_pattern = nil
9254             extension_name = current_extension_name
9255         end

```

```
9256         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
```

```
9257         if type(get_pattern) == "function" then
9258             pattern = get_pattern(previous_pattern)
9259         else
9260             assert(previous_pattern == nil,
9261                 [[Rule ]] .. rule_name ..
9262                 [[ has already been updated by ]] .. extension_name)
9263             pattern = get_pattern
9264         end
9265         local accountable_pattern = { pattern, extension_name, rule_name }
9266         walkable_syntax[rule_name] = { accountable_pattern }
9267     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.

```
9268         local special_characters = {}
9269         self.add_special_character = function(c)
9270             table.insert(special_characters, c)
9271             syntax.SpecialChar = S(table.concat(special_characters, ""))
9272         end
9273
9274         self.add_special_character("*")
9275         self.add_special_character("[")
9276         self.add_special_character("]")
9277         self.add_special_character("<")
9278         self.add_special_character("!")
9279         self.add_special_character("\\")
```

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

```
9280         self.initialize_named_group = function(name, value)
9281             local pattern = Ct("")
9282             if value ~= nil then
9283                 pattern = pattern / value
9284             end
```



```

9285     syntax.InitializeState = syntax.InitializeState
9286         * Cg(pattern, name)
9287     end

```

Add a named group for indentation.

```

9288     self.initialize_named_group("indent_info")

```

Apply syntax extensions.

```

9289     for _, extension in ipairs(extensions) do
9290         current_extension_name = extension.name
9291         extension.extend_writer(writer)
9292         extension.extend_reader(self)
9293     end
9294     current_extension_name = nil

```

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

```

9295     if options.debugExtensions then
9296         local sorted_lhs = {}
9297         for lhs, _ in pairs(walkable_syntax) do
9298             table.insert(sorted_lhs, lhs)
9299         end
9300         table.sort(sorted_lhs)
9301
9302         local output_lines = {"{"}
9303         for lhs_index, lhs in ipairs(sorted_lhs) do
9304             local encoded_lhs = util.encode_json_string(lhs)
9305             table.insert(output_lines, [{" "] .. encoded_lhs .. [{" ": []}])
9306             local rule = walkable_syntax[lhs]
9307             for rhs_index, rhs in ipairs(rule) do
9308                 local human_readable_rhs
9309                 if type(rhs) == "string" then
9310                     human_readable_rhs = rhs
9311                 else
9312                     local pattern_name
9313                     if rhs[3] then
9314                         pattern_name = rhs[3]
9315                     else
9316                         pattern_name = "Anonymous Pattern"
9317                     end
9318                     local extension_name = rhs[2]
9319                     human_readable_rhs = pattern_name .. [{" ("] .. extension_name .. [{" )"}]
9320                 end
9321                 local encoded_rhs = util.encode_json_string(human_readable_rhs)
9322                 local output_line = [{" "] .. encoded_rhs
9323                 if rhs_index < #rule then
9324                     output_line = output_line .. ","
9325                 end

```

```

9326         table.insert(output_lines, output_line)
9327     end
9328     local output_line = "    ]"
9329     if lhs_index < #sorted_lhs then
9330         output_line = output_line .. ", "
9331     end
9332     table.insert(output_lines, output_line)
9333 end
9334 table.insert(output_lines, "}")
9335
9336 local output = table.concat(output_lines, "\n")
9337 local output_filename = options.debugExtensionsFileName
9338 local output_file = assert(io.open(output_filename, "w"),
9339     [[Could not open file ]] .. output_filename .. [{" for writing}])
9340 assert(output_file:write(output))
9341 assert(output_file:close())
9342 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`.

```

9343     for lhs, rule in pairs(walkable_syntax) do
9344         syntax[lhs] = parsers.fail
9345         for _, rhs in ipairs(rule) do
9346             local pattern

```

Although the interface of the `reader->insert_pattern` method does not document this (see Section 2.1.2), we allow the `reader->insert_pattern` and `reader->update_rule` methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```

9347         if type(rhs) == "string" then
9348             pattern = V(rhs)
9349         else
9350             pattern = rhs[1]
9351             if type(pattern) == "string" then
9352                 pattern = V(pattern)
9353             end
9354         end
9355         syntax[lhs] = syntax[lhs] + pattern
9356     end
9357 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.

```

9358     if options.underscores then
9359         self.add_special_character("_")

```

```

9360     end
9361
9362     if not options.codeSpans then
9363         syntax.Code = parsers.fail
9364     else
9365         self.add_special_character("`")
9366     end
9367
9368     if not options.html then
9369         syntax.DisplayHtml = parsers.fail
9370         syntax.InlineHtml = parsers.fail
9371         syntax.HtmlEntity = parsers.fail
9372     else
9373         self.add_special_character("&")
9374     end
9375
9376     if options.preserveTabs then
9377         options.stripIndent = false
9378     end
9379
9380     if not options.smartEllipses then
9381         syntax.Smart = parsers.fail
9382     else
9383         self.add_special_character(".")
9384     end
9385
9386     if not options.relativeReferences then
9387         syntax.AutoLinkRelativeReference = parsers.fail
9388     end
9389
9390     if options.contentLevel == "inline" then
9391         syntax[1] = "Inlines"
9392         syntax.Inlines = V("InitializeState")
9393             * parsers.Inline^0
9394             * ( parsers.spacing^0
9395             * parsers.eof / "" )
9396         syntax.Space = parsers.Space + parsers.blankline / writer.space
9397     end
9398
9399     local blocks_nested_t = util.table_copy(syntax)
9400     blocks_nested_t.ExpectedJekyllData = parsers.fail
9401     parsers.blocks_nested = Ct(blocks_nested_t)
9402
9403     parsers.blocks = Ct(syntax)
9404
9405     local inlines_t = util.table_copy(syntax)
9406     inlines_t[1] = "Inlines"

```

```

9407     inlines_t.Inlines = V("InitializeState")
9408         * parsers.Inline^0
9409         * ( parsers.spacing^0
9410           * parsers.eof / "" )
9411     parsers.inlines = Ct(inlines_t)
9412
9413     local inlines_no_inline_note_t = util.table_copy(inlines_t)
9414     inlines_no_inline_note_t.InlineNote = parsers.fail
9415     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
9416
9417     local inlines_no_html_t = util.table_copy(inlines_t)
9418     inlines_no_html_t.DisplayHtml = parsers.fail
9419     inlines_no_html_t.InlineHtml = parsers.fail
9420     inlines_no_html_t.HtmlEntity = parsers.fail
9421     parsers.inlines_no_html = Ct(inlines_no_html_t)
9422
9423     local inlines_nbsp_t = util.table_copy(inlines_t)
9424     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
9425     inlines_nbsp_t.Space = parsers.NonbreakingSpace
9426     parsers.inlines_nbsp = Ct(inlines_nbsp_t)
9427
9428     local inlines_no_link_or_emphasis_t = util.table_copy(inlines_t)
9429     inlines_no_link_or_emphasis_t.LinkAndEmph = parsers.fail
9430     inlines_no_link_or_emphasis_t.EndlineExceptions = parsers.EndlineExceptions - par
9431     parsers.inlines_no_link_or_emphasis = Ct(inlines_no_link_or_emphasis_t)

```

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

```

9432     return function(input)

```

Unicode-normalize the input.

```

9433         if options.unicodeNormalization then
9434             local form = options.unicodeNormalizationForm
9435             if form == "nfc" then
9436                 input = uni_algos.normalize.NFC(input)
9437             elseif form == "nfd" then
9438                 input = uni_algos.normalize.NFD(input)
9439             elseif form == "nfkc" then
9440                 input = uni_algos.normalize.NFKC(input)
9441             elseif form == "nfkd" then
9442                 input = uni_algos.normalize.NFKD(input)
9443             else
9444                 error(format("Unknown normalization form %s", form))
9445             end
9446         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.

```

9447     input = input:gsub("\r\n?", "\n")
9448     if input:sub(-1) ~= "\n" then
9449         input = input .. "\n"
9450     end

```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```

9451     references = {}
9452     local opt_string = {}
9453     for k, _ in pairs(defaultOptions) do
9454         local v = options[k]
9455         if type(v) == "table" then
9456             for _, i in ipairs(v) do
9457                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
9458             end
9459         elseif k ~= "cacheDir" then
9460             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
9461         end
9462     end
9463     table.sort(opt_string)
9464     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
9465     local output
9466     local function convert(input)
9467         local document = self.parser_functions.parse_blocks(input)
9468         local output = util.ropo_to_string(writer.document(document))

```

Remove block element / paragraph separators immediately followed by the output of `writer->undosep`, possibly interleaved by section ends. Then, remove any leftover output of `writer->undosep`.

```

9469     local undosep_start, undosep_end
9470     local potential_secend_start, secend_start
9471     local potential_sep_start, sep_start
9472     while true do
9473         -- find a `writer->undosep`
9474         undosep_start, undosep_end = output:find(writer.undosep_text, 1, true)
9475         if undosep_start == nil then break end
9476         -- skip any preceding section ends
9477         secend_start = undosep_start
9478         while true do
9479             potential_secend_start = secend_start - #writer.secend_text
9480             if potential_secend_start < 1
9481                 or output:sub(potential_secend_start, secend_start - 1) ~= writer.secend_text
9482             then break
9483             end
9484             secend_start = potential_secend_start
9485         end

```

```

9486     -- find an immediately preceding block element / paragraph separator
9487     sep_start = secend_start
9488     potential_sep_start = sep_start - #writer.interblocksep_text
9489     if potential_sep_start >= 1
9490         and output:sub(potential_sep_start, sep_start - 1) == writer.interblocks
9491         sep_start = potential_sep_start
9492     else
9493         potential_sep_start = sep_start - #writer.paragraphsep_text
9494         if potential_sep_start >= 1
9495             and output:sub(potential_sep_start, sep_start - 1) == writer.paragraph
9496             sep_start = potential_sep_start
9497         end
9498     end
9499     -- remove `writer->undosep` and immediately preceding block element / parag
9500     output = output:sub(1, sep_start - 1)
9501         .. output:sub(secend_start, undosep_start - 1)
9502         .. output:sub(undosep_end + 1)
9503     end
9504     return output
9505 end

```

If we cache markdown documents, produce the cache file and transform its filename to plain TeX output via the `writer->pack` method.

```

9506     if options.eagerCache or options.finalizeCache then
9507         local name = util.cache(options.cacheDir, input, salt, convert,
9508             ".md" .. writer.suffix)
9509         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

9510     else
9511         output = convert(input)
9512     end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```

9513     if options.finalizeCache then
9514         local file, mode
9515         if options.frozenCacheCounter > 0 then
9516             mode = "a"
9517         else
9518             mode = "w"
9519         end
9520         file = assert(io.open(options.frozenCacheFileName, mode),
9521             [[Could not open file ]] .. options.frozenCacheFileName
9522             .. [[ for writing]])
9523         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
9524             .. [[markdownFrozenCache]] .. options.frozenCacheCounter

```

```

9525         .. [[\endcsname{}}] .. output .. [{}]] .. "\n"))
9526         assert(file:close())
9527     end
9528     return output
9529 end
9530 end
9531 return self
9532 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.

```

9533 M.extensions = {}

```

3.1.7.1 Bracketed Spans

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

```

9534 M.extensions.bracketed_spans = function()
9535     return {
9536         name = "built-in bracketed_spans syntax extension",
9537         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.

```

9538         function self.span(s, attr)
9539             if self.flatten_inlines then return s end
9540             return {"\markdownRendererBracketedSpanAttributeContextBegin",
9541                 self.attributes(attr),
9542                 s,
9543                 "\markdownRendererBracketedSpanAttributeContextEnd{}}"}
9544         end
9545     end, extend_reader = function(self)
9546         local parsers = self.parsers
9547         local writer = self.writer
9548
9549         local span_label = parsers.lbracket
9550             * (Cs((parsers.alphanumeric1
9551                 + parsers.inticks
9552                 + parsers.autolink
9553                 + V("InlineHtml")
9554                 + ( parsers.backslash * parsers.backslash)
9555                 + ( parsers.backslash * (parsers.lbracket + parsers.rbr

```

```

9556             + V("Space") + V("Endline")
9557             + (parsers.any
9558               - (parsers.newline + parsers.lbracket + parsers.rbracket
9559                 + parsers.blankline^2))))^1)
9560             / self.parser_functions.parse_inlines)
9561             * parsers.rbracket
9562
9563     local Span = span_label
9564               * Ct(parsers.attributes)
9565               / writer.span
9566
9567     self.insert_pattern("Inline before LinkAndEmph",
9568                       Span, "Span")
9569   end
9570 }
9571 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.

```

9572 M.extensions.citations = function(citation_nbsps)
9573   return {
9574     name = "built-in citations syntax extension",
9575     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.

```

9576     function self.citations(text_cites, cites)
9577       local buffer = {}
9578       if self.flatten_inlines then
9579         for _,cite in ipairs(cites) do

```



```

9580         if cite.prenote then
9581             table.insert(buffer, {cite.prenote, " "})
9582         end
9583         table.insert(buffer, cite.name)
9584         if cite.postnote then
9585             table.insert(buffer, {" ", cite.postnote})
9586         end
9587     end
9588 else
9589     table.insert(buffer, {"\\markdownRenderer", text_cites and "TextCite" or "C
9590     {"", #cites, "}"}))
9591     for _,cite in ipairs(cites) do
9592         table.insert(buffer, {cite.suppress_author and "-" or "+", "{",
9593             cite.prenote or "", "}{"", cite.postnote or "", "}{"", cite.name, "}"}))
9594     end
9595 end
9596 return buffer
9597 end
9598 end, extend_reader = function(self)
9599     local parsers = self.parsers
9600     local writer = self.writer
9601
9602     local citation_chars
9603         = parsers.alphanumeric
9604         + S("#$%&-+<>~/_")
9605
9606     local citation_name
9607         = Cs(parsers.dash^-1) * parsers.at
9608         * Cs(citation_chars
9609             * (((citation_chars + parsers.internal_punctuation
9610                 - parsers.comma - parsers.semicolon)
9611                 * -#((parsers.internal_punctuation - parsers.comma
9612                     - parsers.semicolon)^0
9613                     * -(citation_chars + parsers.internal_punctuation
9614                         - parsers.comma - parsers.semicolon)))^0
9615                 * citation_chars)^-1)
9616
9617     local citation_body_prenote
9618         = Cs((parsers.alphanumeric^1
9619             + parsers.bracketed
9620             + parsers.inticks
9621             + parsers.autolink
9622             + V("InlineHtml")
9623             + V("Space") + V("Endline")
9624             + (parsers.anyescaped
9625                 - (parsers.newline + parsers.rbracket + parsers.blankline^
9626                 - (parsers.spnl * parsers.dash^-1 * parsers.at))^1)

```

```

9627
9628     local citation_body_postnote
9629         = Cs((parsers.alphanumeric^1
9630             + parsers.bracketed
9631             + parsers.inticks
9632             + parsers.autolink
9633             + V("InlineHtml")
9634             + V("Space") + V("Endline")
9635             + (parsers.anyescaped
9636               - (parsers.newline + parsers.rbracket + parsers.semicolon
9637                 + parsers.blankline^2))
9638             - (parsers.spnl * parsers.rbracket))^1)
9639
9640     local citation_body_chunk
9641         = ( citation_body_prenote
9642           * parsers.spnlc_sep
9643           + Cc("")
9644           * parsers.spnlc
9645           )
9646         * citation_name
9647         * (parsers.internal_punctuation - parsers.semicolon)^-
1
9648         * ( parsers.spnlc / function(_) return end
9649           * citation_body_postnote
9650           + Cc("")
9651           * parsers.spnlc
9652           )
9653
9654     local citation_body
9655         = citation_body_chunk
9656         * ( parsers.semicolon
9657           * parsers.spnlc
9658           * citation_body_chunk
9659           )^0
9660
9661     local citation_headless_body_postnote
9662         = Cs((parsers.alphanumeric^1
9663             + parsers.bracketed
9664             + parsers.inticks
9665             + parsers.autolink
9666             + V("InlineHtml")
9667             + V("Space") + V("Endline")
9668             + (parsers.anyescaped
9669               - (parsers.newline + parsers.rbracket + parsers.at
9670                 + parsers.semicolon + parsers.blankline^2))
9671             - (parsers.spnl * parsers.rbracket))^0)
9672

```

```

9673     local citation_headless_body
9674         = citation_headless_body_postnote
9675         * ( parsers.semicolon
9676           * parsers.spnlc
9677           * citation_body_chunk
9678         )^0
9679
9680     local citations
9681         = function(text_cites, raw_cites)
9682         local function normalize(str)
9683             if str == "" then
9684                 str = nil
9685             else
9686                 str = (citation_nbsps and
9687                       self.parser_functions.parse_inlines_nbsp or
9688                       self.parser_functions.parse_inlines)(str)
9689             end
9690             return str
9691         end
9692
9693         local cites = {}
9694         for i = 1,#raw_cites,4 do
9695             cites[#cites+1] = {
9696                 prenote = normalize(raw_cites[i]),
9697                 suppress_author = raw_cites[i+1] == "-",
9698                 name = writer.identifier(raw_cites[i+2]),
9699                 postnote = normalize(raw_cites[i+3]),
9700             }
9701         end
9702         return writer.citations(text_cites, cites)
9703     end
9704
9705     local TextCitations
9706         = Ct((parsers.spnlc
9707           * Cc("")
9708           * citation_name
9709           * ((parsers.spnlc
9710             * parsers.lbracket
9711             * citation_headless_body
9712             * parsers.rbracket) + Cc("")))^1)
9713     / function(raw_cites)
9714         return citations(true, raw_cites)
9715     end
9716
9717     local ParenthesizedCitations
9718         = Ct((parsers.spnlc
9719           * parsers.lbracket

```

```

9720         * citation_body
9721         * parsers.rbracket)^1)
9722         / function(raw_cites)
9723             return citations(false, raw_cites)
9724         end
9725
9726     local Citations = TextCitations + ParenthesizedCitations
9727
9728     self.insert_pattern("Inline before LinkAndEmph",
9729                       Citations, "Citations")
9730
9731     self.add_special_character("@")
9732     self.add_special_character("-")
9733 end
9734 }
9735 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.

```

9736 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.

```

9737 local languages_json = (function()
9738     local base, prev, curr
9739     for _, pathname in ipairs{kpse.lookup(language_map, { all=true })} do
9740         local file = io.open(pathname, "r")
9741         if not file then goto continue end
9742         local input = assert(file:read("*a"))
9743         assert(file:close())
9744         local json = input:gsub('[^\n]-:', '[%1]=')
9745         curr = load("_ENV = {}; return "..json")()
9746         if type(curr) == "table" then
9747             if base == nil then
9748                 base = curr
9749             else
9750                 setmetatable(prev, { __index = curr })
9751             end
9752             prev = curr
9753         end
9754         ::continue::
9755     end

```

```

9756     return base or {}
9757 end()
9758
9759 return {
9760     name = "built-in content_blocks syntax extension",
9761     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.

```

9762     function self.contentblock(src,suf,type,tit)
9763         if not self.is_writing then return "" end
9764         src = src..".."..suf
9765         suf = suf:lower()
9766         if type == "onlineimage" then
9767             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
9768                 "{" ,self.string(src),"} ",
9769                 "{" ,self.uri(src),"} ",
9770                 "{" ,self.string(tit or ""),"} "}
9771         elseif languages_json[suf] then
9772             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
9773                 "{" ,self.string(languages_json[suf]) ,"} ",
9774                 "{" ,self.string(src),"} ",
9775                 "{" ,self.uri(src),"} ",
9776                 "{" ,self.string(tit or ""),"} "}
9777         else
9778             return {"\\markdownRendererContentBlock{" ,suf,"} ",
9779                 "{" ,self.string(src),"} ",
9780                 "{" ,self.uri(src),"} ",
9781                 "{" ,self.string(tit or ""),"} "}
9782         end
9783     end
9784 end, extend_reader = function(self)
9785     local parsers = self.parsers
9786     local writer = self.writer
9787
9788     local contentblock_tail
9789         = parsers.optionaltitle
9790         * (parsers.newline + parsers.eof)
9791
9792     -- case insensitive online image suffix:
9793     local onlineimagesuffix
9794         = (function(...)
9795             local parser = nil
9796             for _, suffix in ipairs({...}) do
9797                 local pattern=nil

```

```

9798         for i=1,#suffix do
9799             local char=suffix:sub(i,i)
9800             char = S(char:lower()..char:upper())
9801             if pattern == nil then
9802                 pattern = char
9803             else
9804                 pattern = pattern * char
9805             end
9806         end
9807         if parser == nil then
9808             parser = pattern
9809         else
9810             parser = parser + pattern
9811         end
9812     end
9813     return parser
9814 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
9815
9816 -- online image url for iA Writer content blocks with mandatory suffix,
9817 -- allowing nested brackets:
9818 local onlineimageurl
9819     = (parsers.less
9820         * Cs((parsers.anyescaped
9821             - parsers.more
9822             - parsers.spacing
9823             - #(parsers.period
9824                 * onlineimagesuffix
9825                 * parsers.more
9826                 * contentblock_tail))^0)
9827         * parsers.period
9828         * Cs(onlineimagesuffix)
9829         * parsers.more
9830         + (Cs((parsers.inparens
9831             + (parsers.anyescaped
9832                 - parsers.spacing
9833                 - parsers.rparent
9834                 - #(parsers.period
9835                     * onlineimagesuffix
9836                     * contentblock_tail))))^0)
9837         * parsers.period
9838         * Cs(onlineimagesuffix))
9839     ) * Cc("onlineimage")
9840
9841 -- filename for iA Writer content blocks with mandatory suffix:
9842 local localfilepath
9843     = parsers.slash
9844     * Cs((parsers.anyescaped

```

```

9845         - parsers.tab
9846         - parsers.newline
9847         - #(parsers.period
9848           * parsers.alphanumeric^1
9849           * contentblock_tail))^1)
9850         * parsers.period
9851         * Cs(parsers.alphanumeric^1)
9852         * Cc("localfile")
9853
9854     local ContentBlock
9855         = parsers.check_trail_no_rem
9856         * (localfilepath + onlineimageurl)
9857         * contentblock_tail
9858         / writer.contentblock
9859
9860     self.insert_pattern("Block before Blockquote",
9861                       ContentBlock, "ContentBlock")
9862 end
9863 }
9864 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.

```

9865 M.extensions.definition_lists = function(tight_lists)
9866   return {
9867     name = "built-in definition_lists syntax extension",
9868     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.

```

9869     local function dliitem(term, defs)
9870       local retVal = {"\\markdownRendererdliitem{",term,""}
9871       for _, def in ipairs(defs) do
9872         retVal[#retVal+1] = {"\\markdownRendererdlidefinitionBegin ",def,
9873                             "\\markdownRendererdlidefinitionEnd "}
9874       end
9875       retVal[#retVal+1] = "\\markdownRendererdliitemEnd "
9876       return retVal
9877     end
9878
9879     function self.definitionlist(items,tight)
9880       if not self.is_writing then return "" end

```

```

9881     local buffer = {}
9882     for _,item in ipairs(items) do
9883         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
9884     end
9885     if tight and tight_lists then
9886         return {"\\markdownRendererDlBeginTight\n", buffer,
9887             "\\markdownRendererDlEndTight"}
9888     else
9889         return {"\\markdownRendererDlBegin\n", buffer,
9890             "\\markdownRendererDlEnd"}
9891     end
9892 end
9893 end, extend_reader = function(self)
9894     local parsers = self.parsers
9895     local writer = self.writer
9896
9897     local defstartchar = S("~:")
9898
9899     local defstart = parsers.check_trail_length(0) * defstartchar * #parsers.spaci
9900         * (parsers.tab + parsers.space^-
3)
9901         + parsers.check_trail_length(1) * defstartchar * #parsers.spaci
9902         * (parsers.tab + parsers.space^-
2)
9903         + parsers.check_trail_length(2) * defstartchar * #parsers.spaci
9904         * (parsers.tab + parsers.space^-
1)
9905         + parsers.check_trail_length(3) * defstartchar * #parsers.spaci
9906
9907     local indented_line = (parsers.check_minimal_indent / "") * parsers.check_code_
9908
9909     local blank = parsers.check_minimal_blank_indent_and_any_trail * parsers.option
9910
9911     local dlchunk = Cs(parsers.line * (indented_line - blank)^0)
9912
9913     local indented_blocks = function(bl)
9914         return Cs( bl
9915             * (blank^1 * (parsers.check_minimal_indent / ""))
9916             * parsers.check_code_trail * -parsers.blankline * bl)^0
9917             * (blank^1 + parsers.eof))
9918     end
9919
9920     local function definition_list_item(term, defs, _)
9921         return { term = self.parser_functions.parse_inlines(term),
9922             definitions = defs }
9923     end
9924 end

```



```

9925     local DefinitionListItemLoose
9926         = C(parsers.line) * blank^0
9927         * Ct((parsers.check_minimal_indent * (defstart
9928             * indented_blocks(dlchunk)
9929             / self.parser_functions.parse_blocks_nested))^1)
9930         * Cc(false) / definition_list_item
9931
9932     local DefinitionListItemTight
9933         = C(parsers.line)
9934         * Ct((parsers.check_minimal_indent * (defstart * dlchunk
9935             / self.parser_functions.parse_blocks_nested))^1)
9936         * Cc(true) / definition_list_item
9937
9938     local DefinitionList
9939         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
9940           + Ct(DefinitionListItemTight^1)
9941           * (blank^0
9942             * -DefinitionListItemLoose * Cc(true))
9943           ) / writer.definitionlist
9944
9945     self.insert_pattern("Block after Heading",
9946                       DefinitionList, "DefinitionList")
9947 end
9948 }
9949 end

```

3.1.7.5 Fancy Lists

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

```

9950 M.extensions.fancy_lists = function()
9951     return {
9952         name = "built-in fancy_lists syntax extension",
9953         extend_writer = function(self)
9954             local options = self.options
9955

```

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.

```

9956     function self.fancylist(items,tight,startnum,numstyle,numdelim)
9957         if not self.is_writing then return "" end
9958         local buffer = {}
9959         local num = startnum
9960         for _,item in ipairs(items) do
9961             if item ~= "" then
9962                 buffer[#buffer + 1] = self.fancyitem(item,num)
9963             end
9964             if num ~= nil and item ~= "" then
9965                 num = num + 1
9966             end
9967         end
9968         local contents = util.intersperse(buffer,"\n")
9969         if tight and options.tightLists then
9970             return {"\\markdownRendererFancyOlBeginTight{",
9971                 numstyle,"}{",numdelim,"}",contents,
9972                 "\\n\\markdownRendererFancyOlEndTight "}
9973         else
9974             return {"\\markdownRendererFancyOlBegin{",
9975                 numstyle,"}{",numdelim,"}",contents,
9976                 "\\n\\markdownRendererFancyOlEnd "}
9977         end
9978     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.

```

9979     function self.fancyitem(s,num)
9980         if num ~= nil then
9981             return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
9982                 "\\n\\markdownRendererFancyOlItemEnd "}
9983         else
9984             return {"\\markdownRendererFancyOlItem ",s,"\\n\\markdownRendererFancyOlItemEnd "}
9985         end
9986     end

```

```

9987     end, extend_reader = function(self)
9988         local parsers = self.parsers
9989         local options = self.options
9990         local writer = self.writer
9991
9992         local function combine_markers_and_delims(markers, delims)
9993             local markers_table = {}
9994             for _,marker in ipairs(markers) do
9995                 local start_marker
9996                 local continuation_marker
9997                 if type(marker) == "table" then
9998                     start_marker = marker[1]
9999                     continuation_marker = marker[2]
10000                 else
10001                     start_marker = marker
10002                     continuation_marker = marker
10003                 end
10004                 for _,delim in ipairs(delims) do
10005                     table.insert(markers_table, {start_marker, continuation_marker, delim})
10006                 end
10007             end
10008             return markers_table
10009         end
10010
10011         local function join_table_with_func(func, markers_table)
10012             local pattern = func(table.unpack(markers_table[1]))
10013             for i = 2, #markers_table do
10014                 pattern = pattern + func(table.unpack(markers_table[i]))
10015             end
10016             return pattern
10017         end
10018
10019         local lowercase_letter_marker = R("az")
10020         local uppercase_letter_marker = R("AZ")
10021
10022         local roman_marker = function(chars)
10023             local m, d, c = P(chars[1]), P(chars[2]), P(chars[3])
10024             local l, x, v, i = P(chars[4]), P(chars[5]), P(chars[6]), P(chars[7])
10025             return m^-3
10026                 * (c*m + c*d + d^-1 * c^-3)
10027                 * (x*c + x*l + l^-1 * x^-3)
10028                 * (i*x + i*v + v^-1 * i^-3)
10029         end
10030
10031         local lowercase_roman_marker = roman_marker({"m", "d", "c", "l", "x", "v", "i"})
10032         local uppercase_roman_marker = roman_marker({"M", "D", "C", "L", "X", "V", "I"})
10033

```

```

10034     local lowercase_opening_roman_marker = P("i")
10035     local uppercase_opening_roman_marker = P("I")
10036
10037     local digit_marker = parsers.dig * parsers.dig^-8
10038
10039     local markers = {
10040         {lowercase_opening_roman_marker, lowercase_roman_marker},
10041         {uppercase_opening_roman_marker, uppercase_roman_marker},
10042         lowercase_letter_marker,
10043         uppercase_letter_marker,
10044         lowercase_roman_marker,
10045         uppercase_roman_marker,
10046         digit_marker
10047     }
10048
10049     local delims = {
10050         parsers.period,
10051         parsers.rparent
10052     }
10053
10054     local markers_table = combine_markers_and_delims(markers, delims)
10055
10056     local function enumerator(start_marker, _, delimiter_type, interrupting)
10057         local delimiter_range
10058         local allowed_end
10059         if interrupting then
10060             delimiter_range = P("1")
10061             allowed_end = C(parsers.spacechar^1) * #parsers.linechar
10062         else
10063             delimiter_range = start_marker
10064             allowed_end = C(parsers.spacechar^1) + #(parsers.newline + parsers.eof)
10065         end
10066
10067         return parsers.check_trail
10068             * Ct(C(delimiter_range) * C(delimiter_type))
10069             * allowed_end
10070     end
10071
10072     local starter = join_table_with_func(enumerator, markers_table)
10073
10074     local TightListItem = function(starter)
10075         return parsers.add_indent(starter, "li")
10076             * parsers.indented_content_tight
10077     end
10078
10079     local LooseListItem = function(starter)
10080         return parsers.add_indent(starter, "li")

```

```

10081         * parsers.indented_content_loose
10082         * remove_indent("li")
10083     end
10084
10085     local function roman2number(roman)
10086         local romans = { ["M"] = 1000, ["D"] = 500, ["C"] = 100, ["L"] = 50, ["X"] =
10087         local numeral = 0
10088
10089         local i = 1
10090         local len = string.len(roman)
10091         while i < len do
10092             local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
10093             if z1 < z2 then
10094                 numeral = numeral + (z2 - z1)
10095                 i = i + 2
10096             else
10097                 numeral = numeral + z1
10098                 i = i + 1
10099             end
10100         end
10101         if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
10102         return numeral
10103     end
10104
10105     local function sniffstyle(numstr, delimend)
10106         local numdelim
10107         if delimend == ")" then
10108             numdelim = "OneParen"
10109         elseif delimend == "." then
10110             numdelim = "Period"
10111         else
10112             numdelim = "Default"
10113         end
10114
10115         local num
10116         num = numstr:match("^([I])$")
10117         if num then
10118             return roman2number(num), "UpperRoman", numdelim
10119         end
10120         num = numstr:match("^([i])$")
10121         if num then
10122             return roman2number(string.upper(num)), "LowerRoman", numdelim
10123         end
10124         num = numstr:match("^([A-Z])$")
10125         if num then
10126             return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
10127         end

```

```

10128     num = numstr:match("^([a-z])$")
10129     if num then
10130         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
10131     end
10132     num = numstr:match("^([IVXLCDM]+)")
10133     if num then
10134         return roman2number(num), "UpperRoman", numdelim
10135     end
10136     num = numstr:match("^([ivxlc dm]+)")
10137     if num then
10138         return roman2number(string.upper(num)), "LowerRoman", numdelim
10139     end
10140     return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
10141 end
10142
10143 local function fancylist(items,tight,start)
10144     local startnum, numstyle, numdelim = sniffstyle(start[2][1], start[2][2])
10145     return writer.fancylist(items,tight,
10146                             options.startNumber and startnum or 1,
10147                             numstyle or "Decimal",
10148                             numdelim or "Default")
10149 end
10150
10151 local FancyListOfType = function(start_marker, continuation_marker, delimiter_t
10152     local enumerator_start = enumerator(start_marker, continuation_marker, delimi
10153     local enumerator_cont = enumerator(continuation_marker, continuation_marker,
10154     return Cg(enumerator_start, "listtype")
10155         * (Ct( TightListItem(Cb("listtype"))
10156             * ((parsers.check_minimal_indent / "") * TightListItem(enumerator_co
10157         * Cc(true)
10158         * -#((parsers.conditionally_indented_blankline^0 / ""))
10159             * parsers.check_minimal_indent * enumerator_cont)
10160     + Ct( LooseListItem(Cb("listtype"))
10161         * ((parsers.conditionally_indented_blankline^0 / ""))
10162         * (parsers.check_minimal_indent / "") * LooseListItem(enumerator_c
10163     * Cc(false)
10164     ) * Ct(Cb("listtype")) / fancylist
10165 end
10166
10167 local FancyList = join_table_with_func(FancyListOfType, markers_table)
10168
10169 local Endline = parsers.newline
10170     * (parsers.check_minimal_indent
10171     * -parsers.EndlineExceptions
10172     + parsers.check_optional_indent
10173     * -parsers.EndlineExceptions
10174     * -starter)

```

```

10175             * parsers.spacechar^0
10176             / writer.soft_line_break
10177
10178         self.update_rule("OrderedList", FancyList)
10179         self.update_rule("Endline", Endline)
10180     end
10181 }
10182 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.

```

10183 M.extensions.fenced_code = function(blank_before_code_fence,
10184                                     allow_attributes,
10185                                     allow_raw_blocks)
10186     return {
10187         name = "built-in fenced_code syntax extension",
10188         extend_writer = function(self)
10189             local options = self.options
10190

```

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.

```

10191     function self.fencedCode(s, i, attr)
10192         if not self.is_writing then return "" end
10193         s = s:gsub("\n$", "")
10194         local buf = {}
10195         if attr ~= nil then
10196             table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
10197                               self.attributes(attr)})
10198         end
10199         local name = util.cache_verbatim(options.cacheDir, s)
10200         table.insert(buf, {"\\markdownRendererInputFencedCode{" ,
10201                           name,"}{" ,self.string(i),"}{" ,self.infostring(i),"}"}))
10202         if attr ~= nil then
10203             table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd{")
10204         end
10205         return buf
10206     end
10207

```

Define `writer->rawBlock` as a function that will transform an input raw block `s` with the raw attribute `attr` to the output format.

```

10208     if allow_raw_blocks then
10209         function self.rawBlock(s, attr)
10210             if not self.is_writing then return "" end
10211             s = s:gsub("\n$", "")
10212             local name = util.cache_verbatim(options.cacheDir, s)
10213             return {"\\markdownRendererInputRawBlock{",
10214                 name,"}{" , self.string(attr),"}" }
10215         end
10216     end
10217 end, extend_reader = function(self)
10218     local parsers = self.parsers
10219     local writer = self.writer
10220
10221     local function captures_geq_length(_,i,a,b)
10222         return #a >= #b and i
10223     end
10224
10225     local function strip_enclosing_whitespaces(str)
10226         return str:gsub("^%s*(.)%s*$", "%1")
10227     end
10228
10229     local tilde_infostring = Cs(Cs((V("HtmlEntity")
10230         + parsers.anyescaped
10231         - parsers.newline)^0)
10232         / strip_enclosing_whitespaces)
10233
10234     local backtick_infostring = Cs(Cs((V("HtmlEntity")
10235         + (-#(parsers.backslash * parsers.backtick) *
10236         - parsers.newline
10237         - parsers.backtick)^0)
10238         / strip_enclosing_whitespaces)
10239
10240     local fenceindent
10241
10242     local function has_trail(indent_table)
10243         return indent_table ~= nil and
10244             indent_table.trail ~= nil and
10245             next(indent_table.trail) ~= nil
10246     end
10247
10248     local function has_indents(indent_table)
10249         return indent_table ~= nil and
10250             indent_table.indents ~= nil and
10251             next(indent_table.indents) ~= nil
10252     end

```



```

10253
10254     local function get_last_indent_name(indent_table)
10255         if has_indents(indent_table) then
10256             return indent_table.indents[#indent_table.indents].name
10257         end
10258     end
10259
10260     local function count_fenced_start_indent(_, _, indent_table, trail)
10261         local last_indent_name = get_last_indent_name(indent_table)
10262         fenceindent = 0
10263         if last_indent_name ~= "li" then
10264             fenceindent = #trail
10265         end
10266         return true
10267     end
10268
10269     local fencehead      = function(char, infostring)
10270         return          Cmt(Cb("indent_info") * parsers.check_trail, count_fence
10271             * Cg(char^3, "fencelength")
10272             * parsers.optionalspace
10273             * infostring
10274             * (parsers.newline + parsers.eof)
10275     end
10276
10277     local fencetail      = function(char)
10278         return          parsers.check_trail_no_rem
10279             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
10280             * parsers.optionalspace * (parsers.newline + parsers.eof)
10281             + parsers.eof
10282     end
10283
10284     local function process_fenced_line(s, i, indent_table, line_content, is_blank)
10285         local remainder = ""
10286         if has_trail(indent_table) then
10287             remainder = indent_table.trail.internal_remainder
10288         end
10289
10290         if is_blank and get_last_indent_name(indent_table) == "li" then
10291             remainder = ""
10292         end
10293
10294         local str = remainder .. line_content
10295         local index = 1
10296         local remaining = fenceindent
10297
10298         while true do
10299             local c = str:sub(index, index)

```

```

10300         if c == " " and remaining > 0 then
10301             remaining = remaining - 1
10302             index = index + 1
10303         elseif c == "\t" and remaining > 3 then
10304             remaining = remaining - 4
10305             index = index + 1
10306         else
10307             break
10308         end
10309     end
10310
10311     return true, str:sub(index)
10312 end
10313
10314 local fencedline = function(char)
10315     return Cmt(Cb("indent_info") * C(parsers.line - fencetail(char)) * Cc(false),
10316 end
10317
10318 local blankfencedline = Cmt(Cb("indent_info") * C(parsers.blankline) * Cc(true)
10319
10320 local TildeFencedCode
10321     = fencehead(parsers.tilde, tilde_infostring)
10322     * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10323         + (parsers.check_minimal_indent / "") * fencedline(parsers.tilde))
10324     * ((parsers.check_minimal_indent / "") * fencetail(parsers.tilde) + pars
10325
10326 local BacktickFencedCode
10327     = fencehead(parsers.backtick, backtick_infostring)
10328     * Cs(((parsers.check_minimal_blank_indent / "") * blankfencedline
10329         + (parsers.check_minimal_indent / "") * fencedline(parsers.backtick)
10330     * ((parsers.check_minimal_indent / "") * fencetail(parsers.backtick) + p
10331
10332 local infostring_with_attributes
10333     = Ct(C((parsers.linechar
10334         - ( parsers.optionalspace
10335           * parsers.attributes))^0)
10336     * parsers.optionalspace
10337     * Ct(parsers.attributes))
10338
10339 local FencedCode
10340     = ((TildeFencedCode + BacktickFencedCode)
10341     / function(infostring, code)
10342         local expanded_code = self.expandtabs(code)
10343
10344         if allow_raw_blocks then
10345             local raw_attr = lpeg.match(parsers.raw_attribute,
10346                                     infostring)

```

```

10347         if raw_attr then
10348             return writer.rawBlock(expanded_code, raw_attr)
10349         end
10350     end
10351
10352     local attr = nil
10353     if allow_attributes then
10354         local match = lpeg.match(infostring_with_attributes,
10355                                 infostring)
10356         if match then
10357             infostring, attr = table.unpack(match)
10358         end
10359     end
10360     return writer.fencedCode(expanded_code, infostring, attr)
10361 end)
10362
10363 self.insert_pattern("Block after Verbatim",
10364                    FencedCode, "FencedCode")
10365
10366 local fencestart
10367 if blank_before_code_fence then
10368     fencestart = parsers.fail
10369 else
10370     fencestart = fencehead(parsers.backtick, backtick_infostring)
10371                 + fencehead(parsers.tilde, tilde_infostring)
10372 end
10373
10374 self.update_rule("EndlineExceptions", function(previous_pattern)
10375     if previous_pattern == nil then
10376         previous_pattern = parsers.EndlineExceptions
10377     end
10378     return previous_pattern + fencestart
10379 end)
10380
10381 self.add_special_character("`")
10382 self.add_special_character("~")
10383 end
10384 }
10385 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.

```

10386 M.extensions.fenced_divs = function(blank_before_div_fence)

```

```

10387 return {
10388     name = "built-in fenced_divs syntax extension",
10389     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.

```

10390     function self.div_begin(attributes)
10391         local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",
10392                               self.attributes(attributes)}
10393         local end_output = {"\\markdownRendererFencedDivAttributeContextEnd{}}"
10394         return self.push_attributes("div", attributes, start_output, end_output)
10395     end

```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```

10396     function self.div_end()
10397         return self.pop_attributes("div")
10398     end
10399 end, extend_reader = function(self)
10400     local parsers = self.parsers
10401     local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

10402     local fenced_div_infostring
10403         = C((parsers.linechar
10404             - ( parsers.spacechar^1
10405               * parsers.colon^1))^1)
10406
10407     local fenced_div_begin = parsers.nonindentSPACE
10408         * parsers.colon^3
10409         * parsers.optionalspace
10410         * fenced_div_infostring
10411         * ( parsers.spacechar^1
10412           * parsers.colon^1)^0
10413         * parsers.optionalspace
10414         * (parsers.newline + parsers.eof)
10415
10416     local fenced_div_end = parsers.nonindentSPACE
10417         * parsers.colon^3
10418         * parsers.optionalspace
10419         * (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.

```

10420 self.initialize_named_group("fenced_div_level", "0")
10421 self.initialize_named_group("fenced_div_num_opening_indents")
10422
10423 local function increment_div_level()
10424     local function push_indent_table(s, i, indent_table, -- luacheck: ignore s i
10425                                     fenced_div_num_opening_indents, fenced_div_l
10426         fenced_div_level = tonumber(fenced_div_level) + 1
10427         local num_opening_indents = 0
10428         if indent_table.indents ~= nil then
10429             num_opening_indents = #indent_table.indents
10430         end
10431         fenced_div_num_opening_indents[fenced_div_level] = num_opening_indents
10432         return true, fenced_div_num_opening_indents
10433     end
10434
10435     local function increment_level(s, i, fenced_div_level) -- luacheck: ignore s i
10436         fenced_div_level = tonumber(fenced_div_level) + 1
10437         return true, tostring(fenced_div_level)
10438     end
10439
10440     return Cg( Cmt( Cb("indent_info")
10441                   * Cb("fenced_div_num_opening_indents")
10442                   * Cb("fenced_div_level"), push_indent_table)
10443               , "fenced_div_num_opening_indents")
10444         * Cg( Cmt( Cb("fenced_div_level"), increment_level)
10445             , "fenced_div_level")
10446     end
10447
10448 local function decrement_div_level()
10449     local function pop_indent_table(s, i, fenced_div_indent_table, fenced_div_lev
10450         fenced_div_level = tonumber(fenced_div_level)
10451         fenced_div_indent_table[fenced_div_level] = nil
10452         return true, tostring(fenced_div_level - 1)
10453     end
10454
10455     return Cg( Cmt( Cb("fenced_div_num_opening_indents")
10456                   * Cb("fenced_div_level"), pop_indent_table)
10457               , "fenced_div_level")
10458 end
10459
10460 local non_fenced_div_block = parsers.check_minimal_indent * V("Block")
10461                             - parsers.check_minimal_indent_and_trail * fenced_d
10462
10463 local non_fenced_div_paragraph = parsers.check_minimal_indent * V("Paragraph")
10464                                 - parsers.check_minimal_indent_and_trail * fenc
10465
10466

```

```

10467     local blank = parsers.minimally_indented_blank
10468
10469     local block_separated = parsers.block_sep_group(blank)
10470                             * non_fenced_div_block
10471
10472     local loop_body_pair = parsers.create_loop_body_pair(block_separated,
10473                                                         non_fenced_div_paragraph,
10474                                                         parsers.block_sep_group(t
10475                                                         parsers.par_sep_group(bla
10476
10477     local content_loop = ( non_fenced_div_block
10478                             * loop_body_pair.block^0
10479                             + non_fenced_div_paragraph
10480                             * block_separated
10481                             * loop_body_pair.block^0
10482                             + non_fenced_div_paragraph
10483                             * loop_body_pair.par^0)
10484                             * blank^0
10485
10486     local FencedDiv = fenced_div_begin
10487                       / function (infostring)
10488                           local attr = lpeg.match(Ct(parsers.attributes), infostring)
10489                           if attr == nil then
10490                               attr = {"." .. infostring}
10491                           end
10492                           return attr
10493                       end
10494                       / writer.div_begin
10495                       * increment_div_level()
10496                       * parsers.skipblanklines
10497                       * Ct(content_loop)
10498                       * parsers.minimally_indented_blank^0
10499                       * parsers.check_minimal_indent_and_trail * fenced_div_end
10500                       * decrement_div_level()
10501                       * (Cc("") / writer.div_end)
10502
10503     self.insert_pattern("Block after Verbatim",
10504                       FencedDiv, "FencedDiv")
10505
10506     self.add_special_character(":")
10507

```

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.

```

10508     local function is_inside_div()
10509         local function check_div_level(s, i, fenced_div_level) -- luacheck: ignore s

```

```

10510         fenced_div_level = tonumber(fenced_div_level)
10511         return fenced_div_level > 0
10512     end
10513
10514     return Cmt(Cb("fenced_div_level"), check_div_level)
10515 end
10516
10517 local function check_indent()
10518     local function compare_indent(s, i, indent_table, -- luacheck: ignore s i
10519                                     fenced_div_num_opening_indents, fenced_div_level)
10520         fenced_div_level = tonumber(fenced_div_level)
10521         local num_current_indents = (indent_table.current_line_indents ~= nil and
10522                                     #indent_table.current_line_indents) or 0
10523         local num_opening_indents = fenced_div_num_opening_indents[fenced_div_level]
10524         return num_current_indents == num_opening_indents
10525     end
10526
10527     return Cmt( Cb("indent_info")
10528                * Cb("fenced_div_num_opening_indents")
10529                * Cb("fenced_div_level"), compare_indent)
10530 end
10531
10532 local fencestart = is_inside_div()
10533                 * fenced_div_end
10534                 * check_indent()
10535
10536 if not blank_before_div_fence then
10537     self.update_rule("EndlineExceptions", function(previous_pattern)
10538         if previous_pattern == nil then
10539             previous_pattern = parsers.EndlineExceptions
10540         end
10541         return previous_pattern + fencestart
10542     end)
10543 end
10544 end
10545 }
10546 end

```

3.1.7.8 Header Attributes

The `extensions.header_attributes` function implements the Pandoc header attribute syntax extension.

```

10547 M.extensions.header_attributes = function()
10548     return {
10549         name = "built-in header_attributes syntax extension",
10550         extend_writer = function()
10551         end, extend_reader = function(self)

```

```

10552     local parsers = self.parsers
10553     local writer = self.writer
10554
10555     local function strip_atx_end(s)
10556         return s:gsub("%s+##%s*$", "")
10557     end
10558
10559     local AtxHeading = Cg(parsers.heading_start, "level")
10560         * parsers.optionalspace
10561         * (C(((parsers.linechar
10562             - (parsers.attributes
10563                 * parsers.optionalspace
10564                 * parsers.newline))
10565             * (parsers.linechar
10566                 - parsers.lbrace)^0)^1)
10567             / strip_atx_end
10568             / parsers.parse_heading_text)
10569         * Cg(Ct(parsers.newline
10570             + (parsers.attributes
10571                 * parsers.optionalspace
10572                 * parsers.newline)), "attributes")
10573         * Cb("level")
10574         * Cb("attributes")
10575         / writer.heading
10576
10577     local function strip_trailing_spaces(s)
10578         return s:gsub("%s*$", "")
10579     end
10580
10581     local heading_line = (parsers.linechar
10582         - (parsers.attributes
10583             * parsers.optionalspace
10584             * parsers.newline))^1
10585         - parsers.thematic_break_lines
10586
10587     local heading_text = heading_line
10588         * ((V("Endline") / "\n") * (heading_line - parsers.heading_
10589             * parsers.newline)^-1)
10590
10591     local SetextHeading = parsers.freeze_trail * parsers.check_trail_no_rem
10592         * #(heading_text
10593             * (parsers.attributes
10594                 * parsers.optionalspace
10595                 * parsers.newline)^-1
10596             * parsers.check_minimal_indent * parsers.check_trail *
10597             * Cs(heading_text) / strip_trailing_spaces
10598             / parsers.parse_heading_text)

```



```

10599         * Cg(Ct((parsers.attributes
10600             * parsers.optionalspace
10601             * parsers.newline)^-1), "attributes")
10602         * parsers.check_minimal_indent_and_trail * parsers.heading
10603         * Cb("attributes")
10604         * parsers.newline
10605         * parsers.unfreeze_trail
10606         / writer.heading
10607
10608     local Heading = AtxHeading + SetextHeading
10609     self.update_rule("Heading", Heading)
10610 end
10611 }
10612 end

```

3.1.7.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```

10613 M.extensions.inline_code_attributes = function()
10614   return {
10615     name = "built-in inline_code_attributes syntax extension",
10616     extend_writer = function()
10617     end, extend_reader = function(self)
10618       local writer = self.writer
10619
10620       local CodeWithAttributes = parsers.inticks
10621         * Ct(parsers.attributes)
10622         / writer.code
10623
10624       self.insert_pattern("Inline before Code",
10625         CodeWithAttributes,
10626         "CodeWithAttributes")
10627     end
10628   }
10629 end

```

3.1.7.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block syntax extension.

```

10630 M.extensions.line_blocks = function()
10631   return {
10632     name = "built-in line_blocks syntax extension",
10633     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.

```

10634     function self.lineblock(lines)
10635         if not self.is_writing then return "" end
10636         local buffer = {}
10637         for i = 1, #lines - 1 do
10638             buffer[#buffer + 1] = { lines[i], self.hard_line_break }
10639         end
10640         buffer[#buffer + 1] = lines[#lines]
10641
10642         return {"\\markdownRendererLineBlockBegin\n"
10643             ,buffer,
10644             "\n\\markdownRendererLineBlockEnd "}
10645     end
10646 end, extend_reader = function(self)
10647     local parsers = self.parsers
10648     local writer = self.writer
10649
10650     local LineBlock = Ct(
10651         (Cs(
10652             ( (parsers.pipe * parsers.space)/"
10653             * ((parsers.space)/entities.char_entity("nbsp"))^0
10654             * parsers.linechar^0 * (parsers.newline/"")
10655             * (-parsers.pipe
10656             * (parsers.space^1/" ")
10657             * parsers.linechar^1
10658             * (parsers.newline/"")
10659             )^0
10660             * (parsers.blankline/"")^0
10661             ) / self.parser_functions.parse_inlines)^1) / writer.lineblock
10662
10663     self.insert_pattern("Block after Blockquote",
10664         LineBlock, "LineBlock")
10665 end
10666 }
10667 end

```

3.1.7.11 Marked text

The `extensions.mark` function implements the Pandoc mark syntax extension.

```

10668 M.extensions.mark = function()
10669     return {
10670         name = "built-in mark syntax extension",
10671         extend_writer = function(self)

```

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

```

10672     function self.mark(s)
10673         if self.flatten_inlines then return s end
10674         return {"\\markdownRendererMark{" , s, "}" }

```

```

10675     end
10676 end, extend_reader = function(self)
10677     local parsers = self.parsers
10678     local writer = self.writer
10679
10680     local doubleequals = P("==")
10681
10682     local Mark = parsers.between(V("Inline"), doubleequals, doubleequals)
10683         / function (inlines) return writer.mark(inlines) end
10684
10685     self.add_special_character("=")
10686     self.insert_pattern("Inline before LinkAndEmph",
10687         Mark, "Mark")
10688 end
10689 }
10690 end

```

3.1.7.12 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```

10691 M.extensions.link_attributes = function()
10692     return {
10693         name = "built-in link_attributes syntax extension",
10694         extend_writer = function()
10695         end, extend_reader = function(self)
10696             local parsers = self.parsers
10697             local options = self.options
10698

```

The following patterns define link reference definitions with attributes.

```

10699     local define_reference_parser = (parsers.check_trail / "") * parsers.link_label
10700         * parsers.spnlc * parsers.url
10701         * ( parsers.spnlc_sep * parsers.title * (parsers.
10702             * parsers.only_blank
10703             + parsers.spnlc_sep * parsers.title * parsers.c
10704             + Cc("") * (parsers.spnlc * Ct(parsers.attribut
10705             + Cc("") * parsers.only_blank)
10706
10707     local ReferenceWithAttributes = define_reference_parser
10708         / self.register_link
10709
10710     self.update_rule("Reference", ReferenceWithAttributes)
10711

```

The following patterns define direct and indirect links with attributes.

```

10712
10713     local LinkWithAttributesAndEmph = Ct(parsers.link_and_emph_table * Cg(Cc(true),

```

```

10714                                     / self.defer_link_and_emphasis_processing
10715
10716     self.update_rule("LinkAndEmph", LinkWithAttributesAndEmph)
10717

```

The following patterns define autolinks with attributes.

```

10718     local AutoLinkUrlWithAttributes
10719         = parsers.auto_link_url
10720         * Ct(parsers.attributes)
10721         / self.auto_link_url
10722
10723     self.insert_pattern("Inline before AutoLinkUrl",
10724                       AutoLinkUrlWithAttributes,
10725                       "AutoLinkUrlWithAttributes")
10726
10727     local AutoLinkEmailWithAttributes
10728         = parsers.auto_link_email
10729         * Ct(parsers.attributes)
10730         / self.auto_link_email
10731
10732     self.insert_pattern("Inline before AutoLinkEmail",
10733                       AutoLinkEmailWithAttributes,
10734                       "AutoLinkEmailWithAttributes")
10735
10736     if options.relativeReferences then
10737
10738         local AutoLinkRelativeReferenceWithAttributes
10739             = parsers.auto_link_relative_reference
10740             * Ct(parsers.attributes)
10741             / self.auto_link_url
10742
10743         self.insert_pattern(
10744             "Inline before AutoLinkRelativeReference",
10745             AutoLinkRelativeReferenceWithAttributes,
10746             "AutoLinkRelativeReferenceWithAttributes")
10747
10748     end
10749
10750 end
10751 }
10752 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.

```
10753 M.extensions.notes = function(notes, inline_notes)
10754   assert(notes or inline_notes)
10755   return {
10756     name = "built-in notes syntax extension",
10757     extend_writer = function(self)
```

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

```
10758     function self.note(s)
10759       if self.flatten_inlines then return "" end
10760       return {"\\markdownRendererNote{" ,s,"}"}
10761     end
10762   end, extend_reader = function(self)
10763     local parsers = self.parsers
10764     local writer = self.writer
10765
10766     local rawnotes = parsers.rawnotes
10767
10768     if inline_notes then
10769       local InlineNote
10770         = parsers.circumflex
10771         * (parsers.link_label / self.parser_functions.parse_inlines_no_in
10772         / writer.note
10773
10774       self.insert_pattern("Inline after LinkAndEmph",
10775         InlineNote, "InlineNote")
10776     end
10777     if notes then
10778       local function strip_first_char(s)
10779         return s:sub(2)
10780       end
10781
10782       local RawNoteRef
10783         = #(parsers.lbracket * parsers.circumflex)
10784         * parsers.link_label / strip_first_char
10785
10786       -- like indirect_link
10787       local function lookup_note(ref)
10788         return writer.defer_call(function()
10789           local found = rawnotes[self.normalize_tag(ref)]
10790           if found then
10791             return writer.note(
10792               self.parser_functions.parse_blocks_nested(found))
10793           else
10794             return {"[" ,
```

```

10795         self.parser_functions.parse_inlines("^" .. ref), "]" }
10796     end
10797 end)
10798 end
10799
10800 local function register_note(ref, rawnote)
10801     local normalized_tag = self.normalize_tag(ref)
10802     if rawnotes[normalized_tag] == nil then
10803         rawnotes[normalized_tag] = rawnote
10804     end
10805     return ""
10806 end
10807
10808 local NoteRef = RawNoteRef / lookup_note
10809
10810 local optionally_indented_line = parsers.check_optional_indent_and_any_trail
10811
10812 local blank = parsers.check_optional_blank_indent_and_any_trail * parsers.opt
10813
10814 local chunk = Cs(parsers.line * (optionally_indented_line - blank)^0)
10815
10816 local indented_blocks = function(bl)
10817     return Cs( bl
10818         * (blank^1 * (parsers.check_optional_indent / ""))
10819         * parsers.check_code_trail * -parsers.blankline * bl)^0)
10820 end
10821
10822 local NoteBlock
10823     = parsers.check_trail_no_rem * RawNoteRef * parsers.colon
10824     * parsers.spnlc * indented_blocks(chunk)
10825     / register_note
10826
10827 local Reference = NoteBlock + parsers.Reference
10828
10829 self.update_rule("Reference", Reference)
10830 self.insert_pattern("Inline before LinkAndEmph",
10831     NoteRef, "NoteRef")
10832 end
10833
10834 self.add_special_character("^")
10835 end
10836 }
10837 end

```

3.1.7.14 Pipe Tables

The `extensions.pipe_table` function implements the PHP Markdown table syn-

tax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc table caption syntax extension for table captions. When the `table_attributes` parameter is also `true`, the function also allows attributes to be attached to the (possibly empty) table captions.

```

10838 M.extensions.pipe_tables = function(table_captions, table_attributes)
10839
10840   local function make_pipe_table_rectangular(rows)
10841     local num_columns = #rows[2]
10842     local rectangular_rows = {}
10843     for i = 1, #rows do
10844       local row = rows[i]
10845       local rectangular_row = {}
10846       for j = 1, num_columns do
10847         rectangular_row[j] = row[j] or ""
10848       end
10849       table.insert(rectangular_rows, rectangular_row)
10850     end
10851     return rectangular_rows
10852   end
10853
10854   local function pipe_table_row(allow_empty_first_column
10855                                 , nonempty_column
10856                                 , column_separator
10857                                 , column)
10858     local row_beginning
10859     if allow_empty_first_column then
10860       row_beginning = -- empty first column
10861                       #(parsers.spacechar^4
10862                         * column_separator)
10863                       * parsers.optionalspace
10864                       * column
10865                       * parsers.optionalspace
10866                       -- non-empty first column
10867                       + parsers.nonindentSPACE
10868                       * nonempty_column^-1
10869                       * parsers.optionalspace
10870     else
10871       row_beginning = parsers.nonindentSPACE
10872                       * nonempty_column^-1
10873                       * parsers.optionalspace
10874     end
10875
10876   return Ct(row_beginning
10877             * (-- single column with no leading pipes
10878               #(column_separator)

```

```

10879         * parsers.optionalspace
10880         * parsers.newline)
10881     * column_separator
10882     * parsers.optionalspace
10883     -- single column with leading pipes or
10884     -- more than a single column
10885     + (column_separator
10886         * parsers.optionalspace
10887         * column
10888         * parsers.optionalspace)^1
10889     * (column_separator
10890         * parsers.optionalspace)^-1))
10891 end
10892
10893 return {
10894     name = "built-in pipe_tables syntax extension",
10895     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.

```

10896     function self.table(rows, caption, attributes)
10897         if not self.is_writing then return "" end
10898         local buffer = {}
10899         if attributes ~= nil then
10900             table.insert(buffer,
10901                 "\\markdownRendererTableAttributeContextBegin\n")
10902             table.insert(buffer, self.attributes(attributes))
10903         end
10904         table.insert(buffer,
10905             {"\\markdownRendererTable{",
10906                 caption or "", "}{" , #rows - 1, "}{" ,
10907                 #rows[1], "}")
10908         local temp = rows[2] -- put alignments on the first row
10909         rows[2] = rows[1]
10910         rows[1] = temp
10911         for i, row in ipairs(rows) do
10912             table.insert(buffer, "{")
10913             for _, column in ipairs(row) do
10914                 if i > 1 then -- do not use braces for alignments
10915                     table.insert(buffer, "{")
10916                 end
10917                 table.insert(buffer, column)
10918                 if i > 1 then
10919                     table.insert(buffer, "}")
10920                 end
10921             end
10922         end

```



```

10922         table.insert(buffer, "}")
10923     end
10924     if attributes ~= nil then
10925         table.insert(buffer,
10926             "\\markdownRendererTableAttributeContextEnd{")
10927     end
10928     return buffer
10929 end
10930 end, extend_reader = function(self)
10931     local parsers = self.parsers
10932     local writer = self.writer
10933
10934     local table_hline_separator = parsers.pipe + parsers.plus
10935
10936     local table_hline_column = (parsers.dash
10937         - #(parsers.dash
10938             * (parsers.spacechar
10939                 + table_hline_separator
10940                 + parsers.newline)))^1
10941     * (parsers.colon * Cc("r")
10942         + parsers.dash * Cc("d"))
10943     + parsers.colon
10944     * (parsers.dash
10945         - #(parsers.dash
10946             * (parsers.spacechar
10947                 + table_hline_separator
10948                 + parsers.newline)))^1
10949     * (parsers.colon * Cc("c")
10950         + parsers.dash * Cc("l"))
10951
10952     local table_hline = pipe_table_row(false
10953         , table_hline_column
10954         , table_hline_separator
10955         , table_hline_column)
10956
10957     local table_caption_beginning = (parsers.check_minimal_blank_indent_and_any_tra
10958         * parsers.optionalspace * parsers.newline)^0
10959     * parsers.check_minimal_indent_and_trail
10960     * (P("Table")^-1 * parsers.colon)
10961     * parsers.optionalspace
10962
10963     local function strip_trailing_spaces(s)
10964         return s:gsub("%s*$", "")
10965     end
10966
10967     local table_row = pipe_table_row(true
10968         , (C((parsers.linechar - parsers.pipe)^1)

```

```

10969         / strip_trailing_spaces
10970         / self.parser_functions.parse_inlines)
10971     , parsers.pipe
10972     , (C((parsers.linechar - parsers.pipe)^0)
10973       / strip_trailing_spaces
10974       / self.parser_functions.parse_inlines))
10975
10976     local table_caption
10977     if table_captions then
10978         table_caption = #table_caption_beginning
10979         * table_caption_beginning
10980         if table_attributes then
10981             table_caption = table_caption
10982             * (C((( parsers.linechar
10983                 - (parsers.attributes
10984                   * parsers.optionalspace
10985                   * parsers.newline
10986                   * -( parsers.optionalspace
10987                     * parsers.linechar)))
10988               + ( parsers.newline
10989                 * #( parsers.optionalspace
10990                   * parsers.linechar)
10991                 * C(parsers.optionalspace) / writer.space))
10992             * (parsers.linechar
10993               - parsers.lbrace)^0)^1)
10994             / self.parser_functions.parse_inlines)
10995             * (parsers.newline
10996               + ( Ct(parsers.attributes)
10997                 * parsers.optionalspace
10998                 * parsers.newline))
10999         else
11000             table_caption = table_caption
11001             * C(( parsers.linechar
11002                 + ( parsers.newline
11003                   * #( parsers.optionalspace
11004                     * parsers.linechar)
11005                   * C(parsers.optionalspace) / writer.space))^1)
11006             / self.parser_functions.parse_inlines
11007             * parsers.newline
11008         end
11009     else
11010         table_caption = parsers.fail
11011     end
11012
11013     local PipeTable = Ct(table_row * parsers.newline * (parsers.check_minimal_indent
11014                       * table_hline * parsers.newline
11015                       * ((parsers.check_minimal_indent / {}) * table_row * parsers.

```

```

11016             / make_pipe_table_rectangular
11017             * table_caption^-1
11018             / writer.table
11019
11020     self.insert_pattern("Block after Blockquote",
11021                       PipeTable, "PipeTable")
11022   end
11023 }
11024 end

```

3.1.7.15 Raw Attributes

The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```

11025 M.extensions.raw_inline = function()
11026   return {
11027     name = "built-in raw_inline syntax extension",
11028     extend_writer = function(self)
11029       local options = self.options
11030

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

11031     function self.rawInline(s, attr)
11032       if not self.is_writing then return "" end
11033       if self.flatten_inlines then return s end
11034       local name = util.cache_verbatim(options.cacheDir, s)
11035       return {"\\markdownRendererInputRawInline{" ,
11036             name,"}{" , self.string(attr),"}"}
11037     end
11038   end, extend_reader = function(self)
11039     local writer = self.writer
11040
11041     local RawInline = parsers.inticks
11042                       * parsers.raw_attribute
11043                       / writer.rawInline
11044
11045     self.insert_pattern("Inline before Code",
11046                       RawInline, "RawInline")
11047   end
11048 }
11049 end

```

3.1.7.16 Strike-Through

The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

11050 M.extensions.strike_through = function()

```

```

11051 return {
11052   name = "built-in strike_through syntax extension",
11053   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.

```

11054     function self.strike_through(s)
11055       if self.flatten_inlines then return s end
11056       return {"\\markdownRendererStrikeThrough{" ,s,"}"}
11057     end
11058   end, extend_reader = function(self)
11059     local parsers = self.parsers
11060     local writer = self.writer
11061
11062     local StrikeThrough = (
11063       parsers.between(parsers.Inline, parsers.doubletildes,
11064         parsers.doubletildes)
11065     ) / writer.strike_through
11066
11067     self.insert_pattern("Inline after LinkAndEmph",
11068       StrikeThrough, "StrikeThrough")
11069
11070     self.add_special_character("~")
11071   end
11072 }
11073 end

```

3.1.7.17 Subscripts

The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

11074 M.extensions.subscripts = function()
11075   return {
11076     name = "built-in subscripts syntax extension",
11077     extend_writer = function(self)

```

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

```

11078     function self.subscript(s)
11079       if self.flatten_inlines then return s end
11080       return {"\\markdownRendererSubscript{" ,s,"}"}
11081     end
11082   end, extend_reader = function(self)
11083     local parsers = self.parsers
11084     local writer = self.writer
11085
11086     local Subscript = (
11087       parsers.between(parsers.Str, parsers.tilde, parsers.tilde)

```

```

11088     ) / writer.subscript
11089
11090     self.insert_pattern("Inline after LinkAndEmph",
11091                        Subscript, "Subscript")
11092
11093     self.add_special_character("~")
11094   end
11095 }
11096 end

```

3.1.7.18 Superscripts

The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

11097 M.extensions.superscripts = function()
11098   return {
11099     name = "built-in superscripts syntax extension",
11100     extend_writer = function(self)

```

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

```

11101     function self.superscript(s)
11102       if self.flatten_inlines then return s end
11103       return {"\\markdownRendererSuperscript{" ,s,"}"}
11104     end
11105   end, extend_reader = function(self)
11106     local parsers = self.parsers
11107     local writer = self.writer
11108
11109     local Superscript = (
11110       parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
11111     ) / writer.superscript
11112
11113     self.insert_pattern("Inline after LinkAndEmph",
11114                        Superscript, "Superscript")
11115
11116     self.add_special_character("^")
11117   end
11118 }
11119 end

```

3.1.7.19 T_EX Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```

11120 M.extensions.tex_math = function(tex_math_dollars,
11121                                  tex_math_single_backslash,
11122                                  tex_math_double_backslash)

```

```

11123 return {
11124     name = "built-in tex_math syntax extension",
11125     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.

```

11126     function self.display_math(s)
11127         if self.flatten_inlines then return s end
11128         return {"\\markdownRendererDisplayMath{" ,self.math(s),"}"}
11129     end

```

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

```

11130     function self.inline_math(s)
11131         if self.flatten_inlines then return s end
11132         return {"\\markdownRendererInlineMath{" ,self.math(s),"}"}
11133     end
11134 end, extend_reader = function(self)
11135     local parsers = self.parsers
11136     local writer = self.writer
11137
11138     local function between(p, starter, ender)
11139         return (starter * Cs(p * (p - ender)^0) * ender)
11140     end
11141
11142     local function strip_preceding_whitespaces(str)
11143         return str:gsub("^%s*(.)$", "%1")
11144     end
11145
11146     local allowed_before_closing = B( parsers.backslash * parsers.any
11147         + parsers.any * (parsers.any - parsers.backslash)
11148
11149     local allowed_before_closing_no_space = B( parsers.backslash * parsers.any
11150         + parsers.any * (parsers.nonspacechar
11151

```

The following patterns implement the Pandoc dollar math syntax extension.

```

11152     local dollar_math_content = (parsers.newline * (parsers.check_optional_indent /
11153         + parsers.backslash^-1
11154         * parsers.linechar)
11155         - parsers.blankline^2
11156         - parsers.dollar
11157
11158     local inline_math_opening_dollars = parsers.dollar
11159         * #(parsers.nonspacechar)
11160
11161     local inline_math_closing_dollars = allowed_before_closing_no_space
11162         * parsers.dollar

```

```

11163             * -(parsers.digit)
11164
11165     local inline_math_dollars = between(Cs( dollar_math_content),
11166                                         inline_math_opening_dollars,
11167                                         inline_math_closing_dollars)
11168
11169     local display_math_opening_dollars = parsers.dollar
11170                                         * parsers.dollar
11171
11172     local display_math_closing_dollars = parsers.dollar
11173                                         * parsers.dollar
11174
11175     local display_math_dollars = between(Cs( dollar_math_content),
11176                                         display_math_opening_dollars,
11177                                         display_math_closing_dollars)

```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```

11178     local backslash_math_content = (parsers.newline * (parsers.check_optional_inde
11179                                         + parsers.linechar)
11180                                     - parsers.blankline^2

```

The following patterns implement the Pandoc double backslash math syntax extension.

```

11181     local inline_math_opening_double = parsers.backslash
11182                                         * parsers.backslash
11183                                         * parsers.lparent
11184
11185     local inline_math_closing_double = allowed_before_closing
11186                                         * parsers.spacechar^0
11187                                         * parsers.backslash
11188                                         * parsers.backslash
11189                                         * parsers.rparent
11190
11191     local inline_math_double = between(Cs( backslash_math_content),
11192                                         inline_math_opening_double,
11193                                         inline_math_closing_double)
11194                                         / strip_preceding_whitespaces
11195
11196     local display_math_opening_double = parsers.backslash
11197                                         * parsers.backslash
11198                                         * parsers.lbracket
11199
11200     local display_math_closing_double = allowed_before_closing
11201                                         * parsers.spacechar^0
11202                                         * parsers.backslash
11203                                         * parsers.backslash
11204                                         * parsers.rbracket

```

```

11205
11206     local display_math_double = between(Cs( backslash_math_content),
11207                                         display_math_opening_double,
11208                                         display_math_closing_double)
11209                                         / strip_preceding_whitespace

```

The following patterns implement the Pandoc single backslash math syntax extension.

```

11210     local inline_math_opening_single = parsers.backslash
11211                                         * parsers.lparent
11212
11213     local inline_math_closing_single = allowed_before_closing
11214                                         * parsers.spacechar^0
11215                                         * parsers.backslash
11216                                         * parsers.rparent
11217
11218     local inline_math_single = between(Cs( backslash_math_content),
11219                                         inline_math_opening_single,
11220                                         inline_math_closing_single)
11221                                         / strip_preceding_whitespace
11222
11223     local display_math_opening_single = parsers.backslash
11224                                         * parsers.lbracket
11225
11226     local display_math_closing_single = allowed_before_closing
11227                                         * parsers.spacechar^0
11228                                         * parsers.backslash
11229                                         * parsers.rbracket
11230
11231     local display_math_single = between(Cs( backslash_math_content),
11232                                         display_math_opening_single,
11233                                         display_math_closing_single)
11234                                         / strip_preceding_whitespace
11235
11236     local display_math = parsers.fail
11237
11238     local inline_math = parsers.fail
11239
11240     if tex_math_dollars then
11241         display_math = display_math + display_math_dollars
11242         inline_math = inline_math + inline_math_dollars
11243     end
11244
11245     if tex_math_double_backslash then
11246         display_math = display_math + display_math_double
11247         inline_math = inline_math + inline_math_double
11248     end
11249
11250     if tex_math_single_backslash then

```



```

11251         display_math = display_math + display_math_single
11252         inline_math = inline_math + inline_math_single
11253     end
11254
11255     local TexMath = display_math / writer.display_math
11256                   + inline_math / writer.inline_math
11257
11258     self.insert_pattern("Inline after LinkAndEmph",
11259                       TexMath, "TexMath")
11260
11261     if tex_math_dollars then
11262         self.add_special_character("$")
11263     end
11264
11265     if tex_math_single_backslash or tex_math_double_backslash then
11266         self.add_special_character("\\")
11267         self.add_special_character("[")
11268         self.add_special_character("]")
11269         self.add_special_character("(")
11270         self.add_special_character("(")
11271     end
11272 end
11273 }
11274 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.

```

11275 M.extensions.jekyll_data = function(expect_jekyll_data)
11276     return {
11277         name = "built-in jekyll_data syntax extension",
11278         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`.

```

11279         function self.jekyllData(d, t, p)
11280             if not self.is_writing then return "" end
11281
11282             local buf = {}
11283
11284             local keys = {}

```

```

11285     for k, _ in pairs(d) do
11286         table.insert(keys, k)
11287     end

```

For reproducibility, sort the keys. For mixed string-and-numeric keys, sort numeric keys before string keys.

```

11288         table.sort(keys, function(first, second)
11289             if type(first) ~= type(second) then
11290                 return type(first) < type(second)
11291             else
11292                 return first < second
11293             end
11294         end)
11295
11296     if not p then
11297         table.insert(buf, "\\markdownRendererJekyllDataBegin")
11298     end
11299
11300     local is_sequence = false
11301     if #d > 0 and #d == #keys then
11302         for i=1, #d do
11303             if d[i] == nil then
11304                 goto not_a_sequence
11305             end
11306         end
11307         is_sequence = true
11308     end
11309     ::not_a_sequence::
11310
11311     if is_sequence then
11312         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
11313         table.insert(buf, self.identifier(p or "null"))
11314         table.insert(buf, "}{"")
11315         table.insert(buf, #keys)
11316         table.insert(buf, "}")
11317     else
11318         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
11319         table.insert(buf, self.identifier(p or "null"))
11320         table.insert(buf, "}{"")
11321         table.insert(buf, #keys)
11322         table.insert(buf, "}")
11323     end
11324
11325     for _, k in ipairs(keys) do
11326         local v = d[k]
11327         local typ = type(v)
11328         k = tostring(k or "null")

```

```

11329         if typ == "table" and next(v) ~= nil then
11330             table.insert(
11331                 buf,
11332                 self.jekyllData(v, t, k)
11333             )
11334         else
11335             k = self.identifier(k)
11336             v = tostring(v)
11337             if typ == "boolean" then
11338                 table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
11339                 table.insert(buf, k)
11340                 table.insert(buf, "}{")
11341                 table.insert(buf, v)
11342                 table.insert(buf, "}")
11343             elseif typ == "number" then
11344                 table.insert(buf, "\\markdownRendererJekyllDataNumber{")
11345                 table.insert(buf, k)
11346                 table.insert(buf, "}{")
11347                 table.insert(buf, v)
11348                 table.insert(buf, "}")
11349             elseif typ == "string" then
11350                 table.insert(buf, "\\markdownRendererJekyllDataString{")
11351                 table.insert(buf, k)
11352                 table.insert(buf, "}{")
11353                 table.insert(buf, t(v))
11354                 table.insert(buf, "}")
11355             elseif typ == "table" then
11356                 table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
11357                 table.insert(buf, k)
11358                 table.insert(buf, "}")
11359             else
11360                 error(format("Unexpected type %s for value of " ..
11361                     "YAML key %s", typ, k))
11362             end
11363         end
11364     end
11365
11366     if is_sequence then
11367         table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
11368     else
11369         table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
11370     end
11371
11372     if not p then
11373         table.insert(buf, "\\markdownRendererJekyllDataEnd")
11374     end
11375

```

```

11376     return buf
11377 end
11378 end, extend_reader = function(self)
11379     local parsers = self.parsers
11380     local writer = self.writer
11381
11382     local JekyllData
11383         = Cmt( C((parsers.line - P("----") - P("..."))^0)
11384             , function(s, i, text) -- luacheck: ignore s i
11385                 local data
11386                 local ran_ok, _ = pcall(function()
11387                     -- TODO: Replace with `require("tinyyaml")` in TeX Live
11388                     local tinyyaml = require("markdown-tinyyaml")
11389                     data = tinyyaml.parse(text, {timestamps=false})
11390                 end)
11391                 if ran_ok and data ~= nil then
11392                     return true, writer.jekyllData(data, function(s)
11393                         return self.parser_functions.parse_blocks_nested(s)
11394                     end, nil)
11395                 else
11396                     return false
11397                 end
11398             end
11399         )
11400
11401     local UnexpectedJekyllData
11402         = P("----")
11403         * parsers.blankline / 0
11404         * #(-parsers.blankline) -- if followed by blank, it's thematic b
11405         * JekyllData
11406         * (P("----") + P("..."))
11407
11408     local ExpectedJekyllData
11409         = ( P("----")
11410             * parsers.blankline / 0
11411             * #(-parsers.blankline) -- if followed by blank, it's thematic
11412             )^-1
11413             * JekyllData
11414             * (P("----") + P("..."))^-1
11415
11416     self.insert_pattern("Block before Blockquote",
11417         UnexpectedJekyllData, "UnexpectedJekyllData")
11418     if expect_jekyll_data then
11419         self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
11420     end
11421 end
11422 }

```

11423 end

3.1.8 Conversion from Markdown to Plain T_EX

The `new` function returns a conversion function that takes a markdown string and turns it into a plain T_EX output. See Section 2.1.1.

11424 function M.new(options)

 Make the `options` table inherit from the `defaultOptions` table.

```
11425 options = options or {}
11426 setmetatable(options, { __index = function (_, key)
11427     return defaultOptions[key] end })
```

 If the singleton cache contains a conversion function for the same `options`, reuse it.

```
11428 if options.singletonCache and singletonCache.convert then
11429     for k, v in pairs(defaultOptions) do
11430         if type(v) == "table" then
11431             for i = 1, math.max(#singletonCache.options[k], #options[k]) do
11432                 if singletonCache.options[k][i] ~= options[k][i] then
11433                     goto miss
11434                 end
11435             end
11436             elseif singletonCache.options[k] ~= options[k] then
11437                 goto miss
11438             end
11439         end
11440     return singletonCache.convert
11441 end
11442 ::miss::
```

 Apply built-in syntax extensions based on `options`.

```
11443 local extensions = {}
11444
11445 if options.bracketedSpans then
11446     local bracketed_spans_extension = M.extensions.bracketed_spans()
11447     table.insert(extensions, bracketed_spans_extension)
11448 end
11449
11450 if options.contentBlocks then
11451     local content_blocks_extension = M.extensions.content_blocks(
11452         options.contentBlocksLanguageMap)
11453     table.insert(extensions, content_blocks_extension)
11454 end
11455
11456 if options.definitionLists then
11457     local definition_lists_extension = M.extensions.definition_lists(
11458         options.tightLists)
```

```

11459     table.insert(extensions, definition_lists_extension)
11460 end
11461
11462 if options.fencedCode then
11463     local fenced_code_extension = M.extensions.fenced_code(
11464         options.blankBeforeCodeFence,
11465         options.fencedCodeAttributes,
11466         options.rawAttribute)
11467     table.insert(extensions, fenced_code_extension)
11468 end
11469
11470 if options.fencedDivs then
11471     local fenced_div_extension = M.extensions.fenced_divs(
11472         options.blankBeforeDivFence)
11473     table.insert(extensions, fenced_div_extension)
11474 end
11475
11476 if options.headerAttributes then
11477     local header_attributes_extension = M.extensions.header_attributes()
11478     table.insert(extensions, header_attributes_extension)
11479 end
11480
11481 if options.inlineCodeAttributes then
11482     local inline_code_attributes_extension =
11483         M.extensions.inline_code_attributes()
11484     table.insert(extensions, inline_code_attributes_extension)
11485 end
11486
11487 if options.jekyllData then
11488     local jekyll_data_extension = M.extensions.jekyll_data(
11489         options.expectJekyllData)
11490     table.insert(extensions, jekyll_data_extension)
11491 end
11492
11493 if options.linkAttributes then
11494     local link_attributes_extension =
11495         M.extensions.link_attributes()
11496     table.insert(extensions, link_attributes_extension)
11497 end
11498
11499 if options.lineBlocks then
11500     local line_block_extension = M.extensions.line_blocks()
11501     table.insert(extensions, line_block_extension)
11502 end
11503
11504 if options.mark then
11505     local mark_extension = M.extensions.mark()

```

```

11506     table.insert(extensions, mark_extension)
11507 end
11508
11509 if options.pipeTables then
11510     local pipe_tables_extension = M.extensions.pipe_tables(
11511         options.tableCaptions, options.tableAttributes)
11512     table.insert(extensions, pipe_tables_extension)
11513 end
11514
11515 if options.rawAttribute then
11516     local raw_inline_extension = M.extensions.raw_inline()
11517     table.insert(extensions, raw_inline_extension)
11518 end
11519
11520 if options.strikeThrough then
11521     local strike_through_extension = M.extensions.strike_through()
11522     table.insert(extensions, strike_through_extension)
11523 end
11524
11525 if options.subscripts then
11526     local subscript_extension = M.extensions.subscripts()
11527     table.insert(extensions, subscript_extension)
11528 end
11529
11530 if options.superscripts then
11531     local superscript_extension = M.extensions.superscripts()
11532     table.insert(extensions, superscript_extension)
11533 end
11534
11535 if options.texMathDollars or
11536     options.texMathSingleBackslash or
11537     options.texMathDoubleBackslash then
11538     local tex_math_extension = M.extensions.tex_math(
11539         options.texMathDollars,
11540         options.texMathSingleBackslash,
11541         options.texMathDoubleBackslash)
11542     table.insert(extensions, tex_math_extension)
11543 end
11544
11545 if options.notes or options.inlineNotes then
11546     local notes_extension = M.extensions.notes(
11547         options.notes, options.inlineNotes)
11548     table.insert(extensions, notes_extension)
11549 end
11550
11551 if options.citations then
11552     local citations_extension = M.extensions.citations(options.citationNbsps)

```

```

11553     table.insert(extensions, citations_extension)
11554 end
11555
11556 if options.fancyLists then
11557     local fancy_lists_extension = M.extensions.fancy_lists()
11558     table.insert(extensions, fancy_lists_extension)
11559 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

11560 for _, user_extension_filename in ipairs(options.extensions) do
11561     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

11562         local pathname = kpse.lookup(filename)
11563         local input_file = assert(io.open(pathname, "r"),
11564             [[Could not open user-defined syntax extension "]]
11565             .. pathname .. [[" for reading]])
11566         local input = assert(input_file:read("*a"))
11567         assert(input_file:close())
11568         local user_extension, err = load([[
11569             local sandbox = {}
11570             setmetatable(sandbox, {__index = _G})
11571             _ENV = sandbox
11572         ]] .. input)()
11573         assert(user_extension,
11574             [[Failed to compile user-defined syntax extension "]]
11575             .. pathname .. [[: ]] .. (err or []))

```

Then, validate the user-defined syntax extension.

```

11576         assert(user_extension.api_version ~= nil,
11577             [[User-defined syntax extension "]] .. pathname
11578             .. [[" does not specify mandatory field "api_version"]])
11579         assert(type(user_extension.api_version) == "number",
11580             [[User-defined syntax extension "]] .. pathname
11581             .. [[" specifies field "api_version" of type "]]
11582             .. type(user_extension.api_version)
11583             .. [[" but "number" was expected]])
11584         assert(user_extension.api_version > 0
11585             and user_extension.api_version <= metadata.user_extension_api_version,
11586             [[User-defined syntax extension "]] .. pathname
11587             .. [[" uses syntax extension API version "]]
11588             .. user_extension.api_version .. [[" but markdown.lua ]]
11589             .. metadata.version .. [[" uses API version ]]
11590             .. metadata.user_extension_api_version
11591             .. [[" which is incompatible]])
11592
11593         assert(user_extension.grammar_version ~= nil,
11594             [[User-defined syntax extension "]] .. pathname

```



```

11595     .. [{" does not specify mandatory field "grammar_version"}])
11596     assert(type(user_extension.grammar_version) == "number",
11597           [{"User-defined syntax extension "} .. pathname
11598           .. [{" specifies field "grammar_version" of type "}]]
11599           .. type(user_extension.grammar_version)
11600           .. [{" but "number" was expected}])
11601     assert(user_extension.grammar_version == metadata.grammar_version,
11602           [{"User-defined syntax extension "} .. pathname
11603           .. [{" uses grammar version "} .. user_extension.grammar_version
11604           .. [{" but markdown.lua }] .. metadata.version
11605           .. [{" uses grammar version }] .. metadata.grammar_version
11606           .. [{" , which is incompatible}])
11607
11608     assert(user_extension.finalize_grammar ~= nil,
11609           [{"User-defined syntax extension "} .. pathname
11610           .. [{" does not specify mandatory "finalize_grammar" field}])
11611     assert(type(user_extension.finalize_grammar) == "function",
11612           [{"User-defined syntax extension "} .. pathname
11613           .. [{" specifies field "finalize_grammar" of type "}]]
11614           .. type(user_extension.finalize_grammar)
11615           .. [{" 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.)

```

11616     local extension = {
11617       name = [{"user-defined "} .. pathname .. [{" syntax extension}],
11618       extend_reader = user_extension.finalize_grammar,
11619       extend_writer = function() end,
11620     }
11621     return extension
11622   end)(user_extension_filename)
11623   table.insert(extensions, user_extension)
11624 end

```

Produce a conversion function from markdown to plain TeX.

```

11625 local writer = M.writer.new(options)
11626 local reader = M.reader.new(writer, options)
11627 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`.

```

11628 collectgarbage("collect")

```

Update the singleton cache.

```

11629 if options.singletonCache then
11630   local singletonCacheOptions = {}
11631   for k, v in pairs(options) do
11632     singletonCacheOptions[k] = v

```

```

11633     end
11634     setmetatable(singletonCacheOptions,
11635         { __index = function (_, key)
11636             return defaultOptions[key] end })
11637     singletonCache.options = singletonCacheOptions
11638     singletonCache.convert = convert
11639 end

```

Return the conversion function from markdown to plain \TeX .

```

11640 return convert
11641 end
11642
11643 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.

```

11644
11645 local input
11646 if input_filename then
11647     local input_file = assert(io.open(input_filename, "r"),
11648         [[Could not open file ]] .. input_filename .. [[ for reading]])
11649     input = assert(input_file:read("*a"))
11650     assert(input_file:close())
11651 else
11652     input = assert(io.read("*a"))
11653 end
11654

```

First, ensure that the `options.cacheDir` directory exists.

```

11655 local lfs = require("lfs")
11656 if options.cacheDir and not lfs.isdir(options.cacheDir) then
11657     assert(lfs.mkdir(options["cacheDir"]))
11658 end

```

If `Kpathsea` has not been loaded before or if `Lua \TeX` has not yet been initialized, configure `Kpathsea` on top of loading it.

```

11659 local kpse
11660 (function()
11661     local should_initialize = package.loaded.kpse == nil
11662         or tex.initialize ~= nil
11663     kpse = require("kpse")
11664     if should_initialize then
11665         kpse.set_program_name("luatex")
11666     end
11667 end)()
11668 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.

```
11669 if metadata.version ~= md.metadata.version then
11670   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
11671       "markdown.lua " .. md.metadata.version .. ".")
11672 end
11673 local convert = md.new(options)
11674 local output = convert(input)
11675
11676 if output_filename then
11677   local output_file = assert(io.open(output_filename, "w"),
11678   [[Could not open file ]] .. output_filename .. [[ for writing]])
11679   assert(output_file:write(output))
11680   assert(output_file:close())
11681 else
11682   assert(io.write(output))
11683 end
```

Remove the `options.cacheDir` directory if it is empty.

```
11684 if options.cacheDir then
11685   lfs.rmdir(options["cacheDir"])
11686 end
```

3.2 Plain T_EX Implementation

The plain T_EX implementation provides macros for the interfacing between T_EX and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain T_EX exposed by the plain T_EX interface (see Section 2.2).

3.2.1 Logging Facilities

```
11687 \ExplSyntaxOn
11688 \cs_if_free:NT
11689   \markdownInfo
11690   {
11691     \cs_new:Npn
11692       \markdownInfo #1
11693       {
11694         \msg_info:nne
11695           { markdown }
11696           { generic-message }
11697           { #1 }
11698       }
11699   }
11700 \cs_if_free:NT
11701   \markdownWarning
```

```

11702 {
11703   \cs_new:Npn
11704     \markdownWarning #1
11705     {
11706       \msg_warning:nne
11707         { markdown }
11708         { generic-message }
11709         { #1 }
11710     }
11711 }
11712 \cs_if_free:NT
11713   \markdownError
11714   {
11715     \cs_new:Npn
11716       \markdownError #1 #2
11717       {
11718         \msg_error:nnee
11719           { markdown }
11720           { generic-message-with-help-text }
11721           { #1 }
11722           { #2 }
11723       }
11724   }
11725 \msg_new:nnn
11726   { markdown }
11727   { generic-message }
11728   { #1 }
11729 \msg_new:nnnn
11730   { markdown }
11731   { generic-message-with-help-text }
11732   { #1 }
11733   { #2 }
11734 \cs_generate_variant:Nn
11735   \msg_info:nnn
11736   { nne }
11737 \cs_generate_variant:Nn
11738   \msg_warning:nnn
11739   { nne }
11740 \cs_generate_variant:Nn
11741   \msg_error:nnnn
11742   { nnee }
11743 \ExplSyntaxOff

```

3.2.2 Themes

This section implements the theme-loading mechanism and the built-in themes

provided with the Markdown package. Furthermore, this section also implements the built-in plain T_EX themes provided with the Markdown package.

```

11744 \ExplSyntaxOn
11745 \prop_new:N \g_@@_plain_tex_loaded_themes_linenos_prop
11746 \cs_new:Nn
11747   \@@_plain_tex_load_theme:nn
11748   {
11749     \prop_get:NnNTF
11750       \g_@@_plain_tex_loaded_themes_linenos_prop
11751       { #1 }
11752       \l_tmpa_tl
11753       {
11754         \msg_warning:nnnV
11755           { markdown }
11756           { repeatedly-loaded-plain-tex-theme }
11757           { #1 }
11758           \l_tmpa_tl
11759       }
11760     {
11761       \msg_info:nnn
11762         { markdown }
11763         { loading-plain-tex-theme }
11764         { #1 }
11765       \prop_gput:Nnx
11766         \g_@@_plain_tex_loaded_themes_linenos_prop
11767         { #1 }
11768         { \tex_the:D \tex_inputlineno:D }
11769       \file_input:n
11770         { markdown theme #2 }
11771     }
11772   }
11773 \msg_new:nnn
11774   { markdown }
11775   { loading-plain-tex-theme }
11776   { Loading~plain~TeX~Markdown~theme~#1 }
11777 \msg_new:nnn
11778   { markdown }
11779   { repeatedly-loaded-plain-tex-theme }
11780   {
11781     Plain~TeX~Markdown~theme~#1~was~previously~
11782     loaded~on~line~#2,~not~loading~it~again
11783   }
11784 \cs_generate_variant:Nn
11785   \prop_gput:Nnn
11786   { Nnx }
11787 \cs_gset_eq:NN
11788   \@@_load_theme:nn

```

```

11789 \@@_plain_tex_load_theme:nn
11790 \cs_generate_variant:Nn
11791 \@@_load_theme:nn
11792 { nV }

```

Developers can use the `\markdownLoadPlainTeXTheme` macro to load a corresponding plain TeX theme from within themes for higher-level TeX formats such as L^AT_EX and ConT_EXt.

```

11793 \cs_new:Npn
11794 \markdownLoadPlainTeXTheme
11795 {

```

First, we extract the name of the current theme from the `\g_@@_current_theme_tl` macro.

```

11796 \tl_set:NV
11797 \l_tmpa_tl
11798 \g_@@_current_theme_tl
11799 \tl_reverse:N
11800 \l_tmpa_tl
11801 \tl_set:Ne
11802 \l_tmpb_tl
11803 {
11804 \tl_tail:V
11805 \l_tmpa_tl
11806 }
11807 \tl_reverse:N
11808 \l_tmpb_tl

```

Next, we munge the theme name.

```

11809 \str_set:NV
11810 \l_tmpa_str
11811 \l_tmpb_tl
11812 \str_replace_all:Nnn
11813 \l_tmpa_str
11814 { / }
11815 { _ }

```

Finally, we load the plain TeX theme.

```

11816 \@@_plain_tex_load_theme:VV
11817 \l_tmpb_tl
11818 \l_tmpa_str
11819 }
11820 \cs_generate_variant:Nn
11821 \tl_set:Nn
11822 { Ne }
11823 \cs_generate_variant:Nn
11824 \@@_plain_tex_load_theme:nn
11825 { VV }
11826 \ExplSyntaxOff

```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
11827 \markdownSetup {
11828   rendererPrototypes = {
11829     tilde = {~},
11830   },
11831 }
```

The `witiko/markdown/defaults` plain T_EX 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.

```
11832 \def\markdownRendererInterblockSeparatorPrototype{\par}%
11833 \def\markdownRendererParagraphSeparatorPrototype{%
11834   \markdownRendererInterblockSeparator}%
11835 \def\markdownRendererHardLineBreakPrototype{\hfil\break}%
11836 \def\markdownRendererSoftLineBreakPrototype{ }%
11837 \let\markdownRendererEllipsisPrototype\dots
11838 \def\markdownRendererNbspPrototype{~}%
11839 \def\markdownRendererLeftBracePrototype{\char` \{}%
11840 \def\markdownRendererRightBracePrototype{\char` \}}%
11841 \def\markdownRendererDollarSignPrototype{\char` $}%
11842 \def\markdownRendererPercentSignPrototype{\char` \%}%
11843 \def\markdownRendererAmpersandPrototype{\&}%
11844 \def\markdownRendererUnderscorePrototype{\char` _}%
11845 \def\markdownRendererHashPrototype{\char` #}%
11846 \def\markdownRendererCircumflexPrototype{\char` ^}%
11847 \def\markdownRendererBackslashPrototype{\char` \}%
11848 \def\markdownRendererTildePrototype{\char` ~}%
11849 \def\markdownRendererPipePrototype{|}%
11850 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
11851 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
11852 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
11853   \markdownInput{#3}}%
11854 \def\markdownRendererContentBlockOnlineImagePrototype{%
11855   \markdownRendererImage}%
11856 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
11857   \markdownRendererInputFencedCode{#3}{#2}{#2}}%
11858 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
11859 \def\markdownRendererUlBeginPrototype{ }%
11860 \def\markdownRendererUlBeginTightPrototype{ }%
11861 \def\markdownRendererUlItemPrototype{ }%
11862 \def\markdownRendererUlItemEndPrototype{ }%
11863 \def\markdownRendererUlEndPrototype{ }%
11864 \def\markdownRendererUlEndTightPrototype{ }
```

```

11865 \def\markdownRendererOlBeginPrototype{}%
11866 \def\markdownRendererOlBeginTightPrototype{}%
11867 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
11868 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
11869 \def\markdownRendererOlItemPrototype{}%
11870 \def\markdownRendererOlItemWithNumberPrototype#1{}%
11871 \def\markdownRendererOlItemEndPrototype{}%
11872 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
11873 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}
11874 \def\markdownRendererFancyOlItemEndPrototype{}%
11875 \def\markdownRendererOlEndPrototype{}%
11876 \def\markdownRendererOlEndTightPrototype{}%
11877 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
11878 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
11879 \def\markdownRendererDlBeginPrototype{}%
11880 \def\markdownRendererDlBeginTightPrototype{}%
11881 \def\markdownRendererDlItemPrototype#1{#1}%
11882 \def\markdownRendererDlItemEndPrototype{}%
11883 \def\markdownRendererDlDefinitionBeginPrototype{}%
11884 \def\markdownRendererDlDefinitionEndPrototype{\par}%
11885 \def\markdownRendererDlEndPrototype{}%
11886 \def\markdownRendererDlEndTightPrototype{}%
11887 \def\markdownRendererEmphasisPrototype#1{\it#1}%
11888 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
11889 \def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
11890 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
11891 \def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
11892 \def\markdownRendererLineBlockEndPrototype{\endgroup}%
11893 \def\markdownRendererInputVerbatimPrototype#1{%
11894   \par{\tt\input#1\relax}}\par}%
11895 \def\markdownRendererInputFencedCodePrototype#1#2#3{%
11896   \markdownRendererInputVerbatim{#1}}%
11897 \def\markdownRendererHeadingOnePrototype#1{#1}%
11898 \def\markdownRendererHeadingTwoPrototype#1{#1}%
11899 \def\markdownRendererHeadingThreePrototype#1{#1}%
11900 \def\markdownRendererHeadingFourPrototype#1{#1}%
11901 \def\markdownRendererHeadingFivePrototype#1{#1}%
11902 \def\markdownRendererHeadingSixPrototype#1{#1}%
11903 \def\markdownRendererThematicBreakPrototype{}%
11904 \def\markdownRendererNotePrototype#1{#1}%
11905 \def\markdownRendererCitePrototype#1{}%
11906 \def\markdownRendererTextCitePrototype#1{}%
11907 \def\markdownRendererTickedBoxPrototype{[X]}%
11908 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
11909 \def\markdownRendererUntickedBoxPrototype{[ ]}%
11910 \def\markdownRendererStrikeThroughPrototype#1{#1}%
11911 \def\markdownRendererSuperscriptPrototype#1{#1}%

```



```

11912 \def\markdownRendererSubscriptPrototype#1{#1}%
11913 \def\markdownRendererDisplayMathPrototype#1{##1$$}%
11914 \def\markdownRendererInlineMathPrototype#1{##1$}%
11915 \ExplSyntaxOn
11916 \cs_gset:Npn
11917   \markdownRendererHeaderAttributeContextBeginPrototype
11918   {
11919     \group_begin:
11920     \color_group_begin:
11921   }
11922 \cs_gset:Npn
11923   \markdownRendererHeaderAttributeContextEndPrototype
11924   {
11925     \color_group_end:
11926     \group_end:
11927   }
11928 \cs_gset_eq:NN
11929   \markdownRendererBracketedSpanAttributeContextBeginPrototype
11930   \markdownRendererHeaderAttributeContextBeginPrototype
11931 \cs_gset_eq:NN
11932   \markdownRendererBracketedSpanAttributeContextEndPrototype
11933   \markdownRendererHeaderAttributeContextEndPrototype
11934 \cs_gset_eq:NN
11935   \markdownRendererFencedDivAttributeContextBeginPrototype
11936   \markdownRendererHeaderAttributeContextBeginPrototype
11937 \cs_gset_eq:NN
11938   \markdownRendererFencedDivAttributeContextEndPrototype
11939   \markdownRendererHeaderAttributeContextEndPrototype
11940 \cs_gset_eq:NN
11941   \markdownRendererFencedCodeAttributeContextBeginPrototype
11942   \markdownRendererHeaderAttributeContextBeginPrototype
11943 \cs_gset_eq:NN
11944   \markdownRendererFencedCodeAttributeContextEndPrototype
11945   \markdownRendererHeaderAttributeContextEndPrototype
11946 \cs_gset:Npn
11947   \markdownRendererReplacementCharacterPrototype
11948   { \codepoint_str_generate:n { fffd } }
11949 \ExplSyntaxOff
11950 \def\markdownRendererSectionBeginPrototype{}%
11951 \def\markdownRendererSectionEndPrototype{}%

```

3.2.3.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

11952 \ExplSyntaxOn

```

```

11953 \cs_new:Nn
11954   \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11955   {
11956     \str_case:nn
11957       { #2 }
11958       {
11959         { md } { \markdownInput{#1} }
11960         { tex } { \markdownEscape{#1} \unskip }
11961       }
11962   }
11963 \cs_new:Nn
11964   \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11965   {
11966     \str_case:nn
11967       { #2 }
11968       {
11969         { md } { \markdownInput{#1} }
11970         { tex } { \markdownEscape{#1} }
11971       }
11972   }
11973 \cs_gset:Npn
11974   \markdownRendererInputRawInlinePrototype#1#2
11975   {
11976     \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
11977       { #1 }
11978       { #2 }
11979   }
11980 \cs_gset:Npn
11981   \markdownRendererInputRawBlockPrototype#1#2
11982   {
11983     \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
11984       { #1 }
11985       { #2 }
11986   }
11987 \ExplSyntaxOff

```

3.2.3.2 YAML Metadata Renderer Prototypes

To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position p :

`\c_@@_jekyll_data_sequence_t1` The currently traversed branch of the YAML document contains a sequence at depth p .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth p .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth p .

```
11988 \ExplSyntaxOn
11989 \seq_new:N \g_@@_jekyll_data_datatypes_seq
11990 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
11991 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
11992 \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.

```
11993 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
11994 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
11995 {
11996   \seq_if_empty:NF
11997     \g_@@_jekyll_data_datatypes_seq
11998     {
11999       \seq_get_right:NN
12000       \g_@@_jekyll_data_datatypes_seq
12001       \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.

```
12002   \str_if_eq:NNTF
12003     \l_tmpa_tl
12004     \c_@@_jekyll_data_sequence_tl
12005     {
12006       \seq_put_right:Nn
12007         \g_@@_jekyll_data_wildcard_absolute_address_seq
12008         { * }
12009     }
12010   {
12011     \seq_put_right:Nn
12012       \g_@@_jekyll_data_wildcard_absolute_address_seq
12013       { #1 }
12014   }
12015 }
12016 }
```

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.

```
12017 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
12018 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
12019 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
12020 {
12021   \seq_pop_left:NN #1 \l_tmpa_tl
12022   \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
12023   \seq_put_left:NV #1 \l_tmpa_tl
12024 }
12025 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
12026 {
12027   \markdown_jekyll_data_concatenate_address:NN
12028   \g_@@_jekyll_data_wildcard_absolute_address_seq
12029   \g_@@_jekyll_data_wildcard_absolute_address_tl
12030   \seq_get_right:NN
12031   \g_@@_jekyll_data_wildcard_absolute_address_seq
12032   \g_@@_jekyll_data_wildcard_relative_address_tl
12033 }
```

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.

```
12034 \cs_new:Nn \markdown_jekyll_data_push:nN
12035 {
12036   \markdown_jekyll_data_push_address_segment:n
```

```

12037     { #1 }
12038     \seq_put_right:NV
12039     \g_@@_jekyll_data_datatypes_seq
12040     #2
12041     \markdown_jekyll_data_update_address_tls:
12042   }
12043 \cs_new:Nn \markdown_jekyll_data_pop:
12044 {
12045     \seq_pop_right:NN
12046     \g_@@_jekyll_data_wildcard_absolute_address_seq
12047     \l_tmpa_tl
12048     \seq_pop_right:NN
12049     \g_@@_jekyll_data_datatypes_seq
12050     \l_tmpa_tl
12051     \markdown_jekyll_data_update_address_tls:
12052 }

```

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.

```

12053 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
12054 {
12055     \keys_set_known:nn
12056     { markdown/jekyllData }
12057     { { #1 } = { #2 } }
12058 }
12059 \cs_generate_variant:Nn
12060 \markdown_jekyll_data_set_keyval:nn
12061 { Vn }
12062 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
12063 {
12064     \markdown_jekyll_data_push:nN
12065     { #1 }
12066     \c_@@_jekyll_data_scalar_tl
12067     \markdown_jekyll_data_set_keyval:Vn
12068     \g_@@_jekyll_data_wildcard_absolute_address_tl
12069     { #2 }
12070     \markdown_jekyll_data_set_keyval:Vn
12071     \g_@@_jekyll_data_wildcard_relative_address_tl
12072     { #2 }
12073     \markdown_jekyll_data_pop:
12074 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

12075 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
12076     \markdown_jekyll_data_push:nN
12077     { #1 }

```

```

12078     \c_@@_jekyll_data_sequence_tl
12079 }
12080 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
12081     \markdown_jekyll_data_push:nN
12082     { #1 }
12083     \c_@@_jekyll_data_mapping_tl
12084 }
12085 \def\markdownRendererJekyllDataSequenceEndPrototype{
12086     \markdown_jekyll_data_pop:
12087 }
12088 \def\markdownRendererJekyllDataMappingEndPrototype{
12089     \markdown_jekyll_data_pop:
12090 }
12091 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
12092     \markdown_jekyll_data_set_keyvals:nN
12093     { #1 }
12094     { #2 }
12095 }
12096 \def\markdownRendererJekyllDataEmptyPrototype#1{}
12097 \def\markdownRendererJekyllDataNumberPrototype#1#2{
12098     \markdown_jekyll_data_set_keyvals:nN
12099     { #1 }
12100     { #2 }
12101 }
12102 \def\markdownRendererJekyllDataStringPrototype#1#2{
12103     \markdown_jekyll_data_set_keyvals:nN
12104     { #1 }
12105     { #2 }
12106 }
12107 \ExplSyntaxOff

```

If plain T_EX is the top layer, we load the [witiko/markdown/defaults](#) plain T_EX theme with the default definitions for token renderer prototypes unless the option `noDefaults` has been enabled (see Section 2.2.2.3).

```

12108 \ExplSyntaxOn
12109 \str_if_eq:VVT
12110     \c_@@_top_layer_tl
12111     \c_@@_option_layer_plain_tex_tl
12112     {
12113         \ExplSyntaxOff
12114         \@@_if_option:nF
12115             { noDefaults }
12116             {
12117                 \@@_setup:n
12118                     {theme = witiko/markdown/defaults}
12119             }
12120     }

```

```

12121 }
12122 \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).

```

12123 \ExplSyntaxOn
12124 \tl_new:N \g_@@_formatted_lua_options_tl
12125 \cs_new:Nn \@@_format_lua_options:
12126 {
12127   \tl_gclear:N
12128     \g_@@_formatted_lua_options_tl
12129   \seq_map_function:NN
12130     \g_@@_lua_options_seq
12131     \@@_format_lua_option:n
12132 }
12133 \cs_new:Nn \@@_format_lua_option:n
12134 {
12135   \@@_typecheck_option:n
12136     { #1 }
12137   \@@_get_option_type:nN
12138     { #1 }
12139   \l_tmpa_tl
12140   \bool_case_true:nF
12141     {
12142       {
12143         \str_if_eq_p:VV
12144           \l_tmpa_tl
12145           \c_@@_option_type_boolean_tl ||
12146         \str_if_eq_p:VV
12147           \l_tmpa_tl
12148           \c_@@_option_type_number_tl ||
12149         \str_if_eq_p:VV
12150           \l_tmpa_tl
12151           \c_@@_option_type_counter_tl
12152       }
12153     {
12154       \@@_get_option_value:nN
12155         { #1 }
12156       \l_tmpa_tl
12157       \tl_gput_right:Nx
12158         \g_@@_formatted_lua_options_tl
12159         { #1~== \l_tmpa_tl ,~ }
12160     }

```

```

12161     {
12162     \str_if_eq_p:VV
12163     \l_tmpa_tl
12164     \c_@@_option_type_clist_tl
12165     }
12166     {
12167     \@@_get_option_value:nN
12168     { #1 }
12169     \l_tmpa_tl
12170     \tl_gput_right:Nx
12171     \g_@@_formatted_lua_options_tl
12172     { #1~::~\c_left_brace_str }
12173     \clist_map_inline:Vn
12174     \l_tmpa_tl
12175     {
12176     \tl_gput_right:Nx
12177     \g_@@_formatted_lua_options_tl
12178     { "##1" ,~ }
12179     }
12180     \tl_gput_right:Nx
12181     \g_@@_formatted_lua_options_tl
12182     { \c_right_brace_str ,~ }
12183     }
12184     }
12185     {
12186     \@@_get_option_value:nN
12187     { #1 }
12188     \l_tmpa_tl
12189     \tl_gput_right:Nx
12190     \g_@@_formatted_lua_options_tl
12191     { #1~::~ "\l_tmpa_tl " ,~ }
12192     }
12193     }
12194 \cs_generate_variant:Nn
12195 \clist_map_inline:nn
12196 { Vn }
12197 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
12198 \def\markdownLuaOptions{ \g_@@_formatted_lua_options_tl }
12199 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain \TeX . It exposes the `convert` function for the use by any further Lua code.

```
12200 \def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```
12201   local lfs = require("lfs")
12202   local cacheDir = "\markdownOptionCacheDir"
```



```

12203   if not lfs.isdir(cacheDir) then
12204       assert(lfs.mkdir(cacheDir))
12205   end

```

Next, load the `markdown` module and create a converter function using the plain `TEX` options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

12206   local md = require("markdown")
12207   local convert = md.new(\markdownLuaOptions)
12208 }%

```

The `\markdownCleanup` macro contains the Lua code that is executed after any conversion from markdown to plain `TEX`.

```

12209 \def\markdownCleanup{%

```

Remove the `options.cacheDir` directory if it is empty.

```

12210   lfs.rmdir(cacheDir)
12211 }%

```

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.

```

12212 \csname newread\endcsname\markdownInputFileStream
12213 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

12214 \begingroup
12215   \catcode\^^I=12%
12216   \gdef\markdownReadAndConvertTab{^^I}%
12217 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the `LATEX 2ε` `\filecontents` macro to plain `TEX`.

```

12218 \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.

```

12219   \catcode\^^M=13%
12220   \catcode\^^I=13%
12221   \catcode|=0%
12222   \catcode\=12%
12223   |catcode%=14%
12224   |catcode|=12@
12225   |gdef|markdownReadAndConvert#1#2{|@
12226     |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

12227     |markdownIfOption{frozenCache}{-}{@
12228         |immediate|openout|markdownOutputFileStream@
12229         |markdownOptionInputTempFileName|relax@
12230     |markdownInfo{@
12231         Buffering block-level markdown input into the temporary @
12232         input file "|markdownOptionInputTempFileName" and scanning @
12233         for the closing token sequence "#1"}@
12234     }@

```

Locally change the category of the special plain T_EX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

12235     |def|do##1{|catcode`##1=12}|dospecials@
12236     |catcode`| =12@
12237     |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.

```

12238     |def|markdownReadAndConvertStripPercentSign##1{@
12239         |markdownIfOption{stripPercentSigns}{-}{@
12240             |if##1%{@
12241                 |expandafter|expandafter|expandafter@
12242                 |markdownReadAndConvertProcessLine@
12243             |else@
12244                 |expandafter|expandafter|expandafter@
12245                 |markdownReadAndConvertProcessLine@
12246                 |expandafter|expandafter|expandafter##1@
12247             |fi@
12248         }{-}{@
12249         |expandafter@
12250         |markdownReadAndConvertProcessLine@
12251         |expandafter##1@
12252     }@
12253 }@

```

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.

```

12254     |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName`

file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```

12255     |ifx|relax##3|relax@
12256     |markdownIfOption{frozenCache}{-}{@
12257     |immediate|write|markdownOutputFileStream{##1}@
12258     }@
12259     |else@

```

When the ending token sequence appears in the line, make the next newline character close the `inputTempFileName` file, return the character categories back to the former state, convert the `inputTempFileName` file from markdown to plain T_EX, `\input` the result of the conversion, and expand the ending control sequence.

```

12260     |def^^M{@
12261     |markdownInfo{The ending token sequence was found}@
12262     |markdownIfOption{frozenCache}{-}{@
12263     |immediate|closeout|markdownOutputFileStream@
12264     }@
12265     |endgroup@
12266     |markdownInput{@
12267     |markdownOptionOutputDir@
12268     /|markdownOptionInputTempFileName@
12269     }@
12270     #2}@
12271     |fi@

```

Repeat with the next line.

```

12272     ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

12273     |catcode`|^I=13@
12274     |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.

```

12275     |catcode`|^M=13@
12276     |def^^M##1^^M{@
12277     |def^^M###1^^M{@
12278     |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
12279     ^^M}@
12280     ^^M}@

```

Reset the character categories back to the former state.

```

12281 |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.

```

12282 \ExplSyntaxOn
12283 \cs_new:Npn
12284   \markdownLuaExecute
12285   #1
12286   {
12287     \int_compare:nNnT
12288       { \g_luabridge_method_int }
12289       =
12290       { \c_luabridge_method_shell_int }
12291       {
12292         \sys_if_shell_unrestricted:F
12293         {
12294           \sys_if_shell:TF
12295           {
12296             \msg_error:nn
12297               { markdown }
12298               { restricted-shell-access }
12299           }
12300           {
12301             \msg_error:nn
12302               { markdown }
12303               { disabled-shell-access }
12304           }
12305         }
12306       }
12307     \str_gset:NV
12308       \g_luabridge_output_dirname_str
12309       \markdownOptionOutputDir
12310     \luabridge_now:e
12311     { #1 }
12312   }
12313 \cs_generate_variant:Nn
12314   \msg_new:nnnn
12315   { nnnV }
12316 \tl_set:Nn
12317   \l_tmpa_tl
12318   {
12319     You~may~need~to~run~TeX~with~the~--shell-escape~or~the~
12320     --enable-write18~flag,~or~write~shell_escape=t~in~the~
12321     texmf.cnf~file.
12322   }
12323 \msg_new:nnnV
12324   { markdown }
12325   { restricted-shell-access }
12326   { Shell~escape~is~restricted }
12327   \l_tmpa_tl
12328 \msg_new:nnnV

```

```

12329 { markdown }
12330 { disabled-shell-access }
12331 { Shell~escape~is~disabled }
12332 \l_tmpa_tl
12333 \ExplSyntaxOff

```

3.2.6 Buffering Inline Markdown Input

This section describes the implementation of the macro `\markinline`.

```

12334 \ExplSyntaxOn
12335 \tl_new:N
12336 \g_@@_after_markinline_tl
12337 \tl_gset:Nn
12338 \g_@@_after_markinline_tl
12339 { \unskip }
12340 \cs_new:Npn
12341 \markinline
12342 {

```

Locally change the category of the special plain \TeX characters to *other* in order to prevent unwanted interpretation of the input markdown text as \TeX code.

```

12343 \group_begin:
12344 \cctab_select:N
12345 \c_other_cctab

```

Unless we are reading markdown documents from the frozen cache, open the file `inputTempFileName` for writing.

```

12346 \@@_if_option:nF
12347 { frozenCache }
12348 {
12349 \immediate
12350 \openout
12351 \markdownOutputFileStream
12352 \markdownOptionInputTempFileName
12353 \relax
12354 \msg_info:nne
12355 { markdown }
12356 { buffering-markinline }
12357 { \markdownOptionInputTempFileName }
12358 }

```

Peek ahead and extract the inline markdown text.

```

12359 \peek_regex_replace_once:nnF
12360 { { (.*) } }
12361 {

```

Unless we are reading markdown documents from the frozen cache, store the text in the file `inputTempFileName` and close it.

```

12362     \c { @@_if_option:nF }
12363     \cB { frozenCache \cE }
12364     \cB {
12365         \c { immediate }
12366         \c { write }
12367         \c { markdownOutputFileStream }
12368         \cB { \1 \cE }
12369         \c { immediate }
12370         \c { closeout }
12371         \c { markdownOutputFileStream }
12372     \cE }

```

Reset the category codes and `\input` the result of the conversion.

```

12373     \c { group_end: }
12374     \c { group_begin: }
12375     \c { @@_setup:n }
12376     \cB { contentLevel = inline \cE }
12377     \c { markdownInput }
12378     \cB {
12379         \c { markdownOptionOutputDir } /
12380         \c { markdownOptionInputTempFileName }
12381     \cE }
12382     \c { group_end: }
12383     \c { tl_use:N }
12384     \c { g_@@_after_markinline_tl }
12385 }
12386 {
12387     \msg_error:nn
12388     { markdown }
12389     { markinline-peek-failure }
12390     \group_end:
12391     \tl_use:N
12392     \g_@@_after_markinline_tl
12393 }
12394 }
12395 \msg_new:nnn
12396 { markdown }
12397 { buffering-markinline }
12398 { Buffering~inline~markdown~input~into~the~temporary~input~file~"#1". }
12399 \msg_new:nnnn
12400 { markdown }
12401 { markinline-peek-failure }
12402 { Use-of~\iow_char:N \\ markinline~doesn't~match~its~definition }
12403 { The~macro~should~be~followed~by~inline~markdown~text~in~curly~braces }
12404 \ExplSyntaxOff

```

3.2.7 Typesetting Markdown

The `\markdownInput` macro uses an implementation of the `\markdownLuaExecute` macro to convert the contents of the file whose filename it has received as its single argument from markdown to plain T_EX.

```
12405 \ExplSyntaxOn
12406 \cs_new:Npn
12407   \markdownInput
12408   #1
12409   {
```

If the file does not exist in the current directory, we will search for it in the directories specified in `\l_file_search_path_seq`. On L^AT_EX, this also includes the directories specified in `\input@path`.

```
12410     \file_get_full_name:nNTF
12411     { #1 }
12412     \l_tmpa_tl
12413     {
12414       \exp_args:NV
12415         \markdownInputRaw
12416         \l_tmpa_tl
12417     }
12418     {
12419       \msg_error:nnnV
12420       { markdown }
12421       { markdown-file-does-not-exist }
12422       { #1 }
12423     }
12424   }
12425 \msg_new:nnn
12426   { markdown }
12427   { markdown-file-does-not-exist }
12428   {
12429     Markdown~file~#1~does-not~exist
12430   }
12431 \ExplSyntaxOff
12432 \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.

```
12433 \catcode`\|=0%
12434 \catcode`\|=12%
12435 \catcode`\&=6%
12436 \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.

```
12437     |begingroup
12438     |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.

```
12439     |catcode`\#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
12440     |markdownIfOption{frozenCache}{%
12441         |ifnum|markdownOptionFrozenCacheCounter=0|relax
12442         |markdownInfo{Reading frozen cache from
12443             "|markdownOptionFrozenCacheFileName"}%
12444         |input|markdownOptionFrozenCacheFileName|relax
12445         |fi
12446         |markdownInfo{Including markdown document number
12447             "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
12448         |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
12449         |global|advance|markdownOptionFrozenCacheCounter by 1|relax
12450     }{%
12451         |markdownInfo{Including markdown document "&1"}%
```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as L^AT_EX_Mk to track changes to the markdown document.

```
12452         |openin|markdownInputFileStream&1
12453         |closein|markdownInputFileStream
12454         |markdownPrepareLuaOptions
12455         |markdownLuaExecute{%
12456             |markdownPrepare
12457             local file = assert(io.open("&1", "r"),
12458                 [[Could not open file "&1" for reading]])
12459             local input = assert(file:read("*a"))
12460             assert(file:close())
12461             print(convert(input))
12462         |markdownCleanup}%
```

If we are finalizing the frozen cache, increment `frozenCacheCounter`.

```
12463         |markdownIfOption{finalizeCache}{%
12464             |global|advance|markdownOptionFrozenCacheCounter by 1|relax}{}%
12465     }%
12466     |endgroup
12467 }%
12468 |endgroup
```


The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of \TeX to execute a \TeX document in the middle of a markdown document fragment.

```
12469 \gdef\markdownEscape#1{%
12470   \catcode`\%=14\relax
12471   \catcode`\#=6\relax
12472   \input #1\relax
12473   \catcode`\%=12\relax
12474   \catcode`\#=12\relax
12475 }%
```

3.3 \LaTeX Implementation

The \LaTeX implementation makes use of the fact that, apart from some subtle differences, \LaTeX implements the majority of the plain \TeX format [12, Section 9]. As a consequence, we can directly reuse the existing plain \TeX implementation.

```
12476 \def\markdownVersionSpace{ }%
12477 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
12478   \markdownVersion\markdownVersionSpace markdown renderer]%
```

3.3.1 Typesetting Markdown

The `\markinlinePlainTeX` macro is used to store the original plain \TeX implementation of the `\markinline` macro. The `\markinline` macro is then redefined to accept an optional argument with options recognized by the \LaTeX interface (see Section 2.3.2).

```
12479 \ExplSyntaxOn
12480 \cs_gset_eq:NN
12481   \markinlinePlainTeX
12482   \markinline
12483 \cs_gset:Npn
12484   \markinline
12485   {
12486     \peek_regex_replace_once:nn
12487       { ( \[ (.*) \] ) ? }
12488     {
```

Apply the options locally.

```
12489     \c { group_begin: }
12490     \c { @@_setup:n }
12491     \cB { \2 \cE }
12492     \c { tl_put_right:Nn }
12493     \c { g_@@_after_markinline_tl }
12494     \cB { \c { group_end: } \cE }
```

```

12495     \c { markinlinePlainTeX }
12496     }
12497   }
12498 \ExplSyntaxOff

```

The `\markdownInputPlainTeX` macro is used to store the original plain \TeX implementation of the `\markdownInput` macro. The `\markdownInput` macro is then redefined to accept an optional argument with options recognized by the \LaTeX interface (see Section 2.3.2).

```

12499 \let\markdownInputPlainTeX\markdownInput
12500 \renewcommand\markdownInput[2] [] {%
12501   \begingroup
12502     \markdownSetup{#1}%
12503     \markdownInputPlainTeX{#2}%
12504   \endgroup}%

```

The `markdown`, and `markdown*` \LaTeX environments are implemented using the `\markdownReadAndConvert` macro.

```

12505 \ExplSyntaxOn
12506 \renewenvironment
12507   { markdown }
12508   {

```

In our implementation of the `markdown` \LaTeX environment, we want to distinguish between the following two cases:

| | |
|---|-------------------------------|
| <code>\begin{markdown} [smartEllipses]</code> | <code>\begin{markdown}</code> |
| <code>% This is an optional argument ^</code> | <code>[smartEllipses]</code> |
| <code>% ...</code> | <code>% ^ This is link</code> |
| <code>\end{markdown}</code> | <code>\end{markdown}</code> |

Therefore, we cannot use the built-in \LaTeX support for environments with optional arguments or packages such as `xparse`. Instead, we must read the optional argument manually and prevent reading past the end of a line.

To prevent reading past the end of a line when looking for the optional argument of the `markdown` \LaTeX environment and accidentally tokenizing markdown text, we change the category code of carriage return (`\r`, ASCII character 13 in decimal) from 5 (end of line).

While any category code other than 5 (end of line) would work, we switch to the category 13 (active), which is also used by the `\markdownReadAndConvert` macro. This is necessary if we read until the end of a line, because then the carriage return character will be produced by \TeX via the `\endlinechar` plain \TeX macro and it needs to have the correct category code, so that `\markdownReadAndConvert` processes it correctly.

```

12509   \group_begin:
12510   \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).

```
12511 \char_set_catcode_letter:n { 35 }
```

After we have matched the opening [that begins the optional argument, we accept carriage returns as well.

```
12512 \peek_regex_replace_once:nnF
12513 { \ *\[r*([^\r]*)\][^\r]* }
12514 {
```

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.

```
12515 \c { group_end: }
12516 \c { tl_set_rescan:Nnn } \c { l_tmpa_tl } { } { \1 }
```

Then, we pass the retokenized content to the `\markdownSetup` macro.

```
12517 \c { @@_setup:V } \c { l_tmpa_tl }
```

Finally, regardless of whether or not we have matched the optional argument, we let the `\markdownReadAndConvert` macro process the rest of the \LaTeX environment.

```
12518 \c { markdownReadAndConvert@markdown } { }
12519 }
12520 {
12521 \group_end:
12522 \markdownReadAndConvert@markdown { }
12523 }
12524 }
12525 { \markdownEnd }
12526 \renewenvironment
12527 { markdown* }
12528 [ 1 ]
12529 {
12530 \msg_warning:nnn
12531 { markdown }
12532 { latex-markdown-star-deprecated }
12533 { #1 }
12534 \@@_setup:n
12535 { #1 }
12536 \markdownReadAndConvert@markdown *
12537 }
12538 { \markdownEnd }
12539 \msg_new:nnn
12540 { markdown }
```

```

12541 { latex-markdown-star-deprecated }
12542 {
12543   The~markdown*~LaTeX~environment~has~been~deprecated~and~will~
12544   be~removed~in~the~next~major~version~of~the~Markdown~package.
12545 }
12546 \ExplSyntaxOff
12547 \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*.

```

12548 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
12549 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
12550 |gdef|markdownReadAndConvert@markdown#1<%
12551   |markdownReadAndConvert<\end{markdown#1}>%
12552   <|end<markdown#1>>>%
12553 |endgroup

```

3.3.2 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

12554 \DeclareOption*{%
12555   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
12556 \ProcessOptions\relax

```

3.3.3 Themes

This section overrides the plain $\text{T}_{\text{E}}\text{X}$ implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ themes provided with the Markdown package.

```

12557 \ExplSyntaxOn
12558 \cs_gset:Nn
12559   \@@_load_theme:nn
12560   {

```

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.

```

12561   \ifmarkdownLaTeXLoaded
12562   \ifx\@onlypreamble\@notprerr

```

If both conditions are true does, end with an error, since we cannot load $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ themes after the preamble. Otherwise, try loading a plain $\text{T}_{\text{E}}\text{X}$ theme instead.

```

12563   \file_if_exist:nTF
12564   { markdown theme #2.sty }

```

```

12565     {
12566         \msg_error:nnn
12567         { markdown }
12568         { latex-theme-after-preamble }
12569         { #1 }
12570     }
12571     {
12572         \@@_plain_tex_load_theme:nn
12573         { #1 }
12574         { #2 }
12575     }
12576     \else

```

If the Markdown package has already been loaded but we are still in the preamble, load a L^AT_EX theme if it exists or load a plain T_EX theme otherwise.

```

12577         \file_if_exist:nTF
12578         { markdown theme #2.sty }
12579         {
12580             \msg_info:nnn
12581             { markdown }
12582             { loading-latex-theme }
12583             { #1 }
12584             \RequirePackage
12585             { markdown theme #2 }
12586         }
12587         {
12588             \@@_plain_tex_load_theme:nn
12589             { #1 }
12590             { #2 }
12591         }
12592     \fi
12593     \else

```

If the Markdown package has not yet been loaded, postpone the loading until the Markdown package has finished loading.

```

12594         \msg_info:nnn
12595         { markdown }
12596         { theme-loading-postponed }
12597         { #1 }
12598         \AtEndOfPackage
12599         {
12600             \@@_load_theme:nn
12601             { #1 }
12602             { #2 }
12603         }
12604     \fi
12605 }
12606 \msg_new:nnn

```

```

12607 { markdown }
12608 { theme-loading-postponed }
12609 {
12610   Postponing~loading~Markdown~theme~#1~until~
12611   Markdown~package~has~finished~loading
12612 }
12613 \msg_new:nnn
12614 { markdown }
12615 { loading-latex-theme }
12616 { Loading~LaTeX~Markdown~theme~#1 }
12617 \cs_generate_variant:Nn
12618 \msg_new:nnnn
12619 { nnVV }
12620 \tl_set:Nn
12621 \l_tmpa_tl
12622 { Cannot~load~LaTeX~Markdown~theme~#1~after~ }
12623 \tl_put_right:NV
12624 \l_tmpa_tl
12625 \c_backslash_str
12626 \tl_put_right:Nn
12627 \l_tmpa_tl
12628 { begin{document} }
12629 \tl_set:Nn
12630 \l_tmpb_tl
12631 { Load~Markdown~theme~#1~before~ }
12632 \tl_put_right:NV
12633 \l_tmpb_tl
12634 \c_backslash_str
12635 \tl_put_right:Nn
12636 \l_tmpb_tl
12637 { begin{document} }
12638 \msg_new:nnVV
12639 { markdown }
12640 { latex-theme-after-preamble }
12641 \l_tmpa_tl
12642 \l_tmpb_tl
12643 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
12644 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
12645 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
12646 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
12647 \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot ...`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain T_EX option is disabled and the code block has not been previously typeset:

```

12648 \renewcommand\markdownRendererInputFencedCodePrototype[3]{%
12649   \def\next##1 ##2\relax{%
12650     \ifthenelse{\equal{##1}{dot}}{%
12651       \markdownIfOption{frozenCache}{}{%
12652         \immediate\write18{%
12653           if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
12654           then
12655             dot -Tpdf -o #1.pdf #1;
12656             cp #1 #1.pdf.source;
12657           fi}}%

```

We include the typeset image using the image token renderer:

```

12658   \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:

```

12659   }{%
12660   \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}{#3}%
12661   }%
12662 }%
12663 \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

12664 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
12665 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```

12666 \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.

```

12667 \newcount\markdown@witiko@graphicx@http@counter
12668 \markdown@witiko@graphicx@http@counter=0
12669 \newcommand\markdown@witiko@graphicx@http@filename{%
12670   \markdownOptionCacheDir/witiko_graphicx_http%
12671   .\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.

```

12672 \newcommand\markdown@witiko@graphicx@http@download[2]{%
12673   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:

```

12674 \begingroup
12675 \catcode`\%=12
12676 \catcode`\^^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```

12677 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
12678   \begingroup
12679     \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:

```

12680     \markdownIfOption{frozenCache}{}{^^A
12681       \immediate\write18{^^A
12682         mkdir -p "\markdownOptionCacheDir";
12683         if printf '%s' "#3" | grep -q -E '^https?:';
12684         then

```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

12685         OUTPUT_PREFIX="\markdownOptionCacheDir";
12686         OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
12687         OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
12688         OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

12689         if ! [ -e "$OUTPUT" ];
12690         then
12691           \markdown@witiko@graphicx@http@download{#3}{"$OUTPUT"};
12692           printf '%s' "$OUTPUT" > "\filename";
12693         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:

```

12694         else
12695           printf '%s' '#3' > "\filename";
12696         fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

12697     \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
12698     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
12699     {#1}{#2}{\filename}{#4}^^A
12700 \endgroup

```



```

12701 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax^^A
12702 \endgroup

```

The `witiko/markdown/defaults` L^AT_EX theme provides default definitions for token renderer prototypes. First, the L^AT_EX theme loads the plain T_EX theme with the default definitions for plain T_EX:

```

12703 \markdownLoadPlainTeXTheme

```

Next, the L^AT_EX theme overrides some of the plain T_EX definitions. See Section 3.3.4 for the actual definitions.

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the option `plain` has been enabled (see Section 2.2.2.3), none of the definitions will take effect.

```

12704 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the `paralist` package.

```

12705 \@ifclassloaded{beamer}{}{%
12706   \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
12707   \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
12708 }

```

If we loaded the `paralist` package, define the respective renderer prototypes to make use of the capabilities of the package. Otherwise, define the renderer prototypes to fall back on the corresponding renderers for the non-tight lists.

```

12709 \ExplSyntaxOn
12710 \@ifpackageloaded{paralist}{
12711   \tl_new:N
12712     \l_@@_latex_fancy_list_item_label_number_style_tl
12713   \tl_new:N
12714     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12715   \cs_new:Nn
12716     \@@_latex_fancy_list_item_label_number:nn
12717     {
12718       \str_case:nn
12719         { #1 }
12720         {
12721           { Decimal } { #2 }
12722           { LowerRoman } { \int_to_roman:n { #2 } }
12723           { UpperRoman } { \int_to_Roman:n { #2 } }
12724           { LowerAlpha } { \int_to_alph:n { #2 } }
12725           { UpperAlpha } { \int_to_Alph:n { #2 } }
12726         }
12727     }
12728   \cs_new:Nn
12729     \@@_latex_fancy_list_item_label_delimiter:n

```

```

12730     {
12731     \str_case:nn
12732     { #1 }
12733     {
12734     { Default } { . }
12735     { OneParen } { ) }
12736     { Period } { . }
12737     }
12738     }
12739 \cs_new:Nn
12740 \@@_latex_fancy_list_item_label:nnn
12741 {
12742 \@@_latex_fancy_list_item_label_number:nn
12743 { #1 }
12744 { #3 }
12745 \@@_latex_fancy_list_item_label_delimiter:n
12746 { #2 }
12747 }
12748 \cs_new:Nn
12749 \@@_latex_paralist_style:nn
12750 {
12751 \str_case:nn
12752 { #1 }
12753 {
12754 { Decimal } { 1 }
12755 { LowerRoman } { i }
12756 { UpperRoman } { I }
12757 { LowerAlpha } { a }
12758 { UpperAlpha } { A }
12759 }
12760 \@@_latex_fancy_list_item_label_delimiter:n
12761 { #2 }
12762 }
12763 \markdownSetup{rendererPrototypes={

```

Make tight bullet lists a little less compact by adding extra vertical space above and below them.

```

12764     ulBeginTight = {%
12765     \group_begin:
12766     \pltopsep=\topsep
12767     \plpartopsep=\partopsep
12768     \begin{compactitem}
12769     },
12770     ulEndTight = {
12771     \end{compactitem}
12772     \group_end:
12773     },

```

```

12774 fancyOlBegin = {
12775   \group_begin:
12776   \tl_set:Nn
12777     \l_@@_latex_fancy_list_item_label_number_style_tl
12778     { #1 }
12779   \tl_set:Nn
12780     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12781     { #2 }
12782   \@@_if_option:nTF
12783     { startNumber }
12784     {
12785       \tl_set:Nn
12786         \l_tmpa_tl
12787         { \begin{enumerate} }
12788     }
12789     {
12790       \tl_set:Nn
12791         \l_tmpa_tl
12792         { \begin{enumerate}[ ] }
12793       \tl_put_right:Nx
12794         \l_tmpa_tl
12795         { \@@_latex_paralist_style:mn { #1 } { #2 } }
12796       \tl_put_right:Nn
12797         \l_tmpa_tl
12798         { ] }
12799     }
12800   \tl_use:N
12801     \l_tmpa_tl
12802 },
12803 fancyOlEnd = {
12804   \end{enumerate}
12805   \group_end:
12806 },

```

Make tight ordered lists a little less compact by adding extra vertical space above and below them.

```

12807 olBeginTight = {%
12808   \group_begin:
12809   \plpartopsep=\partopsep
12810   \pltopsep=\topsep
12811   \begin{compactenum}
12812 },
12813 olEndTight = {
12814   \end{compactenum}
12815   \group_end:
12816 },
12817 fancyOlBeginTight = {

```

```

12818 \group_begin:
12819 \tl_set:Nn
12820 \l_@@_latex_fancy_list_item_label_number_style_tl
12821 { #1 }
12822 \tl_set:Nn
12823 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12824 { #2 }
12825 \@@_if_option:nTF
12826 { startNumber }
12827 {
12828 \tl_set:Nn
12829 \l_tmpa_tl
12830 { \begin{compactenum} }
12831 }
12832 {
12833 \tl_set:Nn
12834 \l_tmpa_tl
12835 { \begin{compactenum}[ ] }
12836 \tl_put_right:Nx
12837 \l_tmpa_tl
12838 { \@@_latex_paralist_style:n { #1 } { #2 } }
12839 \tl_put_right:Nn
12840 \l_tmpa_tl
12841 { ] }
12842 }
12843 \tl_put_left:Nn
12844 \l_tmpa_tl
12845 {
12846 \plpartopsep=\partopsep
12847 \pltopsep=\topsep
12848 }
12849 \tl_use:N
12850 \l_tmpa_tl
12851 },
12852 fancyO1EndTight = {
12853 \end{compactenum}
12854 \group_end:
12855 },
12856 fancyO1ItemWithNumber = {
12857 \item
12858 [
12859 \@@_latex_fancy_list_item_label:VVn
12860 \l_@@_latex_fancy_list_item_label_number_style_tl
12861 \l_@@_latex_fancy_list_item_label_delimiter_style_tl
12862 { #1 }
12863 ]
12864 },

```

Make tight definition lists a little less compact by adding extra vertical space above and below them.

```

12865     dlBeginTight = {
12866         \group_begin:
12867         \plpartopsep=\partopsep
12868         \pltopsep=\topsep
12869         \begin{compactdesc}
12870     },
12871     dlEndTight = {
12872         \end{compactdesc}
12873         \group_end:
12874     }}
12875 \cs_generate_variant:Nn
12876 \@@_latex_fancy_list_item_label:nnn
12877 { VVn }
12878 }{
12879 \markdownSetup{rendererPrototypes={
12880     ulBeginTight = {\markdownRendererUlBegin},
12881     ulEndTight = {\markdownRendererUlEnd},
12882     fancyOlBegin = {\markdownRendererOlBegin},
12883     fancyOlEnd = {\markdownRendererOlEnd},
12884     olBeginTight = {\markdownRendererOlBegin},
12885     olEndTight = {\markdownRendererOlEnd},
12886     fancyOlBeginTight = {\markdownRendererOlBegin},
12887     fancyOlEndTight = {\markdownRendererOlEnd},
12888     dlBeginTight = {\markdownRendererDlBegin},
12889     dlEndTight = {\markdownRendererDlEnd}}}
12890 }
12891 \ExplSyntaxOff
12892 \RequirePackage{amsmath}

```

Unless the unicode-math package has been loaded, load the amssymb package with symbols to be used for tickboxes.

```

12893 \@ifpackageloaded{unicode-math}{
12894     \markdownSetup{rendererPrototypes={
12895         untickedBox = {\mdlgwhtsquare},
12896     }}
12897 }{
12898     \RequirePackage{amssymb}
12899     \markdownSetup{rendererPrototypes={
12900         untickedBox = {\square},
12901     }}
12902 }
12903 \RequirePackage{csvsimple}
12904 \RequirePackage{fancyvrb}
12905 \RequirePackage{graphicx}
12906 \markdownSetup{rendererPrototypes={

```

```

12907  hardLineBreak = {\},
12908  leftBrace = {\textbraceleft},
12909  rightBrace = {\textbraceright},
12910  dollarSign = {\textdollar},
12911  underscore = {\textunderscore},
12912  circumflex = {\textasciicircum},
12913  backslash = {\textbackslash},
12914  tilde = {\textasciitilde},
12915  pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by T_EX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,³⁴ we can reliably detect math mode inside the renderer.

Here, we will redefine the code span renderer prototype to typeset upright text in math formulae and typewriter text outside math formulae.

```

12916  codeSpan = {%
12917    \ifmmode
12918      \text{#1}%
12919    \else
12920      \texttt{#1}%
12921    \fi
12922  }}
12923  \ExplSyntaxOn
12924  \markdownSetup{
12925    rendererPrototypes = {
12926      contentBlock = {
12927        \str_case:nnF
12928          { #1 }
12929          {
12930            { csv }
12931            {
12932              \begin{table}
12933                \begin{center}
12934                  \csvautotabular{#3}
12935                \end{center}
12936                \tl_if_empty:nF
12937                  { #4 }
12938                  { \caption{#4} }
12939              \end{table}
12940            }
12941            { tex } { \markdownEscape{#3} }
12942          }

```

³⁴This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```

12943     { \markdownInput{#3} }
12944   },
12945 },
12946 }
12947 \ExplSyntaxOff
12948 \markdownSetup{rendererPrototypes={
12949   image = {%
12950     \begin{figure}%
12951       \begin{center}%
12952         \includegraphics[alt={#1}]{#3}%
12953       \end{center}%
12954       \ifx\empty#4\empty\else
12955         \caption{#4}%
12956       \fi
12957     \end{figure}},
12958   ulBegin = {\begin{itemize}},
12959   ulEnd = {\end{itemize}},
12960   olBegin = {\begin{enumerate}},
12961   olItem = {\item{}},
12962   olItemWithNumber = {\item[#1.]},
12963   olEnd = {\end{enumerate}},
12964   dlBegin = {\begin{description}},
12965   dlItem = {\item[#1]},
12966   dlEnd = {\end{description}},
12967   emphasis = {\emph{#1}},
12968   tickedBox = {\$\boxtimes$},
12969   halfTickedBox = {\$\boxdot$}}

```

If HTML identifiers appear after a heading, we make them produce `\label` macros.

```

12970 \ExplSyntaxOn
12971 \seq_new:N
12972 \l_@@_header_identifiers_seq
12973 \markdownSetup
12974 {
12975   rendererPrototypes = {
12976     headerAttributeContextBegin = {
12977       \markdownSetup
12978       {
12979         rendererPrototypes = {
12980           attributeIdentifier = {
12981             \seq_put_right:Nn
12982             \l_@@_header_identifiers_seq
12983             { #1 }
12984           },
12985         },
12986       }
12987     },
12988     headerAttributeContextEnd = {

```

```

12989     \seq_map_inline:Nn
12990     \l_@@_header_identifiers_seq
12991     { \label { ##1 } }
12992     \seq_clear:N
12993     \l_@@_header_identifiers_seq
12994   },
12995 },
12996 }

```

If the `unnumbered` HTML class (or the `{-}` shorthand) appears after a heading the heading and all its subheadings will be unnumbered.

```

12997 \bool_new:N
12998 \l_@@_header_unnumbered_bool
12999 \markdownSetup
13000 {
13001   rendererPrototypes = {
13002     headerAttributeContextBegin += {
13003       \markdownSetup
13004       {
13005         rendererPrototypes = {
13006           attributeClassName = {
13007             \bool_if:nT
13008             {
13009               \str_if_eq_p:nn
13010               { ##1 }
13011               { unnumbered } &&
13012               ! \l_@@_header_unnumbered_bool
13013             }
13014             {
13015               \group_begin:
13016               \bool_set_true:N
13017               \l_@@_header_unnumbered_bool
13018               \c@secnumdepth = 0
13019               \markdownSetup
13020               {
13021                 rendererPrototypes = {
13022                   sectionBegin = {
13023                     \group_begin:
13024                   },
13025                   sectionEnd = {
13026                     \group_end:
13027                   },
13028                 },
13029               }
13030             }
13031           },
13032         },

```



```

13033     }
13034   },
13035 },
13036 }
13037 \ExplSyntaxOff
13038 \markdownSetup{rendererPrototypes={
13039   superscript = {\textsuperscript{#1}},
13040   subscript = {\textsubscript{#1}},
13041   blockQuoteBegin = {\begin{quotation}},
13042   blockQuoteEnd = {\end{quotation}},
13043   inputVerbatim = {\VerbatimInput{#1}},
13044   thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
13045   note = {\footnote{#1}}}}

```

3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13046 \RequirePackage{ltxcmds}
13047 \ExplSyntaxOn
13048 \cs_gset:Npn
13049   \markdownRendererInputFencedCodePrototype#1#2#3
13050   {
13051     \tl_if_empty:nTF
13052       { #2 }
13053     { \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.

```

13054     {
13055       \regex_extract_once:nnN
13056         { \w* }
13057         { #2 }
13058         \l_tmpa_seq
13059       \seq_pop_left:NN
13060         \l_tmpa_seq
13061         \l_tmpa_tl

```

When the minted package is loaded, use it for syntax highlighting.

```

13062     \ltx@ifpackageloaded
13063       { minted }
13064     {
13065       \catcode`\#=6\relax
13066       \exp_args:NV
13067         \inputminted
13068         \l_tmpa_tl
13069         { #1 }
13070       \catcode`\#=12\relax
13071     }

```

```

13072      {
    When the listings package is loaded, use it for syntax highlighting.
13073      \ltx@ifpackageloaded
13074      { listings }
13075      { \lstinputlisting[language=\l_tmpa_tl]{#1} }

    When neither the listings package nor the minted package is loaded, act as though
    no infostring were given.
13076      { \markdownRendererInputFencedCode{#1}{}{} }
13077      }
13078  }
13079 }
13080 \ExplSyntaxOff

    Support the nesting of strong emphasis.
13081 \ExplSyntaxOn
13082 \def\markdownLATEXStrongEmphasis#1{%
13083   \str_if_in:NnTF
13084     \f@series
13085     { b }
13086     { \textnormal{#1} }
13087     { \textbf{#1} }
13088 }
13089 \ExplSyntaxOff
13090 \markdownSetup{rendererPrototypes={strongEmphasis={%
13091   \protect\markdownLATEXStrongEmphasis{#1}}}}

    Support LATEX document classes that do not provide chapters.
13092 \@ifundefined{chapter}{%
13093   \markdownSetup{rendererPrototypes = {
13094     headingOne = {\section{#1}},
13095     headingTwo = {\subsection{#1}},
13096     headingThree = {\subsubsection{#1}},
13097     headingFour = {\paragraph{#1}},
13098     headingFive = {\subparagraph{#1}}}}
13099 }{%
13100   \markdownSetup{rendererPrototypes = {
13101     headingOne = {\chapter{#1}},
13102     headingTwo = {\section{#1}},
13103     headingThree = {\subsection{#1}},
13104     headingFour = {\subsubsection{#1}},
13105     headingFive = {\paragraph{#1}},
13106     headingSix = {\subparagraph{#1}}}}
13107 }%

```

3.3.4.2 Tickboxes

If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

13108 \markdownSetup{
13109   rendererPrototypes = {
13110     ulItem = {%
13111       \futurelet\markdownLaTeXCheckbox\markdownLaTeXULItem
13112     },
13113   },
13114 }
13115 \def\markdownLaTeXULItem{%
13116   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
13117     \item[\markdownLaTeXCheckbox]%
13118     \expandafter\@gobble
13119   \else
13120     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
13121       \item[\markdownLaTeXCheckbox]%
13122       \expandafter\expandafter\expandafter\@gobble
13123     \else
13124       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
13125         \item[\markdownLaTeXCheckbox]%
13126         \expandafter\expandafter\expandafter\expandafter
13127         \expandafter\expandafter\expandafter\@gobble
13128       \else
13129         \item{}%
13130       \fi
13131     \fi
13132   \fi
13133 }

```

3.3.4.3 HTML elements

If the `html` option is enabled and we are using $\text{T}_{\text{E}}\text{X}_{4\text{ht}}$ ³⁵, we will pass HTML elements to the output HTML document unchanged.

```

13134 \@ifundefined{HCode}{}{
13135   \markdownSetup{
13136     rendererPrototypes = {
13137       inlineHtmlTag = {%
13138         \ifvmode
13139           \IgnorePar
13140           \EndP
13141         \fi
13142         \HCode{#1}%
13143       },
13144       inputBlockHtmlElement = {%
13145         \ifvmode

```

³⁵See <https://tug.org/tex4ht/>.

```

13146         \IgnorePar
13147         \fi
13148         \EndP
13149         \special{t4ht* <#1}%
13150         \par
13151         \ShowPar
13152     },
13153 },
13154 }
13155 }

```

3.3.4.4 Citations

Here is a basic implementation for citations that uses the L^AT_EX `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibL^AT_EX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

13156 \newcount\markdownLaTeXCitationsCounter
13157
13158 % Basic implementation
13159 \RequirePackage{gobble}
13160 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
13161     \advance\markdownLaTeXCitationsCounter by 1\relax
13162     \ifx\relax#4\relax
13163         \ifx\relax#5\relax
13164             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13165                 \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
13166                 \expandafter\expandafter\expandafter
13167                 \expandafter\expandafter\expandafter\expandafter
13168                 \@gobblethree
13169             \fi
13170         \else% Before a postnote (#5), dump the accumulator
13171             \ifx\relax#1\relax\else
13172                 \cite{#1}%
13173             \fi
13174             \cite[#5]{#6}%
13175             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13176             \else
13177                 \expandafter\expandafter\expandafter
13178                 \expandafter\expandafter\expandafter\expandafter
13179                 \expandafter\expandafter\expandafter
13180                 \expandafter\expandafter\expandafter\expandafter
13181                 \markdownLaTeXBasicCitations
13182             \fi
13183             \expandafter\expandafter\expandafter
13184             \expandafter\expandafter\expandafter\expandafter{%
13185             \expandafter\expandafter\expandafter

```

```

13186     \expandafter\expandafter\expandafter\expandafter}%
13187     \expandafter\expandafter\expandafter
13188     \expandafter\expandafter\expandafter\expandafter{%
13189     \expandafter\expandafter\expandafter
13190     \expandafter\expandafter\expandafter\expandafter}%
13191     \expandafter\expandafter\expandafter
13192     \@gobblethree
13193     \fi
13194 \else% Before a prenote (#4), dump the accumulator
13195     \ifx\relax#1\relax\else
13196         \cite{#1}%
13197     \fi
13198     \ifnum\markdownLaTeXCitationsCounter>1\relax
13199         \space % Insert a space before the prenote in later citations
13200     \fi
13201     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
13202     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13203     \else
13204         \expandafter\expandafter\expandafter
13205         \expandafter\expandafter\expandafter\expandafter
13206         \markdownLaTeXBasicCitations
13207     \fi
13208     \expandafter\expandafter\expandafter{%
13209     \expandafter\expandafter\expandafter}%
13210     \expandafter\expandafter\expandafter{%
13211     \expandafter\expandafter\expandafter}%
13212     \expandafter
13213     \@gobblethree
13214     \fi\markdownLaTeXBasicCitations{#1#2#6},}
13215 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
13216
13217 % Natbib implementation
13218 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
13219     \advance\markdownLaTeXCitationsCounter by 1\relax
13220     \ifx\relax#3\relax
13221         \ifx\relax#4\relax
13222             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13223                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
13224                 \expandafter\expandafter\expandafter
13225                 \expandafter\expandafter\expandafter\expandafter
13226                 \@gobbletwo
13227             \fi
13228         \else% Before a postnote (#4), dump the accumulator
13229             \ifx\relax#1\relax\else
13230                 \citep{#1}%
13231             \fi
13232             \citep[] [#4] {#5}%

```

```

13233 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13234 \else
13235 \expandafter\expandafter\expandafter
13236 \expandafter\expandafter\expandafter\expandafter
13237 \expandafter\expandafter\expandafter
13238 \expandafter\expandafter\expandafter\expandafter
13239 \markdownLaTeXNatbibCitations
13240 \fi
13241 \expandafter\expandafter\expandafter
13242 \expandafter\expandafter\expandafter\expandafter{%
13243 \expandafter\expandafter\expandafter
13244 \expandafter\expandafter\expandafter\expandafter}%
13245 \expandafter\expandafter\expandafter
13246 \@gobbletwo
13247 \fi
13248 \else% Before a prenote (#3), dump the accumulator
13249 \ifx\relax#1\relax\relax\else
13250 \citep{#1}%
13251 \fi
13252 \citep[#3][#4]{#5}%
13253 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13254 \else
13255 \expandafter\expandafter\expandafter
13256 \expandafter\expandafter\expandafter\expandafter
13257 \markdownLaTeXNatbibCitations
13258 \fi
13259 \expandafter\expandafter\expandafter{%
13260 \expandafter\expandafter\expandafter}%
13261 \expandafter
13262 \@gobbletwo
13263 \fi\markdownLaTeXNatbibCitations{#1,#5}}
13264 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
13265 \advance\markdownLaTeXCitationsCounter by 1\relax
13266 \ifx\relax#3\relax
13267 \ifx\relax#4\relax
13268 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13269 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
13270 \expandafter\expandafter\expandafter
13271 \expandafter\expandafter\expandafter\expandafter
13272 \@gobbletwo
13273 \fi
13274 \else% After a prenote or a postnote, dump the accumulator
13275 \ifx\relax#1\relax\else
13276 \citet{#1}%
13277 \fi
13278 , \citet[#3][#4]{#5}%
13279 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax

```

```

13280     ,
13281     \else
13282         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
13283     ,
13284     \fi
13285     \fi
13286     \expandafter\expandafter\expandafter
13287     \expandafter\expandafter\expandafter\expandafter
13288     \markdownLaTeXNatbibTextCitations
13289     \expandafter\expandafter\expandafter
13290     \expandafter\expandafter\expandafter\expandafter{%
13291     \expandafter\expandafter\expandafter
13292     \expandafter\expandafter\expandafter\expandafter}%
13293     \expandafter\expandafter\expandafter
13294     \@gobbletwo
13295     \fi
13296 \else% After a prenote or a postnote, dump the accumulator
13297     \ifx\relax#1\relax\relax\else
13298         \citet{#1}%
13299     \fi
13300     , \citet[#3][#4]{#5}%
13301     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
13302     ,
13303     \else
13304         \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
13305     ,
13306     \fi
13307     \fi
13308     \expandafter\expandafter\expandafter
13309     \markdownLaTeXNatbibTextCitations
13310     \expandafter\expandafter\expandafter{%
13311     \expandafter\expandafter\expandafter}%
13312     \expandafter
13313     \@gobbletwo
13314     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
13315
13316 % BibLaTeX implementation
13317 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
13318     \advance\markdownLaTeXCitationsCounter by 1\relax
13319     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13320         \autocites#1[#3][#4]{#5}%
13321         \expandafter\@gobbletwo
13322     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
13323 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
13324     \advance\markdownLaTeXCitationsCounter by 1\relax
13325     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
13326         \textcites#1[#3][#4]{#5}%

```

```

13327     \expandafter\@gobbletwo
13328     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}
13329
13330 \markdownSetup{rendererPrototypes = {
13331   cite = {%
13332     \markdownLaTeXCitationsCounter=1%
13333     \def\markdownLaTeXCitationsTotal{#1}%
13334     \@ifundefined{autocites}{%
13335       \@ifundefined{citep}{%
13336         \expandafter\expandafter\expandafter
13337         \markdownLaTeXBasicCitations
13338         \expandafter\expandafter\expandafter{%
13339           \expandafter\expandafter\expandafter}%
13340         \expandafter\expandafter\expandafter{%
13341           \expandafter\expandafter\expandafter}%
13342         }{%
13343           \expandafter\expandafter\expandafter
13344           \markdownLaTeXNatbibCitations
13345           \expandafter\expandafter\expandafter{%
13346             \expandafter\expandafter\expandafter}%
13347         }%
13348       }{%
13349         \expandafter\expandafter\expandafter
13350         \markdownLaTeXBibLaTeXCitations
13351         \expandafter{\expandafter}%
13352       }},
13353   textCite = {%
13354     \markdownLaTeXCitationsCounter=1%
13355     \def\markdownLaTeXCitationsTotal{#1}%
13356     \@ifundefined{autocites}{%
13357       \@ifundefined{citep}{%
13358         \expandafter\expandafter\expandafter
13359         \markdownLaTeXBasicTextCitations
13360         \expandafter\expandafter\expandafter{%
13361           \expandafter\expandafter\expandafter}%
13362         \expandafter\expandafter\expandafter{%
13363           \expandafter\expandafter\expandafter}%
13364         }{%
13365           \expandafter\expandafter\expandafter
13366           \markdownLaTeXNatbibTextCitations
13367           \expandafter\expandafter\expandafter{%
13368             \expandafter\expandafter\expandafter}%
13369         }%
13370       }{%
13371         \expandafter\expandafter\expandafter
13372         \markdownLaTeXBibLaTeXTextCitations
13373         \expandafter{\expandafter}%

```



```
13374     }}}
```

3.3.4.5 Links

Here is an implementation for hypertext links and relative references.

```
13375 \RequirePackage{url}
13376 \RequirePackage{expl3}
13377 \ExplSyntaxOn
13378 \def\markdownRendererLinkPrototype#1#2#3#4{
13379   \tl_set:Nn \l_tmpa_tl { #1 }
13380   \tl_set:Nn \l_tmpb_tl { #2 }
13381   \bool_set:Nn
13382     \l_tmpa_bool
13383     {
13384       \tl_if_eq_p:NN
13385         \l_tmpa_tl
13386         \l_tmpb_tl
13387     }
13388   \tl_set:Nn \l_tmpa_tl { #4 }
13389   \bool_set:Nn
13390     \l_tmpb_bool
13391     {
13392       \tl_if_empty_p:N
13393         \l_tmpa_tl
13394     }

```

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.

```
13395   \bool_if:nTF
13396     {
13397       \l_tmpa_bool && \l_tmpb_bool
13398     }
13399     {
13400       \markdownLaTeXRendererAutolink { #2 } { #3 }
13401     }{
13402       \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
13403     }
13404 }
13405 \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.

```
13406   \tl_set:Nn
13407     \l_tmpa_tl
13408     { #2 }
13409   \tl_trim_spaces:N
13410     \l_tmpa_tl

```

```

13411 \tl_set:Nx
13412   \l_tmpb_tl
13413   {
13414     \tl_range:Nnn
13415       \l_tmpa_tl
13416       { 1 }
13417       { 1 }
13418   }
13419 \str_if_eq:NNTF
13420   \l_tmpb_tl
13421   \c_hash_str
13422   {
13423     \tl_set:Nx
13424       \l_tmpb_tl
13425       {
13426         \tl_range:Nnn
13427           \l_tmpa_tl
13428           { 2 }
13429           { -1 }
13430       }
13431     \exp_args:NV
13432     \ref
13433     \l_tmpb_tl
13434   }{
13435     \url { #2 }
13436   }
13437 }
13438 \ExplSyntaxOff
13439 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
13440   #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

3.3.4.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

13441 \newcount\markdownLaTeXRowCount
13442 \newcount\markdownLaTeXRowTotal
13443 \newcount\markdownLaTeXColumnCounter
13444 \newcount\markdownLaTeXColumnTotal
13445 \newtoks\markdownLaTeXTable
13446 \newtoks\markdownLaTeXTableAlignment
13447 \newtoks\markdownLaTeXTableEnd
13448 \AtBeginDocument{%
13449   \@ifpackageloaded{booktabs}{%
13450     \def\markdownLaTeXTopRule{\toprule}%
13451     \def\markdownLaTeXMidRule{\midrule}%
13452     \def\markdownLaTeXBottomRule{\bottomrule}%

```

```

13453 }{%
13454   \def\markdownLaTeXTopRule{\hline}%
13455   \def\markdownLaTeXMidRule{\hline}%
13456   \def\markdownLaTeXBottomRule{\hline}%
13457 }%
13458 }
13459 \markdownSetup{rendererPrototypes={
13460   table = {%
13461     \markdownLaTeXTable={}%
13462     \markdownLaTeXTableAlignment={}%
13463     \markdownLaTeXTableEnd={%
13464       \markdownLaTeXBottomRule
13465       \end{tabular}}%
13466     \ifx\empty#1\empty\else
13467       \addto@hook\markdownLaTeXTable{%
13468         \begin{table}
13469         \centering}%
13470       \addto@hook\markdownLaTeXTableEnd{%
13471         \caption{#1}
13472         \end{table}}%
13473     \fi
13474     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
13475     \markdownLaTeXRowCount=0%
13476     \markdownLaTeXRowTotal=#2%
13477     \markdownLaTeXColumnTotal=#3%
13478     \markdownLaTeXRenderTableRow
13479   }
13480 }}
13481 \def\markdownLaTeXRenderTableRow#1{%
13482   \markdownLaTeXColumnCounter=0%
13483   \ifnum\markdownLaTeXRowCount=0\relax
13484     \markdownLaTeXReadAlignments#1%
13485     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
13486       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
13487         \the\markdownLaTeXTableAlignment}}%
13488     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
13489   \else
13490     \markdownLaTeXRenderTableCell#1%
13491   \fi
13492   \ifnum\markdownLaTeXRowCount=1\relax
13493     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
13494   \fi
13495   \advance\markdownLaTeXRowCount by 1\relax
13496   \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
13497     \the\markdownLaTeXTable
13498     \the\markdownLaTeXTableEnd
13499     \expandafter\@gobble

```

```

13500 \fi\markdownLaTeXRenderTableRow}
13501 \def\markdownLaTeXReadAlignments#1{%
13502 \advance\markdownLaTeXColumnCounter by 1\relax
13503 \if#1d%
13504 \addto@hook\markdownLaTeXTableAlignment{1}%
13505 \else
13506 \addto@hook\markdownLaTeXTableAlignment{#1}%
13507 \fi
13508 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
13509 \expandafter\@gobble
13510 \fi\markdownLaTeXReadAlignments}
13511 \def\markdownLaTeXRenderTableCell#1{%
13512 \advance\markdownLaTeXColumnCounter by 1\relax
13513 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
13514 \addto@hook\markdownLaTeXTable{#1&}%
13515 \else
13516 \addto@hook\markdownLaTeXTable{#1\}%
13517 \expandafter\@gobble
13518 \fi\markdownLaTeXRenderTableCell}

```

3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the `verse` package is loaded, then it is used to produce the verses.

```

13519
13520 \markdownIfOption{lineBlocks}{%
13521 \RequirePackage{verse}
13522 \markdownSetup{rendererPrototypes={
13523 lineBlockBegin = {%
13524 \begingroup
13525 \def\markdownRendererHardLineBreak{\}%
13526 \begin{verse}%
13527 },
13528 lineBlockEnd = {%
13529 \end{verse}%
13530 \endgroup
13531 },
13532 }}
13533 }{}
13534

```

3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```

13535 \ExplSyntaxOn

```

```

13536 \keys_define:nn
13537   { markdown/jekyllData }
13538   {
13539     author .code:n = { \author{#1} },
13540     date   .code:n = { \date{#1}   },
13541     title  .code:n = { \title{#1}  },
13542   }

```

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.

```

13543 \markdownSetup{
13544   rendererPrototypes = {
13545     jekyllDataEnd = {
13546       \AddToHook{begindocument/end}{\maketitle}
13547     },
13548   },
13549 }
13550 \ExplSyntaxOff

```

3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

13551 \markdownIfOption{strikeThrough}{%
13552   \RequirePackage{soulutf8}%
13553   \markdownSetup{
13554     rendererPrototypes = {
13555       strikeThrough = {%
13556         \st{#1}%
13557       },
13558     }
13559   }
13560 }{}

```

3.3.4.10 Marked Text

If the `mark` option is enabled, we will load the `soulutf8` package and use it to implement marked text.

```

13561 \markdownIfOption{mark}{%
13562   \RequirePackage{soulutf8}%
13563   \markdownSetup{
13564     rendererPrototypes = {
13565       mark = {%
13566         \hl{#1}%
13567       },

```

```

13568     }
13569   }
13570 }{}

```

3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the `graphicx` package. Furthermore, in image attribute contexts, we will make attributes in the form $\langle key \rangle = \langle value \rangle$ set the corresponding keys of the `graphicx` package to the corresponding values.

```

13571 \ExplSyntaxOn
13572 \@@_if_option:nT
13573   { linkAttributes }
13574   {
13575     \RequirePackage{graphicx}
13576     \markdownSetup{
13577       rendererPrototypes = {
13578         imageAttributeContextBegin = {
13579           \group_begin:
13580           \markdownSetup{
13581             rendererPrototypes = {
13582               attributeKeyValue = {
13583                 \setkeys
13584                 { Gin }
13585                 { { ##1 } = { ##2 } }
13586               },
13587             },
13588           }
13589         },
13590         imageAttributeContextEnd = {
13591           \group_end:
13592         },
13593       },
13594     }
13595   }
13596 \ExplSyntaxOff

```

3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```

13597 \ExplSyntaxOn
13598 \cs_gset:Npn
13599   \markdownRendererInputRawInlinePrototype#1#2
13600   {
13601     \str_case:nnF
13602     { #2 }

```

```

13603     {
13604     { latex }
13605     {
13606         \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13607         { #1 }
13608         { tex }
13609     }
13610     }
13611     {
13612     \@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13613     { #1 }
13614     { #2 }
13615     }
13616     }
13617 \cs_gset:Npn
13618 \markdownRendererInputRawBlockPrototype#1#2
13619 {
13620     \str_case:nnF
13621     { #2 }
13622     {
13623     { latex }
13624     {
13625         \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13626         { #1 }
13627         { tex }
13628     }
13629     }
13630     {
13631     \@_plain_tex_default_input_raw_block_renderer_prototype:nn
13632     { #1 }
13633     { #2 }
13634     }
13635     }
13636 \ExplSyntaxOff
13637 \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.

```

13638 \newcommand\markdownMakeOther{%
13639     \count0=128\relax
13640     \loop
13641     \catcode\count0=11\relax

```

```

13642     \advance\count0 by 1\relax
13643     \ifnum\count0<256\repeat}%

```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain T_EX implementation. As a consequence, we can directly reuse the existing plain T_EX implementation after supplying the missing plain T_EX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L^AT_EX package.

```

13644 \def\markdownMakeOther{%
13645     \count0=128\relax
13646     \loop
13647         \catcode\count0=11\relax
13648         \advance\count0 by 1\relax
13649     \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_EXt.

```

13650     \catcode`|=12}%

```

3.4.1 Typesetting Markdown

The `\inputmarkdown` macro is defined to accept an optional argument with options recognized by the ConT_EXt interface (see Section 2.4.2).

```

13651 \long\def\inputmarkdown{%
13652     \dosingleempty
13653     \doinputmarkdown}%
13654 \long\def\doinputmarkdown[#1]#2{%
13655     \begingroup
13656         \iffirstargument
13657             \setupmarkdown[#1]%
13658         \fi
13659     \markdownInput{#2}%
13660 \endgroup}%

```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth’s T_EX, trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because “these spaces are hard to see in an editor”. At the moment, there is no option to

suppress this behavior in (Lua)TeX, but ConTeXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTeXt MkIV and therefore to insert hard line breaks into markdown text.

```

13661 \startluacode
13662   document.markdown_buffering = false
13663   local function preserve_trailing_spaces(line)
13664     if document.markdown_buffering then
13665       line = line:gsub("[ \t][ \t]$", "\t\t")
13666     end
13667     return line
13668   end
13669   resolvers.installinputlinehandler(preserve_trailing_spaces)
13670 \stopluacode
13671 \begingroup
13672   \catcode\|=0%
13673   \catcode\|=12%
13674   |gdef|startmarkdown{%
13675     |ctxlua{document.markdown_buffering = true}%
13676     |markdownReadAndConvert{\stopmarkdown}%
13677     {|stopmarkdown}}%
13678   |gdef|stopmarkdown{%
13679     |ctxlua{document.markdown_buffering = false}%
13680     |markdownEnd}%
13681 |endgroup

```

3.4.2 Themes

This section overrides the plain TeX implementation of the theme-loading mechanism from Section 3.2.2. Furthermore, this section also implements the built-in ConTeXt themes provided with the Markdown package.

```

13682 \ExplSyntaxOn
13683 \cs_gset:Nn
13684   \@@_load_theme:nn
13685   {

```

Determine whether a file named `t-markdowntheme<munged theme name>.tex` exists. If it does, load it. Otherwise, try loading a plain TeX theme instead.

```

13686   \file_if_exist:nTF
13687     { t - markdown theme #2.tex }
13688     {
13689       \msg_info:nnn
13690         { markdown }
13691         { loading-context-theme }
13692         { #1 }
13693       \usemodule
13694         [ t ]

```

```

13695         [ markdown theme #2 ]
13696     }
13697     {
13698         \@@_plain_tex_load_theme:nn
13699         { #1 }
13700         { #2 }
13701     }
13702 }
13703 \msg_new:nnn
13704 { markdown }
13705 { loading-context-theme }
13706 { Loading~ConTeXt~Markdown~theme~#1 }
13707 \ExplSyntaxOff

```

The `witiko/markdown/defaults` ConTeXt theme provides default definitions for token renderer prototypes. First, the ConTeXt theme loads the plain TeX theme with the default definitions for plain TeX:

```
13708 \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.

```

13709 \markdownIfOption{plain}{\iffalse}{\iftrue}
13710 \def\markdownRendererHardLineBreakPrototype{\blank}%
13711 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
13712 \def\markdownRendererRightBracePrototype{\textbraceright}%
13713 \def\markdownRendererDollarSignPrototype{\textdollar}%
13714 \def\markdownRendererPercentSignPrototype{\percent}%
13715 \def\markdownRendererUnderscorePrototype{\textunderscore}%
13716 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
13717 \def\markdownRendererBackslashPrototype{\textbackslash}%
13718 \def\markdownRendererTildePrototype{\textasciitilde}%
13719 \def\markdownRendererPipePrototype{\char`|}%
13720 \def\markdownRendererLinkPrototype#1#2#3#4{%
13721   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
13722   \fi\texttt<\hyphenatedurl{#3}>}}%
13723 \usemodule[database]
13724 \defineseparatedlist
13725 [MarkdownConTeXtCSV]
13726 [separator={,},
13727  before=\bTABLE,after=\eTABLE,
13728  first=\bTR,last=\eTR,
13729  left=\bTD,right=\eTD]

```

```

13730 \def\markdownConTeXtCSV{csv}
13731 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
13732   \def\markdownConTeXtCSV@arg{#1}%
13733   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
13734     \placetable [] [tab:#1]{#4}{%
13735       \processeparatedfile [MarkdownConTeXtCSV] [#3]}%
13736   \else
13737     \markdownInput{#3}%
13738   \fi}%
13739 \def\markdownRendererImagePrototype#1#2#3#4{%
13740   \placefigure [] []{#4}{\externalfigure [#3]}%
13741 \def\markdownRendererUlBeginPrototype{\startitemize}%
13742 \def\markdownRendererUlBeginTightPrototype{\startitemize [packed]}%
13743 \def\markdownRendererUlItemPrototype{\item}%
13744 \def\markdownRendererUlEndPrototype{\stopitemize}%
13745 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
13746 \def\markdownRendererOlBeginPrototype{\startitemize [n]}%
13747 \def\markdownRendererOlBeginTightPrototype{\startitemize [packed,n]}%
13748 \def\markdownRendererOlItemPrototype{\item}%
13749 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
13750 \def\markdownRendererOlEndPrototype{\stopitemize}%
13751 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
13752 \definedescription
13753   [MarkdownConTeXtDlItemPrototype]
13754   [location=hanging,
13755    margin=standard,
13756    headstyle=bold]%
13757 \definestartstop
13758   [MarkdownConTeXtDlPrototype]
13759   [before=\blank,
13760    after=\blank]%
13761 \definestartstop
13762   [MarkdownConTeXtDlTightPrototype]
13763   [before=\blank\startpacked,
13764    after=\stoppacked\blank]%
13765 \def\markdownRendererDlBeginPrototype{%
13766   \startMarkdownConTeXtDlPrototype}%
13767 \def\markdownRendererDlBeginTightPrototype{%
13768   \startMarkdownConTeXtDlTightPrototype}%
13769 \def\markdownRendererDlItemPrototype#1{%
13770   \startMarkdownConTeXtDlItemPrototype{#1}}%
13771 \def\markdownRendererDlItemEndPrototype{%
13772   \stopMarkdownConTeXtDlItemPrototype}%
13773 \def\markdownRendererDlEndPrototype{%
13774   \stopMarkdownConTeXtDlPrototype}%
13775 \def\markdownRendererDlEndTightPrototype{%
13776   \stopMarkdownConTeXtDlTightPrototype}%

```

```

13777 \def\markdownRendererEmphasisPrototype#1{\em#1}%
13778 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
13779 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
13780 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
13781 \def\markdownRendererLineBlockBeginPrototype{%
13782   \begingroup
13783     \def\markdownRendererHardLineBreak{
13784       }%
13785     \startlines
13786   }%
13787 \def\markdownRendererLineBlockEndPrototype{%
13788   \stoptlines
13789   \endgroup
13790 }%
13791 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%

```

3.4.3.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```

13792 \ExplSyntaxOn
13793 \cs_gset:Npn
13794   \markdownRendererInputFencedCodePrototype#1#2#3
13795   {
13796     \tl_if_empty:nTF
13797       { #2 }
13798       { \markdownRendererInputVerbatim{#1} }

```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTeXt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```

\definetyping [latex]
\setuptyping [latex] [option=TEX]

\starttext
  \startmarkdown
  ~~~ latex
  \documentclass{article}
  \begin{document}
  Hello world!
  \end{document}
  ~~~
  \stopmarkdown
\stoptext

```

```

13799     {
13800     \regex_extract_once:nnN
13801     { \w* }
13802     { #2 }
13803     \l_tmpa_seq
13804     \seq_pop_left:NN
13805     \l_tmpa_seq
13806     \l_tmpa_tl
13807     \typefile[\l_tmpa_tl] []{#1}
13808     }
13809   }
13810 \ExplSyntaxOff
13811 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
13812 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
13813 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
13814 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
13815 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
13816 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
13817 \def\markdownRendererThematicBreakPrototype{%
13818   \blackrule[height=1pt, width=\hsize]}%
13819 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
13820 \def\markdownRendererTickedBoxPrototype{\boxtimes$}
13821 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
13822 \def\markdownRendererUntickedBoxPrototype{\square$}
13823 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
13824 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
13825 \def\markdownRendererSubscriptPrototype#1{\low{#1}}
13826 \def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%

```

3.4.3.2 Tables

There is a basic implementation of tables.

```

13827 \newcount\markdownConTeXtRowCounter
13828 \newcount\markdownConTeXtRowTotal
13829 \newcount\markdownConTeXtColumnCounter
13830 \newcount\markdownConTeXtColumnTotal
13831 \newtoks\markdownConTeXtTable
13832 \newtoks\markdownConTeXtTableFloat
13833 \def\markdownRendererTablePrototype#1#2#3{%
13834   \markdownConTeXtTable={}%
13835   \ifx\empty#1\empty
13836     \markdownConTeXtTableFloat={%
13837       \the\markdownConTeXtTable}%
13838   \else
13839     \markdownConTeXtTableFloat={%
13840       \placetable{#1}{\the\markdownConTeXtTable}}%
13841   \fi

```

```

13842 \begingroup
13843 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13844 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
13845 \setupTABLE[r][1][topframe=on, bottomframe=on]
13846 \setupTABLE[r][#1][bottomframe=on]
13847 \markdownConTeXtRowCounter=0%
13848 \markdownConTeXtRowTotal=#2%
13849 \markdownConTeXtColumnTotal=#3%
13850 \markdownConTeXtRenderTableRow}
13851 \def\markdownConTeXtRenderTableRow#1{%
13852 \markdownConTeXtColumnCounter=0%
13853 \ifnum\markdownConTeXtRowCounter=0\relax
13854 \markdownConTeXtReadAlignments#1%
13855 \markdownConTeXtTable={\bTABLE}%
13856 \else
13857 \markdownConTeXtTable=\expandafter{%
13858 \the\markdownConTeXtTable\bTR}%
13859 \markdownConTeXtRenderTableCell#1%
13860 \markdownConTeXtTable=\expandafter{%
13861 \the\markdownConTeXtTable\eTR}%
13862 \fi
13863 \advance\markdownConTeXtRowCounter by 1\relax
13864 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
13865 \markdownConTeXtTable=\expandafter{%
13866 \the\markdownConTeXtTable\eTABLE}%
13867 \the\markdownConTeXtTableFloat
13868 \endgroup
13869 \expandafter\gobbleoneargument
13870 \fi\markdownConTeXtRenderTableRow}
13871 \def\markdownConTeXtReadAlignments#1{%
13872 \advance\markdownConTeXtColumnCounter by 1\relax
13873 \if#1d%
13874 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13875 \fi\if#1l%
13876 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
13877 \fi\if#1c%
13878 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
13879 \fi\if#1r%
13880 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
13881 \fi
13882 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13883 \expandafter\gobbleoneargument
13884 \fi\markdownConTeXtReadAlignments}
13885 \def\markdownConTeXtRenderTableCell#1{%
13886 \advance\markdownConTeXtColumnCounter by 1\relax
13887 \markdownConTeXtTable=\expandafter{%
13888 \the\markdownConTeXtTable\bTD#1\eTD}%

```

```

13889 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
13890 \expandafter\gobbleoneargument
13891 \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`.

```

13892 \ExplSyntaxOn
13893 \cs_gset:Npn
13894 \markdownRendererInputRawInlinePrototype#1#2
13895 {
13896   \str_case:nnF
13897     { #2 }
13898     {
13899       { latex }
13900       {
13901         \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13902         { #1 }
13903         { context }
13904       }
13905     }
13906     {
13907       \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
13908       { #1 }
13909       { #2 }
13910     }
13911 }
13912 \cs_gset:Npn
13913 \markdownRendererInputRawBlockPrototype#1#2
13914 {
13915   \str_case:nnF
13916     { #2 }
13917     {
13918       { context }
13919       {
13920         \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13921         { #1 }
13922         { tex }
13923       }
13924     }
13925     {
13926       \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
13927       { #1 }
13928       { #2 }
13929     }
13930 }

```

```

13931 \cs_gset_eq:NN
13932   \markdownRendererInputRawBlockPrototype
13933   \markdownRendererInputRawInlinePrototype
13934 \fi % Closes ` \markdownIfOption{plain}{\iffalse}{\iftrue}`
13935 \ExplSyntaxOff
13936 \stopmodule
13937 \protect

```

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

```

13938 \markdownIfOption{noDefaults}{}{
13939   \setupmarkdown[theme=witiko/markdown/defaults]
13940 }
13941 \stopmodule
13942 \protect

```

References

- [1] LuaTeX development team. *LuaTeX reference manual*. Version 1.10 (stable). July 23, 2021. URL: <https://www.pragma-ade.com/general/manuals/luatex.pdf> (visited on 09/30/2022).
- [2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: <https://www.muni.cz/en/research/projects/32984> (visited on 02/19/2018).
- [3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: <https://github.com/iainc/Markdown-Content-Blocks> (visited on 01/08/2018).
- [4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: <https://pandoc.org/> (visited on 10/05/2022).
- [5] Bonita Sharif and Jonathan I. Maletic. “An Eye Tracking Study on camelCase and under_score Identifier Styles.” In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: [10.1109/ICPC.2010.41](https://doi.org/10.1109/ICPC.2010.41).
- [6] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [7] Frank Mittelbach. *The doc and shortvrb Packages*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/doc.pdf> (visited on 02/19/2018).
- [8] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).

- [9] Vít Starý Novotný et al. *Convert control sequence with a variable number of undelimited parameters into a token list*. URL: <https://tex.stackexchange.com/q/716362/70941> (visited on 04/28/2024).
- [10] Geoffrey M. Poore. *The `minted` Package. Highlighted source code in \LaTeX* . July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [11] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [12] Johannes Braams et al. *The $\LaTeX_{2\epsilon}$ Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).
- [13] Donald Ervin Knuth. *T \TeX : The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 978-0-201-13437-7.
- [14] Victor Eijkhout. *T \TeX by Topic. A T \TeX nician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 978-0-201-56882-0.

Index

| | |
|---------------------------------------|-----------------------------------|
| <code>autoIdentifiers</code> | 20, 31, 73, 87 |
| <code>blankBeforeBlockquote</code> | 20 |
| <code>blankBeforeCodeFence</code> | 20 |
| <code>blankBeforeDivFence</code> | 21 |
| <code>blankBeforeHeading</code> | 21 |
| <code>blankBeforeList</code> | 21 |
| <code>bracketedSpans</code> | 22, 75 |
| <code>breakableBlockquotes</code> | 22 |
| <code>cacheDir</code> | 4, 16, 18, 55, 133, 285, 352, 368 |
| <code>citationNbsps</code> | 22 |
| <code>citations</code> | 23, 78 |
| <code>codeSpans</code> | 23 |
| <code>contentBlocks</code> | 19, 24 |
| <code>contentBlocksLanguageMap</code> | 19 |
| <code>contentLevel</code> | 24 |
| <code>debugExtensions</code> | 9, 19, 25, 281 |
| <code>debugExtensionsFileName</code> | 19, 25 |
| <code>defaultOptions</code> | 10, 49, 333 |
| <code>definitionLists</code> | 25, 82 |

| | |
|-----------------------------------|-------------------------------------|
| eagerCache | 16 |
| entities.char_entity | 191 |
| entities.dec_entity | 191 |
| entities.hex_entity | 191 |
| entities.hex_entity_with_x_char | 191 |
| expandtabs | 244 |
| expectJekyllData | 26 |
| extensions | 27, 140, 287 |
| extensions.bracketed_spans | 287 |
| extensions.citations | 288 |
| extensions.content_blocks | 292 |
| extensions.definition_lists | 295 |
| extensions.fancy_lists | 297 |
| extensions.fenced_code | 303 |
| extensions.fenced_divs | 307 |
| extensions.header_attributes | 311 |
| extensions.inline_code_attributes | 313 |
| extensions.jekyll_data | 329 |
| extensions.line_blocks | 313 |
| extensions.link_attributes | 315 |
| extensions.mark | 314 |
| extensions.notes | 316 |
| extensions.pipe_table | 318 |
| extensions.raw_inline | 323 |
| extensions.strike_through | 323 |
| extensions.subscripts | 324 |
| extensions.superscripts | 325 |
| extensions.tex_math | 325 |
| fancyLists | 28, 96–101, 369 |
| fencedCode | 29, 37, 79, 85, 102, 366 |
| fencedCodeAttributes | 29, 73 |
| fencedDiv | 86 |
| fencedDivs | 30, 39 |
| finalizeCache | 16, 19, 30, 31, 54, 55, 132, 286 |
| frozenCache | 19, 30, 55, 132, 135, 136, 367, 368 |
| frozenCacheCounter | 31, 286, 360 |
| frozenCacheFileName | 19, 30, 54, 286 |
| gfmAutoIdentifiers | 20, 31, 73, 87 |
| hashEnumerators | 31 |
| headerAttributes | 32, 39, 73, 87 |

| | |
|--|---|
| html | 32, 89, 90, 379 |
| hybrid | 33, 37, 43, 46, 57, 66, 103, 133, 196, 245, 360 |
| inlineCodeAttributes | 33, 74, 80 |
| inlineNotes | 33 |
| \input | 52 |
| \inputmarkdown | 137, 138, 392 |
| inputTempFileName | 55, 58, 354, 355, 357 |
| iterlines | 244 |
| jekyllData | 3, 26, 34, 110–113 |
| \l_file_search_path_seq | 359 |
| languages_json | 292, 292 |
| lineBlocks | 35, 92 |
| linkAttributes | 34, 74, 91, 94, 263, 390 |
| mark | 35, 94, 389 |
| \markdown | 131 |
| markdown | 130, 130, 131, 362 |
| markdown* | 130, 130–132, 362 |
| \markdown_jekyll_data_concatenate_address:NN | 348 |
| \markdown_jekyll_data_pop: | 348 |
| \markdown_jekyll_data_push:nN | 348 |
| \markdown_jekyll_data_push_address_segment:n | 347 |
| \markdown_jekyll_data_set_keyval:Nn | 349 |
| \markdown_jekyll_data_set_keyvals:nn | 349 |
| \markdown_jekyll_data_update_address_tls: | 348 |
| \markdownBegin | 50, 51, 52, 129, 131, 138 |
| \markdownCleanup | 353 |
| \markdownEnd | 50, 51, 52, 129, 131, 138 |
| \markdownError | 128, 128 |
| \markdownEscape | 50, 52, 361 |
| \markdownIfOption | 54 |
| \markdownIfSnippetExists | 68 |
| \markdownInfo | 128, 128 |
| \markdownInput | 50, 52, 130–132, 138, 359, 362 |
| \markdownInputFileStream | 353 |
| \markdownInputPlainTeX | 362 |
| \markdownLoadPlainTeXTheme | 133, 140, 342 |
| \markdownLuaExecute | 355, 359 |
| \markdownLuaOptions | 351, 353 |
| \markdownMakeOther | 129, 391, 392 |

| | |
|--|-------------------------|
| <code>\markdownOptionFinalizeCache</code> | 54 |
| <code>\markdownOptionFrozenCache</code> | 55 |
| <code>\markdownOptionHybrid</code> | 57 |
| <code>\markdownOptionInputTempFileName</code> | 55 |
| <code>\markdownOptionNoDefaults</code> | 56 |
| <code>\markdownOptionOutputDir</code> | 55, 55, 58 |
| <code>\markdownOptionPlain</code> | 56 |
| <code>\markdownOptionStripPercentSigns</code> | 57 |
| <code>\markdownOutputFileStream</code> | 353 |
| <code>\markdownPrepare</code> | 352 |
| <code>\markdownPrepareLuaOptions</code> | 351 |
| <code>\markdownReadAndConvert</code> | 129, 353, 362, 363, 392 |
| <code>\markdownReadAndConvertProcessLine</code> | 354, 355 |
| <code>\markdownReadAndConvertStripPercentSigns</code> | 354 |
| <code>\markdownReadAndConvertTab</code> | 353 |
| <code>\markdownRendererAttributeClassName</code> | 74 |
| <code>\markdownRendererAttributeIdentifier</code> | 74 |
| <code>\markdownRendererAttributeKeyValue</code> | 74 |
| <code>\markdownRendererBlockQuoteBegin</code> | 75 |
| <code>\markdownRendererBlockQuoteEnd</code> | 75 |
| <code>\markdownRendererBracketedSpanAttributeContextBegin</code> | 75 |
| <code>\markdownRendererBracketedSpanAttributeContextEnd</code> | 75 |
| <code>\markdownRendererCite</code> | 78, 78 |
| <code>\markdownRendererCodeSpan</code> | 79 |
| <code>\markdownRendererCodeSpanAttributeContextBegin</code> | 80 |
| <code>\markdownRendererCodeSpanAttributeContextEnd</code> | 80 |
| <code>\markdownRendererContentBlock</code> | 80, 81 |
| <code>\markdownRendererContentBlockCode</code> | 81 |
| <code>\markdownRendererContentBlockOnlineImage</code> | 81 |
| <code>\markdownRendererDisplayMath</code> | 108 |
| <code>\markdownRendererDlBegin</code> | 82 |
| <code>\markdownRendererDlBeginTight</code> | 82 |
| <code>\markdownRendererDlDefinitionBegin</code> | 83 |
| <code>\markdownRendererDlDefinitionEnd</code> | 83 |
| <code>\markdownRendererDlEnd</code> | 84 |
| <code>\markdownRendererDlEndTight</code> | 84 |
| <code>\markdownRendererDlItem</code> | 83 |
| <code>\markdownRendererDlItemEnd</code> | 83 |
| <code>\markdownRendererDocumentBegin</code> | 95 |
| <code>\markdownRendererDocumentEnd</code> | 95 |
| <code>\markdownRendererEllipsis</code> | 39, 84 |
| <code>\markdownRendererEmphasis</code> | 85, 116 |

| | |
|---|--------|
| <code>\markdownRendererFancyOlBegin</code> | 97, 97 |
| <code>\markdownRendererFancyOlBeginTight</code> | 97 |
| <code>\markdownRendererFancyOlEnd</code> | 100 |
| <code>\markdownRendererFancyOlEndTight</code> | 101 |
| <code>\markdownRendererFancyOlItem</code> | 99 |
| <code>\markdownRendererFancyOlItemEnd</code> | 99 |
| <code>\markdownRendererFancyOlItemWithNumber</code> | 99 |
| <code>\markdownRendererFencedCodeAttributeContextBegin</code> | 86 |
| <code>\markdownRendererFencedCodeAttributeContextEnd</code> | 86 |
| <code>\markdownRendererFencedDivAttributeContextBegin</code> | 86 |
| <code>\markdownRendererFencedDivAttributeContextEnd</code> | 86 |
| <code>\markdownRendererHalfTickedBox</code> | 109 |
| <code>\markdownRendererHardLineBreak</code> | 93 |
| <code>\markdownRendererHeaderAttributeContextBegin</code> | 87 |
| <code>\markdownRendererHeaderAttributeContextEnd</code> | 87 |
| <code>\markdownRendererHeadingFive</code> | 89 |
| <code>\markdownRendererHeadingFour</code> | 88 |
| <code>\markdownRendererHeadingOne</code> | 87 |
| <code>\markdownRendererHeadingSix</code> | 89 |
| <code>\markdownRendererHeadingThree</code> | 88 |
| <code>\markdownRendererHeadingTwo</code> | 88 |
| <code>\markdownRendererImage</code> | 90 |
| <code>\markdownRendererImageAttributeContextBegin</code> | 91 |
| <code>\markdownRendererImageAttributeContextEnd</code> | 91 |
| <code>\markdownRendererInlineHtmlComment</code> | 89 |
| <code>\markdownRendererInlineHtmlTag</code> | 90 |
| <code>\markdownRendererInlineMath</code> | 108 |
| <code>\markdownRendererInputBlockHtmlElement</code> | 90 |
| <code>\markdownRendererInputFencedCode</code> | 79 |
| <code>\markdownRendererInputRawBlock</code> | 101 |
| <code>\markdownRendererInputRawInline</code> | 101 |
| <code>\markdownRendererInputVerbatim</code> | 79 |
| <code>\markdownRendererInterblockSeparator</code> | 91 |
| <code>\markdownRendererJekyllDataBegin</code> | 110 |
| <code>\markdownRendererJekyllDataBoolean</code> | 112 |
| <code>\markdownRendererJekyllDataEmpty</code> | 113 |
| <code>\markdownRendererJekyllDataEnd</code> | 110 |
| <code>\markdownRendererJekyllDataMappingBegin</code> | 110 |
| <code>\markdownRendererJekyllDataMappingEnd</code> | 111 |
| <code>\markdownRendererJekyllDataNumber</code> | 112 |
| <code>\markdownRendererJekyllDataSequenceBegin</code> | 111 |
| <code>\markdownRendererJekyllDataSequenceEnd</code> | 111 |

| | |
|--|--------------------------------|
| <code>\markdownRendererJekyllDataString</code> | 112 |
| <code>\markdownRendererLineBlockBegin</code> | 92 |
| <code>\markdownRendererLineBlockEnd</code> | 92 |
| <code>\markdownRendererLink</code> | 93, 116 |
| <code>\markdownRendererLinkAttributeContextBegin</code> | 94 |
| <code>\markdownRendererLinkAttributeContextEnd</code> | 94 |
| <code>\markdownRendererMark</code> | 94 |
| <code>\markdownRendererNbsp</code> | 95 |
| <code>\markdownRendererNote</code> | 96 |
| <code>\markdownRendererOlBegin</code> | 96 |
| <code>\markdownRendererOlBeginTight</code> | 97 |
| <code>\markdownRendererOlEnd</code> | 100 |
| <code>\markdownRendererOlEndTight</code> | 100 |
| <code>\markdownRendererOlItem</code> | 40, 98 |
| <code>\markdownRendererOlItemEnd</code> | 98 |
| <code>\markdownRendererOlItemWithNumber</code> | 40, 98 |
| <code>\markdownRendererParagraphSeparator</code> | 92 |
| <code>\markdownRendererReplacementCharacter</code> | 102 |
| <code>\markdownRendererSectionBegin</code> | 102 |
| <code>\markdownRendererSectionEnd</code> | 102 |
| <code>\markdownRendererSoftLineBreak</code> | 93 |
| <code>\markdownRendererStrikeThrough</code> | 105 |
| <code>\markdownRendererStrongEmphasis</code> | 85 |
| <code>\markdownRendererSubscript</code> | 106 |
| <code>\markdownRendererSuperscript</code> | 106 |
| <code>\markdownRendererTable</code> | 107 |
| <code>\markdownRendererTableAttributeContextBegin</code> | 107 |
| <code>\markdownRendererTableAttributeContextEnd</code> | 107 |
| <code>\markdownRendererTextCite</code> | 78 |
| <code>\markdownRendererThematicBreak</code> | 108 |
| <code>\markdownRendererTickedBox</code> | 109 |
| <code>\markdownRendererUlBegin</code> | 76 |
| <code>\markdownRendererUlBeginTight</code> | 76 |
| <code>\markdownRendererUlEnd</code> | 77 |
| <code>\markdownRendererUlEndTight</code> | 77 |
| <code>\markdownRendererUlItem</code> | 77 |
| <code>\markdownRendererUlItemEnd</code> | 77 |
| <code>\markdownRendererUntickedBox</code> | 109 |
| <code>\markdownSetup</code> | 53, 53, 57, 132, 138, 363, 364 |
| <code>\markdownSetupSnippet</code> | 67, 67 |
| <code>\markdownWarning</code> | 128, 128 |
| <code>\markinline</code> | 50, 51, 52, 130, 131, 357, 361 |

| | |
|--|---------------------------|
| <code>\markinlinePlainTeX</code> | 361 |
| <code>new</code> | 7, 16, 17, 333 |
| <code>notes</code> | 36, 96 |
| <code>parsers</code> | 210, 243, 244 |
| <code>parsers.punctuation</code> | 211 |
| <code>pipeTables</code> | 7, 36, 42, 107 |
| <code>preserveTabs</code> | 37, 40, 244 |
| <code>rawAttribute</code> | 37, 37, 101, 102 |
| <code>reader</code> | 8, 28, 140, 210, 243, 287 |
| <code>reader->add_special_character</code> | 8, 9, 28, 280 |
| <code>reader->auto_link_email</code> | 270 |
| <code>reader->auto_link_url</code> | 270 |
| <code>reader->create_parser</code> | 244 |
| <code>reader->finalize_grammar</code> | 276, 337 |
| <code>reader->initialize_named_group</code> | 280 |
| <code>reader->insert_pattern</code> | 8, 9, 28, 276, 282 |
| <code>reader->lookup_note_reference</code> | 257 |
| <code>reader->lookup_reference</code> | 257 |
| <code>reader->normalize_tag</code> | 244 |
| <code>reader->options</code> | 243 |
| <code>reader->parser_functions</code> | 244 |
| <code>reader->parser_functions.name</code> | 244 |
| <code>reader->parsers</code> | 243, 243, 244 |
| <code>reader->register_link</code> | 256 |
| <code>reader->update_rule</code> | 276, 279, 282 |
| <code>reader->writer</code> | 243 |
| <code>reader.new</code> | 243, 243, 337 |
| <code>relativeReferences</code> | 38 |
| <code>\setupmarkdown</code> | 138 |
| <code>shiftHeadings</code> | 7, 38 |
| <code>singletonCache</code> | 16 |
| <code>slice</code> | 7, 39, 192, 204 |
| <code>smartEllipses</code> | 39, 84, 133 |
| <code>\startmarkdown</code> | 137, 138, 392 |
| <code>startNumber</code> | 40, 98, 99 |
| <code>\stopmarkdown</code> | 137, 138, 392 |
| <code>strikeThrough</code> | 40, 106, 389 |
| <code>stripIndent</code> | 40, 245 |
| <code>stripPercentSigns</code> | 353, 354 |

| | |
|-------------------------------|---------------------------------------|
| subscripts | 41, 106 |
| superscripts | 41, 106 |
| syntax | 277, 282 |
| tableAttributes | 42, 107 |
| tableCaptions | 7, 42, 42, 107 |
| taskLists | 43, 109, 379 |
| texComments | 43, 245 |
| texMathDollars | 44, 108 |
| texMathDoubleBackslash | 44, 108 |
| texMathSingleBackslash | 44, 108 |
| tightLists | 45, 76, 78, 82, 84, 97, 100, 101, 369 |
| underscores | 45 |
| unicodeNormalization | 17, 18 |
| unicodeNormalizationForm | 17, 18 |
| util.cache | 141, 141 |
| util.cache_verbatim | 141 |
| util.encode_json_string | 142 |
| util.err | 141 |
| util.escaper | 144 |
| util.expand_tabs_in_line | 142 |
| util.flatten | 143 |
| util.intersperse | 144 |
| util.map | 144 |
| util.pathname | 145 |
| util.rope_last | 143 |
| util.rope_to_string | 143 |
| util.table_copy | 142 |
| util.walk | 142, 143 |
| walkable_syntax | 8, 19, 25, 276, 279–282 |
| writer | 140, 140, 192, 287 |
| writer->active_attributes | 203, 203, 204 |
| writer->attribute_type_levels | 203 |
| writer->attributes | 201 |
| writer->block_html_element | 199 |
| writer->blockquote | 200 |
| writer->bulletitem | 198 |
| writer->bulletlist | 197 |
| writer->citations | 288 |
| writer->code | 196 |
| writer->contentblock | 293 |

| | |
|----------------------------------|----------|
| writer->defer_call | 210, 210 |
| writer->definitionlist | 295 |
| writer->display_math | 326 |
| writer->div_begin | 308 |
| writer->div_end | 308 |
| writer->document | 200 |
| writer->ellipsis | 194 |
| writer->emphasis | 199 |
| writer->escape | 196 |
| writer->escape_minimal | 195 |
| writer->escape_programmatic_text | 195 |
| writer->escape_typographic_text | 195 |
| writer->escaped_chars | 195, 195 |
| writer->escaped_minimal_strings | 194, 195 |
| writer->escaped_strings | 195 |
| writer->escaped_uri_chars | 194, 195 |
| writer->fancyitem | 298 |
| writer->fancylist | 297 |
| writer->fencedCode | 303 |
| writer->flatten_inlines | 192, 192 |
| writer->get_state | 209 |
| writer->hard_line_break | 194 |
| writer->heading | 208 |
| writer->identifier | 196 |
| writer->image | 197 |
| writer->infostring | 196 |
| writer->inline_html_comment | 199 |
| writer->inline_html_tag | 199 |
| writer->inline_math | 326 |
| writer->interblocksep | 194 |
| writer->is_writing | 192, 192 |
| writer->jekyllData | 329 |
| writer->lineblock | 313 |
| writer->link | 197 |
| writer->mark | 314 |
| writer->math | 196 |
| writer->nbsp | 193 |
| writer->note | 317 |
| writer->options | 192 |
| writer->orderedititem | 199 |
| writer->orderedlist | 198 |
| writer->pack | 193, 286 |

| | |
|-------------------------|---------------|
| writer->paragraph | 193 |
| writer->paragraphsep | 194 |
| writer->plain | 193 |
| writer->pop_attributes | 203, 204 |
| writer->push_attributes | 203, 203, 204 |
| writer->rawBlock | 304 |
| writer->rawInline | 323 |
| writer->set_state | 209 |
| writer->slice_begin | 192 |
| writer->slice_end | 192 |
| writer->soft_line_break | 194 |
| writer->space | 193 |
| writer->span | 287 |
| writer->strike_through | 324 |
| writer->string | 196 |
| writer->strong | 200 |
| writer->subscript | 324 |
| writer->suffix | 193 |
| writer->superscript | 325 |
| writer->table | 320 |
| writer->thematic_break | 194 |
| writer->checkbox | 200 |
| writer->undosep | 194, 285 |
| writer->uri | 196 |
| writer->verbatim | 200 |
| writer.new | 192, 192, 337 |