

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.19.0-0-g80fcf20
2022-12-23

Contents

1	Introduction	1	3	Implementation	107
1.1	Requirements	2	3.1	Lua Implementation	107
1.2	Feedback	5	3.2	Plain T _E X Implementation	234
1.3	Acknowledgements	5	3.3	L ^A T _E X Implementation	251
2	Interfaces	6	3.4	ConT _E Xt Implementation	276
2.1	Lua Interface	6		References	282
2.2	Plain T _E X Interface	42		Index	283
2.3	L ^A T _E X Interface	87			
2.4	ConT _E Xt Interface	103			

List of Figures

1	A block diagram of the Markdown package	7
2	A sequence diagram of typesetting a document using the T _E X interface	38
3	A sequence diagram of typesetting a document using the Lua CLI	39
4	Various formats of mathematical formulae	94
5	The banner of the Markdown package	95
6	A pushdown automaton that recognizes T _E X comments	170

1 Introduction

The Markdown package¹ converts markdown² 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 convertors, 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

¹See <https://ctan.org/pkg/markdown>.

²See <https://daringfireball.net/projects/markdown/basics>.

implementation of the package. The technical documentation contains only a limited number of tutorials and code examples. You can find more of these in the user manual.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 Novotný",
5   copyright = {"2009-2016 John MacFarlane, Hans Hagen",
6               "2016-2022 Vít Novotný"},
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 Lua_{TEX} engine:

LPeg \geq 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg \geq 0.10 is included in Lua_{TEX} \geq 0.72.0 (T_EXLive \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 Lua_{TEX} (T_EXLive \geq 2008).

```
13 local unicode
14 (function()
15   local ran_ok
16   ran_ok, unicode = pcall(require, "unicode")
```

If the Selene Unicode library is unavailable and we are using Lua \geq 5.3, we will use the built-in support for Unicode.

```
17   if not ran_ok then
```

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

```

18     unicode = {"utf8"={char=utf8.char}}
19   end
20 end()

```

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 (TeXLive \geq 2008).

```

21 local md5 = require("md5")

```

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

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

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^AT_EX3 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.

```

22 <@@=markdown>
23 \ifx\ExplSyntaxOn\undefined
24   \input expl3-generic\relax
25 \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.

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 2.1.6), 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) or if you enforce the use of shell using the `\markdownMode` macro, then unless your TeX engine is globally configured to enable shell access, you will need to provide the `-shell-escape` parameter to your engine when typesetting a document.

1.1.3 L^AT_EX Requirements

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

```
26 \NeedsTeXFormat{LaTeX2e}%
```

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 2.2.4 and 3.3.4) or L^AT_EX themes (see Section 2.3.2.2) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.1):

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

graphicx A package that provides the `\includegraphics` macro for the typesetting of images.

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

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 2.3.2.2).

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

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

gobble A package that provides the `\@gobblethree` TeX command that is used in the default renderer prototype for citations. The package is included in TeXLive ≥ 2016.

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

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 2.3.2.2.

grffile A package that extends the name processing of package graphics to support a larger range of file names in $2006 \leq \text{T}_{\text{E}}\text{X Live} \leq 2019$. Since $\text{T}_{\text{E}}\text{X Live} \geq 2020$, 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 2.3.2.2.

etoolbox A package that is used to polyfill the general hook management system in the default renderer prototypes for YAML metadata, see Section 3.3.4.6, and also in the default renderer prototype for attribute identifiers.

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

```
27 \RequirePackage{expl3}
```

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 2.2.4).

1.2 Feedback

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

1.3 Acknowledgements

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

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

⁵See <https://tex.stackexchange.com>.

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.

```
28 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` 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.

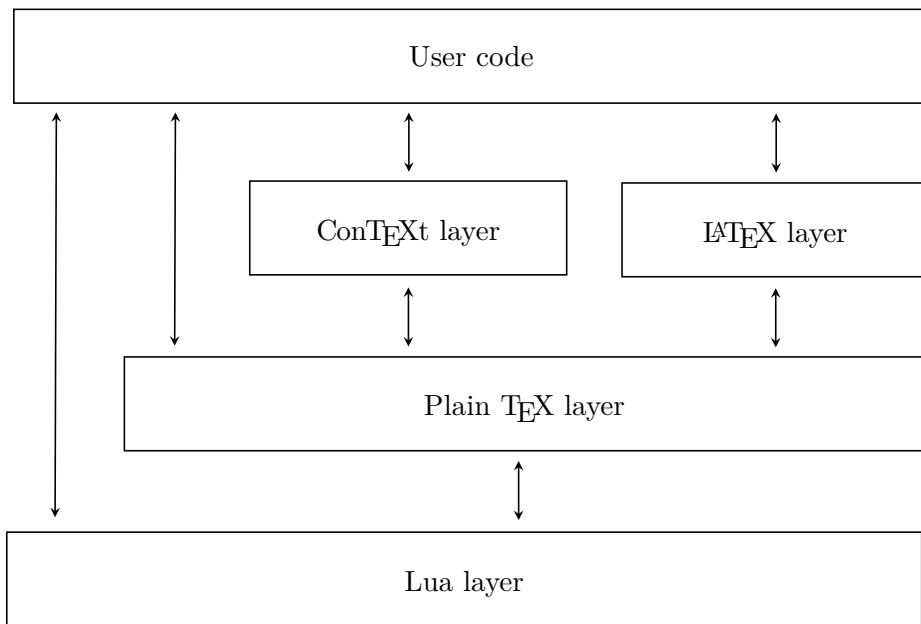


Figure 1: A block diagram of the Markdown package

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

```

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

```

2.1.2 User-Defined Syntax Extensions

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

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

```

29 local walkable_syntax = {
30   Block = {
31     "Blockquote",
32     "Verbatim",

```

```

33     "ThematicBreak",
34     "BulletList",
35     "OrderedList",
36     "Heading",
37     "DisplayHtml",
38     "Paragraph",
39     "Plain",
40   },
41   Inline = {
42     "Str",
43     "Space",
44     "Endline",
45     "UlOrStarLine",
46     "Strong",
47     "Emph",
48     "Link",
49     "Image",
50     "Code",
51     "AutoLinkUrl",
52     "AutoLinkEmail",
53     "AutoLinkRelativeReference",
54     "InlineHtml",
55     "HtmlEntity",
56     "EscapedChar",
57     "Smart",
58     "Symbol",
59   },
60 }

```

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 -> Emph` and `Inline -> Link` rules, we would call `reader->insert_pattern` with "`Inline after Emph`" (or "`Inline before Link`") 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.

```
61 local defaultOptions = {}
```

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

```
62 \ExplSyntaxOn
63 \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.

```
64 \prop_new:N \g_@@_lua_option_types_prop
65 \prop_new:N \g_@@_default_lua_options_prop
66 \seq_new:N \g_@@_option_layers_seq
67 \tl_const:Nn \c_@@_option_layer_lua_tl { lua }
68 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
69 \cs_new:Nn
70   \@@_add_lua_option:nnn
71   {
72     \@@_add_option:Vnnn
73     \c_@@_option_layer_lua_tl
74     { #1 }
75     { #2 }
76     { #3 }
77   }
78 \cs_new:Nn
79   \@@_add_option:nnnn
80   {
81     \seq_gput_right:cn
82     { g_@@_ #1 _options_seq }
83     { #2 }
84     \prop_gput:cnn
85     { g_@@_ #1 _option_types_prop }
86     { #2 }
87     { #3 }
88     \prop_gput:cnn
89     { g_@@_default_ #1 _options_prop }
90     { #2 }
91     { #4 }
92     \@@_typecheck_option:n
93     { #2 }
94   }
95 \cs_generate_variant:Nn
96   \@@_add_option:nnnn
```

```

97   { Vnnn }
98 \tl_const:Nn \c_@@_option_value_true_tl { true }
99 \tl_const:Nn \c_@@_option_value_false_tl { false }
100 \cs_new:Nn \@@_typecheck_option:n
101   {
102     \@@_get_option_type:nN
103     { #1 }
104     \l_tmpa_tl
105     \str_case_e:Vn
106     \l_tmpa_tl
107     {
108       { \c_@@_option_type_boolean_tl }
109       {
110         \@@_get_option_value:nN
111         { #1 }
112         \l_tmpa_tl
113         \bool_if:nF
114         {
115           \str_if_eq_p:VV
116           \l_tmpa_tl
117           \c_@@_option_value_true_tl ||
118           \str_if_eq_p:VV
119           \l_tmpa_tl
120           \c_@@_option_value_false_tl
121         }
122         {
123           \msg_error:nnnV
124           { @@ }
125           { failed-typecheck-for-boolean-option }
126           { #1 }
127           \l_tmpa_tl
128         }
129       }
130     }
131   }
132 \msg_new:nnn
133   { @@ }
134   { failed-typecheck-for-boolean-option }
135   {
136     Option~#1~has~value~#2,~
137     but~a~boolean~(true~or~false)~was~expected.
138   }
139 \cs_generate_variant:Nn
140   \str_case_e:nn
141   { Vn }
142 \cs_generate_variant:Nn
143   \msg_error:nnnn

```

```

144 { nnnV }
145 \seq_new:N \g_@@_option_types_seq
146 \tl_const:Nn \c_@@_option_type_clist_tl { clist }
147 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
148 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
149 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
150 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
151 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
152 \tl_const:Nn \c_@@_option_type_number_tl { number }
153 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
154 \tl_const:Nn \c_@@_option_type_path_tl { path }
155 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
156 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
157 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
158 \tl_const:Nn \c_@@_option_type_string_tl { string }
159 \seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
160 \cs_new:Nn
161   \@@_get_option_type:nN
162   {
163     \bool_set_false:N
164       \l_tmpa_bool
165     \seq_map_inline:Nn
166       \g_@@_option_layers_seq
167       {
168         \prop_get:cnNT
169           { g_@@_ ##1 _option_types_prop }
170           { #1 }
171         \l_tmpa_tl
172         {
173           \bool_set_true:N
174             \l_tmpa_bool
175           \seq_map_break:
176         }
177       }
178     \bool_if:nF
179       \l_tmpa_bool
180     {
181       \msg_error:nnn
182         { @@ }
183         { undefined-option }
184         { #1 }
185     }
186     \seq_if_in:NVF
187       \g_@@_option_types_seq
188       \l_tmpa_tl
189     {
190       \msg_error:nnnV

```

```

191         { @@ }
192         { unknown-option-type }
193         { #1 }
194         \l_tmpa_tl
195     }
196     \tl_set_eq:NN
197     #2
198     \l_tmpa_tl
199 }
200 \msg_new:nnn
201 { @@ }
202 { unknown-option-type }
203 {
204     Option~#1~has~unknown~type~#2.
205 }
206 \msg_new:nnn
207 { @@ }
208 { undefined-option }
209 {
210     Option~#1~is~undefined.
211 }
212 \cs_new:Nn
213 \@@_get_default_option_value:nN
214 {
215     \bool_set_false:N
216     \l_tmpa_bool
217     \seq_map_inline:Nn
218     \g_@@_option_layers_seq
219     {
220         \prop_get:cnNT
221         { g_@@_default_ ##1 _options_prop }
222         { #1 }
223         #2
224         {
225             \bool_set_true:N
226             \l_tmpa_bool
227             \seq_map_break:
228         }
229     }
230     \bool_if:nF
231     \l_tmpa_bool
232     {
233         \msg_error:nnn
234         { @@ }
235         { undefined-option }
236         { #1 }
237     }

```

```

238 }
239 \cs_new:Nn
240   \@@_get_option_value:nN
241   {
242     \@@_option_tl_to_csname:nN
243     { #1 }
244     \l_tmpa_tl
245     \cs_if_free:cTF
246     { \l_tmpa_tl }
247     {
248       \@@_get_default_option_value:nN
249       { #1 }
250       #2
251     }
252   {
253     \@@_get_option_type:nN
254     { #1 }
255     \l_tmpa_tl
256     \str_if_eq:NNTF
257     \c_@@_option_type_counter_tl
258     \l_tmpa_tl
259     {
260       \@@_option_tl_to_csname:nN
261       { #1 }
262       \l_tmpa_tl
263       \tl_set:Nx
264       #2
265       { \the \cs:w \l_tmpa_tl \cs_end: }
266     }
267     {
268       \@@_option_tl_to_csname:nN
269       { #1 }
270       \l_tmpa_tl
271       \tl_set:Nv
272       #2
273       { \l_tmpa_tl }
274     }
275   }
276 }
277 \cs_new:Nn \@@_option_tl_to_csname:nN
278   {
279     \tl_set:Nn
280     \l_tmpa_tl
281     { \str_uppercase:n { #1 } }
282     \tl_set:Nx
283     #2
284     {

```

```

285     markdownOption
286     \tl_head:f { \l_tmpa_tl }
287     \tl_tail:n { #1 }
288   }
289 }
290 \seq_new:N \g_@@_cases_seq
291 \cs_new:Nn \@@_with_various_cases:nn
292 {
293   \seq_clear:N
294   \l_tmpa_seq
295   \seq_map_inline:Nn
296   \g_@@_cases_seq
297   {
298     \tl_set:Nn
299     \l_tmpa_tl
300     { #1 }
301     \use:c { ##1 }
302     \l_tmpa_tl
303     \seq_put_right:NV
304     \l_tmpa_seq
305     \l_tmpa_tl
306   }
307   \seq_map_inline:Nn
308   \l_tmpa_seq
309   { #2 }
310 }
311 \cs_new:Nn \@@_camel_case:N
312 {
313   \regex_replace_all:nnN
314   { _ ([a-z]) }
315   { \c { str_uppercase:n } \cB\{ \1 \cE\} }
316   #1
317   \tl_set:Nx
318   #1
319   { #1 }
320 }
321 \seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
322 \cs_new:Nn \@@_snake_case:N
323 {
324   \regex_replace_all:nnN
325   { ([a-z])([A-Z]) }
326   { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
327   #1
328   \tl_set:Nx
329   #1
330   { #1 }
331 }

```

```
332 \seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }
```

2.1.4 File and Directory Names

`cacheDir`= $\langle path \rangle$ default: .

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

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

```
333 \@@_add_lua_option:nmn
334   { cacheDir }
335   { path }
336   { \markdownOptionOutputDir / _markdown_\jobname }
337 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 2.2.3.7 for more information.

```
338 \@@_add_lua_option:nmn
339   { contentBlocksLanguageMap }
340   { path }
341   { markdown-languages.json }
342 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 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied.

```

343 \@@_add_lua_option:nnn
344   { debugExtensionsFileName }
345   { path }
346   { \markdownOptionOutputDir / \jobname .debug-extensions.json }
347 defaultOptions.debugExtensionsFileName = "debug-extensions.json"

```

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

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

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

```

348 \@@_add_lua_option:nnn
349   { frozenCacheFileName }
350   { path }
351   { \markdownOptionCacheDir / frozenCache.tex }
352 defaultOptions.frozenCacheFileName = "frozenCache.tex"

```

2.1.5 Parser Options

`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.

```

353 \@@_add_lua_option:nnn
354   { blankBeforeBlockquote }
355   { boolean }
356   { false }
357 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.

```
358 \@@_add_lua_option:nnn
359 { blankBeforeCodeFence }
360 { boolean }
361 { false }

362 defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeDivFence=true, false` default: false

`true` Require a blank line before the closing fence of a fenced div.

`false` Do not require a blank line before the closing fence of a fenced div.

```
363 \@@_add_lua_option:nnn
364 { blankBeforeDivFence }
365 { boolean }
366 { false }

367 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.

```
368 \@@_add_lua_option:nnn
369 { blankBeforeHeading }
370 { boolean }
371 { false }

372 defaultOptions.blankBeforeHeading = false
```

`bracketedSpans=true, false`

default: `false`

`true` Enable the Pandoc bracketed spans extension:

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

`false` Disable the Pandoc bracketed spans extension:

```
373 \@@_add_lua_option:nnn
374 { bracketedSpans }
375 { boolean }
376 { false }

377 defaultOptions.bracketedSpans = false
```

`breakableBlockquotes=true, false`

default: `false`

`true` A blank line separates block quotes.

`false` Blank lines in the middle of a block quote are ignored.

```
378 \@@_add_lua_option:nnn
379 { breakableBlockquotes }
380 { boolean }
381 { false }

382 defaultOptions.breakableBlockquotes = false
```

`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.

```
383 \@@_add_lua_option:nnn
384 { citationNbsps }
385 { boolean }
386 { true }

387 defaultOptions.citationNbsps = true
```

`citations=true, false`

default: false

`true` Enable the Pandoc citation syntax extension:

```
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.

```
388 \@@_add_lua_option:nnn
389   { citations }
390   { boolean }
391   { false }
392 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.``
```

```
393 \@@_add_lua_option:nnn
394   { codeSpans }
395   { boolean }
396   { true }
397 defaultOptions.codeSpans = true
```

`contentBlocks=true, false` default: false

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

```
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.

```
398 \@@_add_lua_option:nnn
399 { contentBlocks }
400 { boolean }
401 { false }

402 defaultOptions.contentBlocks = false
```

`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 3.1.6) 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.

```
403 \@@_add_lua_option:nnn
404 { debugExtensions }
405 { boolean }
406 { false }

407 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.

```

408 \@@_add_lua_option:nnn
409   { definitionLists }
410   { boolean }
411   { false }

412 defaultOptions.definitionLists = false

```

eagerCache=true, false **default: true**

true Converted markdown documents will be cached in [cacheDir](#). This can be useful for post-processing the converted documents and for recovering historical versions of the documents from the cache. 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. Recursive nesting of markdown document fragments is undefined behavior when [eagerCache](#) is disabled.

```

413 \@@_add_lua_option:nnn
414   { eagerCache }
415   { boolean }
416   { true }

417 defaultOptions.eagerCache = true

```

`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}

```

```

418 \@_add_lua_option:nmn
419   { expectJekyllData }
420   { boolean }
421   { false }

422 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_EX directory structure.

A user-defined syntax extension is a Lua file in the following format:

```

local strike_through = {
  api_version = 2,
  grammar_version = 2,
  finalize_grammar = function(reader)
    local nonspacechar = lpeg.P(1) - lpeg.S("\t ")
    local doubleslashes = lpeg.P("//")
    local function between(p, starter, ender)
      ender = lpeg.B(nonspacechar) * ender
      return (starter * #nonspacechar
        * lpeg.Ct(p * (p - ender)^0) * ender)
    end

    local read_strike_through = between(
      lpeg.V("Inline"), doubleslashes, doubleslashes
    ) / function(s) return {"\st{", s, "}" end

    reader.insert_pattern("Inline after Emph", 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:

```
423 metadata.user_extension_api_version = 2
424 metadata.grammar_version = 2
```

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 `\luamref{reader}` object, such as the `\luamref{reader->insert_pattern}` and `\luamref{reader->add_special_character}` methods, see Section `<#luauserextensions>`.

```
425 \cs_generate_variant:Nn
426   \@@_add_lua_option:nnn
427   { nnV }
428 \@@_add_lua_option:nnV
429   { extensions }
430   { clist }
431   \c_empty_clist
432 defaultOptions.extensions = {}
```

`fancyLists=true, false`

default: `false`

`true` Enable the Pandoc fancy list extension:

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

`false` Disable the Pandoc fancy list extension.

```
433 \@@_add_lua_option:nnn
434   { fancyLists }
435   { boolean }
436   { false }
```



```
437 defaultOptions.fancyLists = false
```

`fencedCode=true, false`

default: false

`true` Enable the commonmark fenced code block extension:

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

``` html


```

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


```

`false` Disable the commonmark fenced code block extension.

```
438 \@@_add_lua_option:nnn
439 { fencedCode }
440 { boolean }
441 { false }

442 defaultOptions.fencedCode = false
```

`fencedDivs=true, false`

default: false

`true` Enable the Pandoc fenced divs extension:

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

And another.
:::::
```

`false` Disable the Pandoc fenced divs extension:

```

443 \@@_add_lua_option:nnn
444   { fencedDivs }
445   { boolean }
446   { false }

447 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 \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.

```

448 \@@_add_lua_option:nnn
449   { finalizeCache }
450   { boolean }
451   { false }

452 defaultOptions.finalizeCache = false

```

`frozenCacheCounter=<number>`

default: 0

The number of the current markdown document that will be stored in an output file (frozen cache) when the `finalizeCache` is enabled. When the document number is 0, then a new frozen cache will be created. Otherwise, the frozen cache will be appended.

Each frozen cache entry will define a \TeX macro `\markdownFrozenCache<number>` that will typeset markdown document number `<number>`.

```

453 \@@_add_lua_option:nnn
454   { frozenCacheCounter }
455   { counter }
456   { 0 }

457 defaultOptions.frozenCacheCounter = 0

```

`hardLineBreaks=true, false` default: false

`true` Interpret all newlines within a paragraph as hard line breaks instead of spaces.

`false` Interpret all newlines within a paragraph as spaces.

```
458 \@@_add_lua_option:nnn
459 { hardLineBreaks }
460 { boolean }
461 { false }
462 defaultOptions.hardLineBreaks = 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.

```
463 \@@_add_lua_option:nnn
464 { hashEnumerators }
465 { boolean }
466 { false }
467 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.

```
468 \@@_add_lua_option:nnn
469 { headerAttributes }
470 { boolean }
471 { false }
472 defaultOptions.headerAttributes = false
```

`html=true, false` default: `false`

- `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.

```
473 \@@_add_lua_option:nnn
474   { html }
475   { boolean }
476   { false }

477 defaultOptions.html = false
```

`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.

```
478 \@@_add_lua_option:nnn
479   { hybrid }
480   { boolean }
481   { false }

482 defaultOptions.hybrid = false
```

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

- `true` Enable the Pandoc inline note syntax extension:

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

`false` Disable the Pandoc inline note syntax extension.

The `inlineFootnotes` option has been deprecated and will be removed in Markdown 3.0.0.

```
483 \@@_add_lua_option:nnn
484   { inlineFootnotes }
485   { boolean }
486   { false }
487 \@@_add_lua_option:nnn
488   { inlineNotes }
489   { boolean }
490   { false }

491 defaultOptions.inlineFootnotes = false
492 defaultOptions.inlineNotes = false
```

`jeekyllData=true, false`

default: `false`

`true` Enable the Pandoc `yaml_metadata_block` syntax extension 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.

```
493 \@@_add_lua_option:nnn
494   { jeekyllData }
495   { boolean }
496   { false }

497 defaultOptions.jekyllData = false
```

`notes=true, false`

default: false

`true` Enable the Pandoc note syntax extension:

```
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.

The footnotes option has been deprecated and will be removed in Markdown 3.0.0.

```
498 \@@_add_lua_option:nnn
499   { footnotes }
500   { boolean }
501   { false }
502 \@@_add_lua_option:nnn
503   { notes }
504   { boolean }
505   { false }

506 defaultOptions.footnotes = false
507 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.

```
508 \@@_add_lua_option:nnn
509   { pipeTables }
510   { boolean }
511   { false }

512 defaultOptions.pipeTables = false
```

`preserveTabs=true, false` default: `false`

`true` Preserve tabs in code block and fenced code blocks.
`false` Convert any tabs in the input to spaces.

```
513 \@@_add_lua_option:nnn
514   { preserveTabs }
515   { boolean }
516   { false }

517 defaultOptions.preserveTabs = false
```

`rawAttribute=true, false` default: `false`

`true` Enable the Pandoc raw attribute syntax extension:

```
~$H_2 O$~{=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.

```
518 \@@_add_lua_option:nnn
519 { rawAttribute }
520 { boolean }
521 { false }

522 defaultOptions.rawAttribute = true
```

`relativeReferences=true, false` default: `false`

`true` Enable relative references<sup>6</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.

```
523 \@@_add_lua_option:nnn
524 { relativeReferences }
525 { boolean }
526 { false }

527 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.

```
528 \@@_add_lua_option:nnn
529 { shiftHeadings }
530 { number }
531 { 0 }

532 defaultOptions.shiftHeadings = 0
```

---

<sup>6</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 (`$`) selects the end of a document.
- `^<identifier>` selects the beginning of a section with the HTML attribute `#<identifier>` (see the `headerAttributes` option).
- `$<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.

```
533 \@@_add_lua_option:nnn
534 { slice }
535 { slice }
536 { ^-$ }
537 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.

```
538 \@@_add_lua_option:nnn
539 { smartEllipses }
540 { boolean }
541 { false }
542 defaultOptions.smartEllipses = false
```

`startNumber`=`true, false` default: `true`

- `true` Make the number in the first item of an ordered lists significant. The item numbers will be passed to the `\markdownRendererO1ItemWithNumber`  $\TeX$  macro.
- `false` Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererO1Item`  $\TeX$  macro.

```

543 \@@_add_lua_option:nnn
544 { startNumber }
545 { boolean }
546 { true }

547 defaultOptions.startNumber = true

```

`strikeThrough=true, false`

default: false

**true** Enable the Pandoc strike-through syntax extension:

```
This is deleted text.
```

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

```

548 \@@_add_lua_option:nnn
549 { strikeThrough }
550 { boolean }
551 { false }

552 defaultOptions.strikeThrough = false

```

`stripIndent=true, false`

default: false

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

```

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

```

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

```

553 \@@_add_lua_option:nnn
554 { stripIndent }
555 { boolean }
556 { false }

557 defaultOptions.stripIndent = false

```

`subscripts=true, false` default: false

`true` Enable the Pandoc subscript syntax extension:

```
H2O is a liquid.
```

`false` Disable the Pandoc subscript syntax extension.

```
558 \@@_add_lua_option:nnn
559 { subscripts }
560 { boolean }
561 { false }

562 defaultOptions.subscripts = false
```

`superscripts=true, false` default: false

`true` Enable the Pandoc superscript syntax extension:

```
210 is 1024.
```

`false` Disable the Pandoc superscript syntax extension.

```
563 \@@_add_lua_option:nnn
564 { superscripts }
565 { boolean }
566 { false }

567 defaultOptions.superscripts = false
```

`tableCaptions=true, false` default: false

`true` Enable the Pandoc `table_captions` syntax extension for pipe tables (see the `pipeTables` option).

```
| 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_captions` syntax extension.

```

568 \@@_add_lua_option:nnn
569 { tableCaptions }
570 { boolean }
571 { false }

572 defaultOptions.tableCaptions = false

```

`taskLists=true, false`

default: false

**true** Enable the Pandoc `task_lists` syntax extension.

```

- [] an unticked task list item
- [/] a half-checked task list item
- [X] a ticked task list item

```

**false** Disable the Pandoc `task_lists` syntax extension.

```

573 \@@_add_lua_option:nnn
574 { taskLists }
575 { boolean }
576 { false }

577 defaultOptions.taskLists = false

```

`texComments=true, false`

default: false

**true** Strip T<sub>E</sub>X-style comments.

```

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

```

Always enabled when `hybrid` is enabled.

**false** Do not strip T<sub>E</sub>X-style comments.

```

578 \@@_add_lua_option:nnn
579 { texComments }
580 { boolean }
581 { false }

582 defaultOptions.texComments = 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.

```
583 \@@_add_lua_option:nnn
584 { tightLists }
585 { boolean }
586 { true }
587 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.

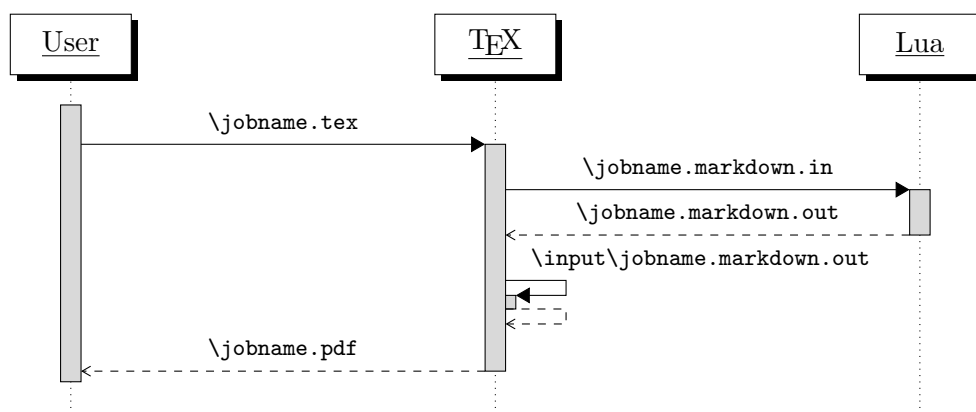
```
588 \@@_add_lua_option:nnn
589 { underscores }
590 { boolean }
591 { true }
592 \ExplSyntaxOff
593 defaultOptions.underscores = true
```

## 2.1.6 Command-Line Interface

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

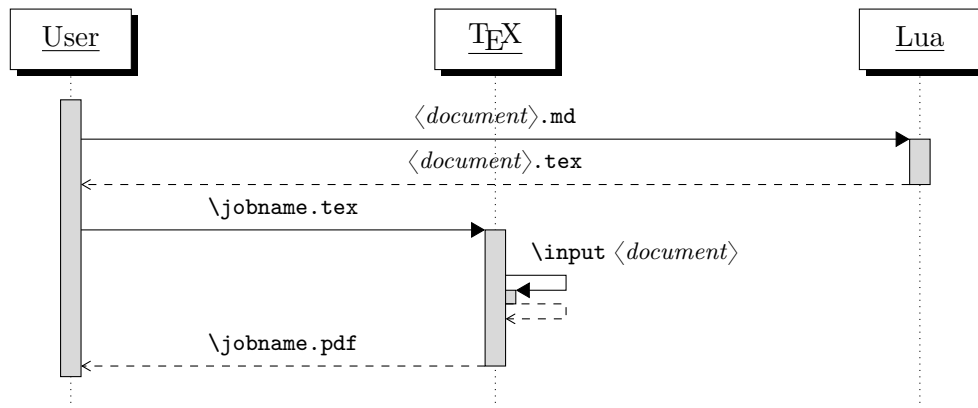
This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted T<sub>E</sub>X documents are cached on the file system, taking up increasing amount of space. Unless the T<sub>E</sub>X engine includes a Lua interpreter, the package also requires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

A solution to the above problems is to decouple the conversion from the typesetting. For this reason, a command-line Lua interface for converting a markdown document to T<sub>E</sub>X is also provided, see Figure 3.



**Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the T<sub>E</sub>X interface**

```
594
595 local HELP_STRING = [[
596 Usage: texlua]] .. arg[0] .. [[[OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
597 where OPTIONS are documented in the Lua interface section of the
598 technical Markdown package documentation.
599
600 When OUTPUT_FILE is unspecified, the result of the conversion will be
601 written to the standard output. When INPUT_FILE is also unspecified, the
602 result of the conversion will be read from the standard input.
603
604 Report bugs to: witiko@mail.muni.cz
605 Markdown package home page: <https://github.com/witiko/markdown>]]
606
```



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

```

607 local VERSION_STRING = [[
608 markdown-cli.lua (Markdown)]] .. metadata.version .. [[
609
610 Copyright (C)]] .. table.concat(metadata.copyright,
611 "\nCopyright (C) ") .. [[
612
613 License:]] .. metadata.license
614
615 local function warn(s)
616 io.stderr:write("Warning: " .. s .. "\n") end
617
618 local function error(s)
619 io.stderr:write("Error: " .. s .. "\n")
620 os.exit(1)
621 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.

```

622 local function camel_case(option_name)
623 local cased_option_name = option_name:gsub("_(%l)", function(match)
624 return match:sub(2, 2):upper()
625 end)
626 return cased_option_name
627 end
628
629 local function snake_case(option_name)
630 local cased_option_name = option_name:gsub("%l%u", function(match)
631 return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()

```

```

632 end)
633 return cased_option_name
634 end
635
636 local cases = {camel_case, snake_case}
637 local various_case_options = {}
638 for option_name, _ in pairs(defaultOptions) do
639 for _, case in ipairs(cases) do
640 various_case_options[case(option_name)] = option_name
641 end
642 end
643
644 local process_options = true
645 local options = {}
646 local input_filename
647 local output_filename
648 for i = 1, #arg do
649 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.

```

650 if arg[i] == "--" then
651 process_options = false
652 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.

```

653 elseif arg[i]:match("=") then
654 local key, value = arg[i]:match("(.)=(.*)")
655 if defaultOptions[key] == nil then
656 key = various_case_options[key]
657 end

```

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

```

658 local default_type = type(defaultOptions[key])
659 if default_type == "boolean" then
660 options[key] = (value == "true")
661 elseif default_type == "number" then
662 options[key] = tonumber(value)
663 elseif default_type == "table" then
664 options[key] = {}
665 for item in value:gmatch("[^,]+") do
666 table.insert(options[key], item)
667 end

```



```

668 else
669 if default_type ~= "string" then
670 if default_type == "nil" then
671 warn('Option "' .. key .. '" not recognized.')
672 else
673 warn('Option "' .. key .. '" type not recognized, please file ' ..
674 'a report to the package maintainer.')
675 end
676 warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
677 key .. '" as a string.')
678 end
679 options[key] = value
680 end
681 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.

```

682 elseif arg[i] == "--help" or arg[i] == "-h" then
683 print(HELP_STRING)
684 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.

```

685 elseif arg[i] == "--version" or arg[i] == "-v" then
686 print(VERSION_STRING)
687 os.exit()
688 end
689 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  $\text{\TeX}$  document.

```

690 if input_filename == nil then
691 input_filename = arg[i]

```

The first argument that matches none of the above patterns is assumed to be the output filename. The output filename should correspond to the  $\text{\TeX}$  document that will result from the conversion.

```

692 elseif output_filename == nil then
693 output_filename = arg[i]
694 else
695 error('Unexpected argument: "' .. arg[i] .. "'.')
696 end
697 ::continue::
698 end

```

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

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

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

```
\input hello
```

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

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

```
699 \def\markdownLastModified{((LASTMODIFIED))}%
700 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

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

### 2.2.1 Typesetting Markdown

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

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

```
701 \let\markdownBegin\relax
702 \let\markdownEnd\relax
```

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

There are several limitations to the macros you need to be aware of. The first limitation concerns the `\markdownEnd` macro, which must be visible directly from the input line buffer (it may not be produced as a result of input expansion). Otherwise, it will not be recognized as the end of the markdown string. As a corollary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T<sub>E</sub>X [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 T<sub>E</sub>X code, the characters `c`, `e`, and `f` will not appear in the output.

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

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

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

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

The `\markdownInput` macro accepts a single parameter with the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain T<sub>E</sub>X.

```
703 \let\markdownInput\relax
```

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

The following example plain T<sub>E</sub>X 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 T<sub>E</sub>X document and executes the T<sub>E</sub>X 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.

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

```
705 \ExplSyntaxOn
706 \seq_new:N \g_@@_plain_tex_options_seq
```

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

```
707 \prop_new:N \g_@@_plain_tex_option_types_prop
708 \prop_new:N \g_@@_default_plain_tex_options_prop
709 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
710 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
711 \cs_new:Nn
712 \@@_add_plain_tex_option:nnn
713 {
714 \@@_add_option:Vnnn
715 \c_@@_option_layer_plain_tex_tl
716 { #1 }
717 { #2 }
718 { #3 }
719 }
```

**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`.

```
720 \@@_add_plain_tex_option:nnn
721 { frozenCache }
722 { boolean }
```

```
723 { false }
```

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

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

**2.2.2.2 File and Directory Names** The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain T<sub>E</sub>X in T<sub>E</sub>X engines without the `\directlua` primitive. It defaults to `\jobname.markdown.lua`, where `\jobname` is the base name of the document being typeset.

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

```
724 \@@_add_plain_tex_option:nnn
725 { helperScriptFileName }
726 { path }
727 { \jobname.markdown.lua }
```

The `helperScriptFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the helper Lua script file, use the `\g_luabridge_helper_script_filename_str` macro from the `lt3luabridge` package.

```
728 \str_new:N
729 \g_luabridge_helper_script_filename_str
730 \tl_gset:Nn
731 \g_luabridge_helper_script_filename_str
732 { \markdownOptionHelperScriptFileName }
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T<sub>E</sub>X source. It defaults to `\jobname.markdown.in`. The same limitations as in the case of the `helperScriptFileName` macro apply here.

```
733 \@@_add_plain_tex_option:nnn
734 { inputTempFileName }
735 { path }
736 { \jobname.markdown.in }
```

The `\markdownOptionOutputTempFileName` macro sets the filename of the temporary output file that is created during the conversion from markdown to plain T<sub>E</sub>X in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `helperScriptFileName` macro.

```
737 \@@_add_plain_tex_option:nnn
738 { outputTempFileName }
```

```

739 { path }
740 { \jobname.markdown.out }

```

The `outputTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0.

```

741 \str_new:N
742 \g_luabridge_standard_output_filename_str
743 \tl_gset:Nn
744 \g_luabridge_standard_output_filename_str
745 { \markdownOptionOutputTempFileName }

```

The `\markdownOptionErrorTempFileName` macro sets the filename of the temporary output file that is created when a Lua error is encountered during the conversion from markdown to plain  $\TeX$  in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `helperScriptFileName` macro.

```

746 \@@_add_plain_tex_option:nnn
747 { errorTempFileName }
748 { path }
749 { \jobname.markdown.err }

```

The `errorTempFileName` macro has been deprecated and will be removed in Markdown 3.0.0. To control the filename of the temporary file for Lua errors, use the `\g_luabridge_error_output_filename_str` macro from the `lt3luabridge` package.

```

750 \str_new:N
751 \g_luabridge_error_output_filename_str
752 \tl_gset:Nn
753 \g_luabridge_error_output_filename_str
754 { \markdownOptionErrorTempFileName }

```

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 `..`.

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 `helperScriptFileName` macro.

```

755 \@@_add_plain_tex_option:nnn
756 { outputDir }
757 { path }
758 { . }

```

Here, we automatically define plain  $\TeX$  macros for the above plain  $\TeX$  options.

Furthermore, we also define macros 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.

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 `helperScriptFileName` macro.

```

759 \cs_new:Nn \@@_plain_tex_define_option_commands:
760 {
761 \seq_map_inline:Nn
762 \g_@@_option_layers_seq
763 {
764 \seq_map_inline:cn
765 { g_@@_ ##1 _options_seq }
766 {
767 \@@_plain_tex_define_option_command:n
768 { ####1 }
769 }
770 }
771 }
772 \cs_new:Nn \@@_plain_tex_define_option_command:n
773 {
774 \@@_get_default_option_value:nN
775 { #1 }
776 \l_tmpa_tl
777 \@@_set_option_value:nV
778 { #1 }
779 \l_tmpa_tl
780 }
781 \cs_new:Nn
782 \@@_set_option_value:nn
783 {
784 \@@_define_option:n
785 { #1 }
786 \@@_get_option_type:nN
787 { #1 }
788 \l_tmpa_tl
789 \str_if_eq:NNTF
790 \c_@@_option_type_counter_tl
791 \l_tmpa_tl
792 {
793 \@@_option_tl_to_csname:nN
794 { #1 }
795 \l_tmpa_tl
796 \int_gset:cn
797 { \l_tmpa_tl }
798 { #2 }
799 }
800 {
801 \@@_option_tl_to_csname:nN
802 { #1 }

```

```

803 \l_tmpa_tl
804 \cs_set:cpn
805 { \l_tmpa_tl }
806 { #2 }
807 }
808 }
809 \cs_generate_variant:Nn
810 \@@_set_option_value:nn
811 { nV }
812 \cs_new:Nn
813 \@@_define_option:n
814 {
815 \@@_option_tl_to_csname:nN
816 { #1 }
817 \l_tmpa_tl
818 \cs_if_free:cT
819 { \l_tmpa_tl }
820 {
821 \@@_get_option_type:nN
822 { #1 }
823 \l_tmpb_tl
824 \str_if_eq:NNT
825 \c_@@_option_type_counter_tl
826 \l_tmpb_tl
827 {
828 \@@_option_tl_to_csname:nN
829 { #1 }
830 \l_tmpa_tl
831 \int_new:c
832 { \l_tmpa_tl }
833 }
834 }
835 }
836 \@@_plain_tex_define_option_commands:

```

**2.2.2.3 Miscellaneous Options** The `\markdownOptionStripPercentSigns` macro controls whether a percent sign (%) at the beginning of a line will be discarded when buffering Markdown input (see Section 3.2.4) or not. Notably, this enables the use of markdown when writing T<sub>E</sub>X package documentation using the Doc L<sup>A</sup>T<sub>E</sub>X package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

837 \seq_gput_right:Nn
838 \g_@@_plain_tex_options_seq
839 { stripPercentSigns }
840 \prop_gput:Nnn
841 \g_@@_plain_tex_option_types_prop

```



```

842 { stripPercentSigns }
843 { boolean }
844 \prop_gput:Nnx
845 \g_@@_default_plain_tex_options_prop
846 { stripPercentSigns }
847 { false }
848 \ExplSyntaxOff

```

### 2.2.3 Token Renderers

The following T<sub>E</sub>X macros may occur inside the output of the converter functions exposed by the Lua interface (see Section 2.1.1) and represent the parsed markdown tokens. These macros are intended to be redefined by the user who is typesetting a document. By default, they point to the corresponding prototypes (see Section 2.2.4).

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

```

849 \ExplSyntaxOn
850 \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.

```

851 \prop_new:N \g_@@_renderer_arities_prop
852 \ExplSyntaxOff

```

**2.2.3.1 Attribute Renderers** The following macros are only produced, when the `headerAttributes` option is enabled.

`\markdownRendererAttributeIdentifier` represents the  $\langle identifier \rangle$  of a markdown element (`id="⟨identifier⟩"` in HTML and `#⟨identifier⟩` in Markdown's `headerAttributes` syntax extension). 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's `headerAttributes` syntax extension). 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.

```

853 \def\markdownRendererAttributeIdentifier{%
854 \markdownRendererAttributeIdentifierPrototype}%
855 \ExplSyntaxOn
856 \seq_gput_right:Nn
857 \g_@@_renderers_seq
858 { attributeIdentifier }
859 \prop_gput:Nnn

```

```

860 \g_@@_renderer_arities_prop
861 { attributeIdentifier }
862 { 1 }
863 \ExplSyntaxOff
864 \def\markdownRendererAttributeName{%
865 \markdownRendererAttributeNamePrototype}%
866 \ExplSyntaxOn
867 \seq_gput_right:Nn
868 \g_@@_renderers_seq
869 { attributeName }
870 \prop_gput:Nnn
871 \g_@@_renderer_arities_prop
872 { attributeName }
873 { 1 }
874 \ExplSyntaxOff
875 \def\markdownRendererAttributeKeyValue{%
876 \markdownRendererAttributeKeyValuePrototype}%
877 \ExplSyntaxOn
878 \seq_gput_right:Nn
879 \g_@@_renderers_seq
880 { attributeKeyValue }
881 \prop_gput:Nnn
882 \g_@@_renderer_arities_prop
883 { attributeKeyValue }
884 { 2 }
885 \ExplSyntaxOff

```

**2.2.3.2 Block Quote Renderers** The `\markdownRendererBlockQuoteBegin` macro represents the beginning of a block quote. The macro receives no arguments.

```

886 \def\markdownRendererBlockQuoteBegin{%
887 \markdownRendererBlockQuoteBeginPrototype}%
888 \ExplSyntaxOn
889 \seq_gput_right:Nn
890 \g_@@_renderers_seq
891 { blockQuoteBegin }
892 \prop_gput:Nnn
893 \g_@@_renderer_arities_prop
894 { blockQuoteBegin }
895 { 0 }
896 \ExplSyntaxOff

```

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

```

897 \def\markdownRendererBlockQuoteEnd{%
898 \markdownRendererBlockQuoteEndPrototype}%
899 \ExplSyntaxOn

```

```

900 \seq_gput_right:Nn
901 \g_@@_renderers_seq
902 { blockQuoteEnd }
903 \prop_gput:Nnn
904 \g_@@_renderer_arities_prop
905 { blockQuoteEnd }
906 { 0 }
907 \ExplSyntaxOff

```

**2.2.3.3 Bracketed Spans 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 an inline bracketed span in which the attributes of the span apply. The macros receive no arguments.

```

908 \def\markdownRendererBracketedSpanAttributeContextBegin{%
909 \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
910 \ExplSyntaxOn
911 \seq_gput_right:Nn
912 \g_@@_renderers_seq
913 { bracketedSpanAttributeContextBegin }
914 \prop_gput:Nnn
915 \g_@@_renderer_arities_prop
916 { bracketedSpanAttributeContextBegin }
917 { 0 }
918 \ExplSyntaxOff
919 \def\markdownRendererBracketedSpanAttributeContextEnd{%
920 \markdownRendererBracketedSpanAttributeContextEndPrototype}%
921 \ExplSyntaxOn
922 \seq_gput_right:Nn
923 \g_@@_renderers_seq
924 { bracketedSpanAttributeContextEnd }
925 \prop_gput:Nnn
926 \g_@@_renderer_arities_prop
927 { bracketedSpanAttributeContextEnd }
928 { 0 }
929 \ExplSyntaxOff

```

**2.2.3.4 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.

```

930 \def\markdownRendererUlBegin{%
931 \markdownRendererUlBeginPrototype}%
932 \ExplSyntaxOn
933 \seq_gput_right:Nn
934 \g_@@_renderers_seq

```

```

935 { ulBegin }
936 \prop_gput:Nnn
937 \g_@@_renderer_arities_prop
938 { ulBegin }
939 { 0 }
940 \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.

```

941 \def\markdownRendererUlBeginTight{%
942 \markdownRendererUlBeginTightPrototype}%
943 \ExplSyntaxOn
944 \seq_gput_right:Nn
945 \g_@@_renderers_seq
946 { ulBeginTight }
947 \prop_gput:Nnn
948 \g_@@_renderer_arities_prop
949 { ulBeginTight }
950 { 0 }
951 \ExplSyntaxOff

```

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

```

952 \def\markdownRendererUlItem{%
953 \markdownRendererUlItemPrototype}%
954 \ExplSyntaxOn
955 \seq_gput_right:Nn
956 \g_@@_renderers_seq
957 { ulItem }
958 \prop_gput:Nnn
959 \g_@@_renderer_arities_prop
960 { ulItem }
961 { 0 }
962 \ExplSyntaxOff

```

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

```

963 \def\markdownRendererUlItemEnd{%
964 \markdownRendererUlItemEndPrototype}%
965 \ExplSyntaxOn
966 \seq_gput_right:Nn
967 \g_@@_renderers_seq
968 { ulItemEnd }
969 \prop_gput:Nnn

```

```

970 \g_@@_renderer_arities_prop
971 { ulItemEnd }
972 { 0 }
973 \ExplSyntaxOff

```

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

```

974 \def\markdownRendererUEnd{%
975 \markdownRendererUEndPrototype}%
976 \ExplSyntaxOn
977 \seq_gput_right:Nn
978 \g_@@_renderers_seq
979 { ulEnd }
980 \prop_gput:Nnn
981 \g_@@_renderer_arities_prop
982 { ulEnd }
983 { 0 }
984 \ExplSyntaxOff

```

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

```

985 \def\markdownRendererUEndTight{%
986 \markdownRendererUEndTightPrototype}%
987 \ExplSyntaxOn
988 \seq_gput_right:Nn
989 \g_@@_renderers_seq
990 { ulEndTight }
991 \prop_gput:Nnn
992 \g_@@_renderer_arities_prop
993 { ulEndTight }
994 { 0 }
995 \ExplSyntaxOff

```

**2.2.3.5 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.

```

996 \def\markdownRendererInputVerbatim{%
997 \markdownRendererInputVerbatimPrototype}%
998 \ExplSyntaxOn
999 \seq_gput_right:Nn
1000 \g_@@_renderers_seq
1001 { inputVerbatim }

```

```

1002 \prop_gput:Nnn
1003 \g_@@_renderer_arities_prop
1004 { inputVerbatim }
1005 { 1 }
1006 \ExplSyntaxOff

```

The `\markdownRendererInputFencedCode` macro represents a fenced code block. This macro will only be produced, when the `fencedCode` option is enabled. The macro receives two arguments that correspond to the filename of a file containing the code block contents and to the code fence infostring.

```

1007 \def\markdownRendererInputFencedCode{%
1008 \markdownRendererInputFencedCodePrototype}%
1009 \ExplSyntaxOn
1010 \seq_gput_right:Nn
1011 \g_@@_renderers_seq
1012 { inputFencedCode }
1013 \prop_gput:Nnn
1014 \g_@@_renderer_arities_prop
1015 { inputFencedCode }
1016 { 2 }
1017 \ExplSyntaxOff

```

**2.2.3.6 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.

```

1018 \def\markdownRendererCodeSpan{%
1019 \markdownRendererCodeSpanPrototype}%
1020 \ExplSyntaxOn
1021 \seq_gput_right:Nn
1022 \g_@@_renderers_seq
1023 { codeSpan }
1024 \prop_gput:Nnn
1025 \g_@@_renderer_arities_prop
1026 { codeSpan }
1027 { 1 }
1028 \ExplSyntaxOff

```

**2.2.3.7 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.

```

1029 \def\markdownRendererContentBlock{%
1030 \markdownRendererContentBlockPrototype}%

```

```

1031 \ExplSyntaxOn
1032 \seq_gput_right:Nn
1033 \g_@@_renderers_seq
1034 { contentBlock }
1035 \prop_gput:Nnn
1036 \g_@@_renderer_arities_prop
1037 { contentBlock }
1038 { 4 }
1039 \ExplSyntaxOff

```

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

```

1040 \def\markdownRendererContentBlockOnlineImage{%
1041 \markdownRendererContentBlockOnlineImagePrototype}%
1042 \ExplSyntaxOn
1043 \seq_gput_right:Nn
1044 \g_@@_renderers_seq
1045 { contentBlockOnlineImage }
1046 \prop_gput:Nnn
1047 \g_@@_renderer_arities_prop
1048 { contentBlockOnlineImage }
1049 { 4 }
1050 \ExplSyntaxOff

```

The `\markdownRendererContentBlockCode` macro represents an `iA,Writer` content block that was recognized as a file in a known programming language by its filename extension  $s$ . If any `markdown-languages.json` file found by `kpathsea`<sup>7</sup> contains a record  $(k, v)$ , then a non-online-image content block with the filename extension  $s, s:\text{lower}() = k$  is considered to be in a known programming language  $v$ . The macro receives five arguments: the local file name extension  $s$  cast to the lower case, the language  $v$ , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

1051 \def\markdownRendererContentBlockCode{%
1052 \markdownRendererContentBlockCodePrototype}%
1053 \ExplSyntaxOn

```

---

<sup>7</sup>Filenames other than `markdown-languages.json` may be specified using the `contentBlocksLanguageMap` Lua option.

```

1054 \seq_gput_right:Nn
1055 \g_@@_renderers_seq
1056 { contentBlockCode }
1057 \prop_gput:Nnn
1058 \g_@@_renderer_arities_prop
1059 { contentBlockCode }
1060 { 5 }
1061 \ExplSyntaxOff

```

**2.2.3.8 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.

```

1062 \def\markdownRendererDlBegin{%
1063 \markdownRendererDlBeginPrototype}%
1064 \ExplSyntaxOn
1065 \seq_gput_right:Nn
1066 \g_@@_renderers_seq
1067 { dlBegin }
1068 \prop_gput:Nnn
1069 \g_@@_renderer_arities_prop
1070 { dlBegin }
1071 { 0 }
1072 \ExplSyntaxOff

```

The `\markdownRendererDlBeginTight` macro represents the beginning of a definition list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `tightLists` option is disabled. The macro receives no arguments.

```

1073 \def\markdownRendererDlBeginTight{%
1074 \markdownRendererDlBeginTightPrototype}%
1075 \ExplSyntaxOn
1076 \seq_gput_right:Nn
1077 \g_@@_renderers_seq
1078 { dlBeginTight }
1079 \prop_gput:Nnn
1080 \g_@@_renderer_arities_prop
1081 { dlBeginTight }
1082 { 0 }
1083 \ExplSyntaxOff

```

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

```

1084 \def\markdownRendererDlItem{%

```



```

1085 \markdownRendererDlItemPrototype}%
1086 \ExplSyntaxOn
1087 \seq_gput_right:Nn
1088 \g_@@_renderers_seq
1089 { dlItem }
1090 \prop_gput:Nnn
1091 \g_@@_renderer_arities_prop
1092 { dlItem }
1093 { 1 }
1094 \ExplSyntaxOff

```

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

```

1095 \def\markdownRendererDlItemEnd{%
1096 \markdownRendererDlItemEndPrototype}%
1097 \ExplSyntaxOn
1098 \seq_gput_right:Nn
1099 \g_@@_renderers_seq
1100 { dlItemEnd }
1101 \prop_gput:Nnn
1102 \g_@@_renderer_arities_prop
1103 { dlItemEnd }
1104 { 0 }
1105 \ExplSyntaxOff

```

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

```

1106 \def\markdownRendererDlDefinitionBegin{%
1107 \markdownRendererDlDefinitionBeginPrototype}%
1108 \ExplSyntaxOn
1109 \seq_gput_right:Nn
1110 \g_@@_renderers_seq
1111 { dlDefinitionBegin }
1112 \prop_gput:Nnn
1113 \g_@@_renderer_arities_prop
1114 { dlDefinitionBegin }
1115 { 0 }
1116 \ExplSyntaxOff

```

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

```

1117 \def\markdownRendererDlDefinitionEnd{%
1118 \markdownRendererDlDefinitionEndPrototype}%
1119 \ExplSyntaxOn
1120 \seq_gput_right:Nn
1121 \g_@@_renderers_seq
1122 { dlDefinitionEnd }

```

```

1123 \prop_gput:Nnn
1124 \g_@@_renderer_arities_prop
1125 { dlDefinitionEnd }
1126 { 0 }
1127 \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.

```

1128 \def\markdownRendererDlEnd{%
1129 \markdownRendererDlEndPrototype}%
1130 \ExplSyntaxOn
1131 \seq_gput_right:Nn
1132 \g_@@_renderers_seq
1133 { dlEnd }
1134 \prop_gput:Nnn
1135 \g_@@_renderer_arities_prop
1136 { dlEnd }
1137 { 0 }
1138 \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.

```

1139 \def\markdownRendererDlEndTight{%
1140 \markdownRendererDlEndTightPrototype}%
1141 \ExplSyntaxOn
1142 \seq_gput_right:Nn
1143 \g_@@_renderers_seq
1144 { dlEndTight }
1145 \prop_gput:Nnn
1146 \g_@@_renderer_arities_prop
1147 { dlEndTight }
1148 { 0 }
1149 \ExplSyntaxOff

```

**2.2.3.9 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.

```

1150 \def\markdownRendererEllipsis{%
1151 \markdownRendererEllipsisPrototype}%
1152 \ExplSyntaxOn
1153 \seq_gput_right:Nn
1154 \g_@@_renderers_seq

```

```

1155 { ellipsis }
1156 \prop_gput:Nnn
1157 \g_@@_renderer_arities_prop
1158 { ellipsis }
1159 { 0 }
1160 \ExplSyntaxOff

```

**2.2.3.10 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.

```

1161 \def\markdownRendererEmphasis{%
1162 \markdownRendererEmphasisPrototype}%
1163 \ExplSyntaxOn
1164 \seq_gput_right:Nn
1165 \g_@@_renderers_seq
1166 { emphasis }
1167 \prop_gput:Nnn
1168 \g_@@_renderer_arities_prop
1169 { emphasis }
1170 { 1 }
1171 \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.

```

1172 \def\markdownRendererStrongEmphasis{%
1173 \markdownRendererStrongEmphasisPrototype}%
1174 \ExplSyntaxOn
1175 \seq_gput_right:Nn
1176 \g_@@_renderers_seq
1177 { strongEmphasis }
1178 \prop_gput:Nnn
1179 \g_@@_renderer_arities_prop
1180 { strongEmphasis }
1181 { 1 }
1182 \ExplSyntaxOff

```

**2.2.3.11 Fenced Div 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 div in which the attributes of the div apply. The macros receive no arguments.

```

1183 \def\markdownRendererFencedDivAttributeContextBegin{%
1184 \markdownRendererFencedDivAttributeContextBeginPrototype}%

```

```

1185 \ExplSyntaxOn
1186 \seq_gput_right:Nn
1187 \g_@@_renderers_seq
1188 { fencedDivAttributeContextBegin }
1189 \prop_gput:Nnn
1190 \g_@@_renderer_arities_prop
1191 { fencedDivAttributeContextBegin }
1192 { 0 }
1193 \ExplSyntaxOff
1194 \def\markdownRendererFencedDivAttributeContextEnd{%
1195 \markdownRendererFencedDivAttributeContextEndPrototype}%
1196 \ExplSyntaxOn
1197 \seq_gput_right:Nn
1198 \g_@@_renderers_seq
1199 { fencedDivAttributeContextEnd }
1200 \prop_gput:Nnn
1201 \g_@@_renderer_arities_prop
1202 { fencedDivAttributeContextEnd }
1203 { 0 }
1204 \ExplSyntaxOff

```

**2.2.3.12 Header Attribute Context Renderers** The following macros are only produced, when the `headerAttributes` option is enabled.

The `\markdownRendererHeaderAttributeContextBegin` and `\markdownRendererHeaderAttributeContextEnd` macros represent the beginning and the end of a section in which the attributes of a heading apply. The macros receive no arguments.

```

1205 \def\markdownRendererHeaderAttributeContextBegin{%
1206 \markdownRendererHeaderAttributeContextBeginPrototype}%
1207 \ExplSyntaxOn
1208 \seq_gput_right:Nn
1209 \g_@@_renderers_seq
1210 { headerAttributeContextBegin }
1211 \prop_gput:Nnn
1212 \g_@@_renderer_arities_prop
1213 { headerAttributeContextBegin }
1214 { 0 }
1215 \ExplSyntaxOff
1216 \def\markdownRendererHeaderAttributeContextEnd{%
1217 \markdownRendererHeaderAttributeContextEndPrototype}%
1218 \ExplSyntaxOn
1219 \seq_gput_right:Nn
1220 \g_@@_renderers_seq
1221 { headerAttributeContextEnd }
1222 \prop_gput:Nnn
1223 \g_@@_renderer_arities_prop
1224 { headerAttributeContextEnd }

```

```
1225 { 0 }
1226 \ExplSyntaxOff
```

**2.2.3.13 Heading Renderers** The `\markdownRendererHeadingOne` macro represents a first level heading. The macro receives a single argument that corresponds to the heading text.

```
1227 \def\markdownRendererHeadingOne{%
1228 \markdownRendererHeadingOnePrototype}%
1229 \ExplSyntaxOn
1230 \seq_gput_right:Nn
1231 \g_@@_renderers_seq
1232 { headingOne }
1233 \prop_gput:Nnn
1234 \g_@@_renderer_arities_prop
1235 { headingOne }
1236 { 1 }
1237 \ExplSyntaxOff
```

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

```
1238 \def\markdownRendererHeadingTwo{%
1239 \markdownRendererHeadingTwoPrototype}%
1240 \ExplSyntaxOn
1241 \seq_gput_right:Nn
1242 \g_@@_renderers_seq
1243 { headingTwo }
1244 \prop_gput:Nnn
1245 \g_@@_renderer_arities_prop
1246 { headingTwo }
1247 { 1 }
1248 \ExplSyntaxOff
```

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

```
1249 \def\markdownRendererHeadingThree{%
1250 \markdownRendererHeadingThreePrototype}%
1251 \ExplSyntaxOn
1252 \seq_gput_right:Nn
1253 \g_@@_renderers_seq
1254 { headingThree }
1255 \prop_gput:Nnn
1256 \g_@@_renderer_arities_prop
1257 { headingThree }
1258 { 1 }
1259 \ExplSyntaxOff
```

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

```
1260 \def\markdownRendererHeadingFour{%
1261 \markdownRendererHeadingFourPrototype}%
1262 \ExplSyntaxOn
1263 \seq_gput_right:Nn
1264 \g_@@_renderers_seq
1265 { headingFour }
1266 \prop_gput:Nnn
1267 \g_@@_renderer_arities_prop
1268 { headingFour }
1269 { 1 }
1270 \ExplSyntaxOff
```

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

```
1271 \def\markdownRendererHeadingFive{%
1272 \markdownRendererHeadingFivePrototype}%
1273 \ExplSyntaxOn
1274 \seq_gput_right:Nn
1275 \g_@@_renderers_seq
1276 { headingFive }
1277 \prop_gput:Nnn
1278 \g_@@_renderer_arities_prop
1279 { headingFive }
1280 { 1 }
1281 \ExplSyntaxOff
```

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

```
1282 \def\markdownRendererHeadingSix{%
1283 \markdownRendererHeadingSixPrototype}%
1284 \ExplSyntaxOn
1285 \seq_gput_right:Nn
1286 \g_@@_renderers_seq
1287 { headingSix }
1288 \prop_gput:Nnn
1289 \g_@@_renderer_arities_prop
1290 { headingSix }
1291 { 1 }
1292 \ExplSyntaxOff
```

**2.2.3.14 HTML Comment Renderers** 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.

The `\markdownRendererBlockHtmlCommentBegin` and `\markdownRendererBlockHtmlCommentEnd` macros represent the beginning and the end of a block HTML comment. The macros receive no arguments.

```

1293 \def\markdownRendererInlineHtmlComment{%
1294 \markdownRendererInlineHtmlCommentPrototype}%
1295 \ExplSyntaxOn
1296 \seq_gput_right:Nn
1297 \g_@@_renderers_seq
1298 { inlineHtmlComment }
1299 \prop_gput:Nnn
1300 \g_@@_renderer_arities_prop
1301 { inlineHtmlComment }
1302 { 1 }
1303 \ExplSyntaxOff
1304 \def\markdownRendererBlockHtmlCommentBegin{%
1305 \markdownRendererBlockHtmlCommentBeginPrototype}%
1306 \ExplSyntaxOn
1307 \seq_gput_right:Nn
1308 \g_@@_renderers_seq
1309 { blockHtmlCommentBegin }
1310 \prop_gput:Nnn
1311 \g_@@_renderer_arities_prop
1312 { blockHtmlCommentBegin }
1313 { 0 }
1314 \ExplSyntaxOff
1315 \def\markdownRendererBlockHtmlCommentEnd{%
1316 \markdownRendererBlockHtmlCommentEndPrototype}%
1317 \ExplSyntaxOn
1318 \seq_gput_right:Nn
1319 \g_@@_renderers_seq
1320 { blockHtmlCommentEnd }
1321 \prop_gput:Nnn
1322 \g_@@_renderer_arities_prop
1323 { blockHtmlCommentEnd }
1324 { 0 }
1325 \ExplSyntaxOff

```

**2.2.3.15 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.

```

1326 \def\markdownRendererInlineHtmlTag{%
1327 \markdownRendererInlineHtmlTagPrototype}%
1328 \ExplSyntaxOn
1329 \seq_gput_right:Nn
1330 \g_@@_renderers_seq
1331 { inlineHtmlTag }
1332 \prop_gput:Nnn
1333 \g_@@_renderer_arities_prop
1334 { inlineHtmlTag }
1335 { 1 }
1336 \ExplSyntaxOff
1337 \def\markdownRendererInputBlockHtmlElement{%
1338 \markdownRendererInputBlockHtmlElementPrototype}%
1339 \ExplSyntaxOn
1340 \seq_gput_right:Nn
1341 \g_@@_renderers_seq
1342 { inputBlockHtmlElement }
1343 \prop_gput:Nnn
1344 \g_@@_renderer_arities_prop
1345 { inputBlockHtmlElement }
1346 { 1 }
1347 \ExplSyntaxOff

```

**2.2.3.16 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.

```

1348 \def\markdownRendererImage{%
1349 \markdownRendererImagePrototype}%
1350 \ExplSyntaxOn
1351 \seq_gput_right:Nn
1352 \g_@@_renderers_seq
1353 { image }
1354 \prop_gput:Nnn
1355 \g_@@_renderer_arities_prop
1356 { image }
1357 { 4 }
1358 \ExplSyntaxOff

```

**2.2.3.17 Interblock Separator Renderer** The `\markdownRendererInterblockSeparator` macro represents a separator between two markdown block elements. The macro receives no arguments.

```

1359 \def\markdownRendererInterblockSeparator{%
1360 \markdownRendererInterblockSeparatorPrototype}%
1361 \ExplSyntaxOn

```



```

1362 \seq_gput_right:Nn
1363 \g_@@_renderers_seq
1364 { interblockSeparator }
1365 \prop_gput:Nnn
1366 \g_@@_render_aritys_prop
1367 { interblockSeparator }
1368 { 0 }
1369 \ExplSyntaxOff

```

**2.2.3.18 Line Break Renderer** The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

```

1370 \def\markdownRendererLineBreak{%
1371 \markdownRendererLineBreakPrototype}%
1372 \ExplSyntaxOn
1373 \seq_gput_right:Nn
1374 \g_@@_renderers_seq
1375 { lineBreak }
1376 \prop_gput:Nnn
1377 \g_@@_render_aritys_prop
1378 { lineBreak }
1379 { 0 }
1380 \ExplSyntaxOff

```

**2.2.3.19 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.

```

1381 \def\markdownRendererLink{%
1382 \markdownRendererLinkPrototype}%
1383 \ExplSyntaxOn
1384 \seq_gput_right:Nn
1385 \g_@@_renderers_seq
1386 { link }
1387 \prop_gput:Nnn
1388 \g_@@_render_aritys_prop
1389 { link }
1390 { 4 }
1391 \ExplSyntaxOff

```

**2.2.3.20 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.

```
1392 \def\markdownRendererDocumentBegin{%
1393 \markdownRendererDocumentBeginPrototype}%
1394 \ExplSyntaxOn
1395 \seq_gput_right:Nn
1396 \g_@@_renderers_seq
1397 { documentBegin }
1398 \prop_gput:Nnn
1399 \g_@@_renderer_arities_prop
1400 { documentBegin }
1401 { 0 }
1402 \ExplSyntaxOff
1403 \def\markdownRendererDocumentEnd{%
1404 \markdownRendererDocumentEndPrototype}%
1405 \ExplSyntaxOn
1406 \seq_gput_right:Nn
1407 \g_@@_renderers_seq
1408 { documentEnd }
1409 \prop_gput:Nnn
1410 \g_@@_renderer_arities_prop
1411 { documentEnd }
1412 { 0 }
1413 \ExplSyntaxOff
```

**2.2.3.21 Non-Breaking Space Renderer** The `\markdownRendererNbsp` macro represents a non-breaking space.

```
1414 \def\markdownRendererNbsp{%
1415 \markdownRendererNbspPrototype}%
1416 \ExplSyntaxOn
1417 \seq_gput_right:Nn
1418 \g_@@_renderers_seq
1419 { nbsp }
1420 \prop_gput:Nnn
1421 \g_@@_renderer_arities_prop
1422 { nbsp }
1423 { 0 }
1424 \ExplSyntaxOff
```

**2.2.3.22 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.

The `\markdownRendererFootnote` and `\markdownRendererFootnotePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

1425 \ExplSyntaxOn
1426 \cs_new:Npn
1427 \markdownRendererNote
1428 {
1429 \cs_if_exist:NTF
1430 \markdownRendererFootnote
1431 {
1432 \markdownWarning
1433 {
1434 Footnote~renderer~has~been~deprecated,~
1435 to~be~removed~in~Markdown~3.0.0
1436 }
1437 \markdownRendererFootnote
1438 }
1439 {
1440 \cs_if_exist:NTF
1441 \markdownRendererFootnotePrototype
1442 {
1443 \markdownWarning
1444 {
1445 Footnote~renderer~prototype~has~been~deprecated,~
1446 to~be~removed~in~Markdown~3.0.0
1447 }
1448 \markdownRendererFootnotePrototype
1449 }
1450 {
1451 \markdownRendererNotePrototype
1452 }
1453 }
1454 }
1455 \seq_gput_right:Nn
1456 \g_@@_renderers_seq
1457 { footnote }
1458 \prop_gput:Nnn
1459 \g_@@_renderer_arities_prop
1460 { footnote }
1461 { 1 }
1462 \seq_gput_right:Nn
1463 \g_@@_renderers_seq
1464 { note }
1465 \prop_gput:Nnn
1466 \g_@@_renderer_arities_prop
1467 { note }
1468 { 1 }
1469 \ExplSyntaxOff

```

**2.2.3.23 Ordered List Renderers** The `\markdownRenderer01Begin` macro repre-

sents 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.

```

1470 \def\markdownRendererOlBegin{%
1471 \markdownRendererOlBeginPrototype}%
1472 \ExplSyntaxOn
1473 \seq_gput_right:Nn
1474 \g_@@_renderers_seq
1475 { olBegin }
1476 \prop_gput:Nnn
1477 \g_@@_renderer_arities_prop
1478 { olBegin }
1479 { 0 }
1480 \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.

```

1481 \def\markdownRendererOlBeginTight{%
1482 \markdownRendererOlBeginTightPrototype}%
1483 \ExplSyntaxOn
1484 \seq_gput_right:Nn
1485 \g_@@_renderers_seq
1486 { olBeginTight }
1487 \prop_gput:Nnn
1488 \g_@@_renderer_arities_prop
1489 { olBeginTight }
1490 { 0 }
1491 \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`).

```

1492 \def\markdownRendererFancyOlBegin{%
1493 \markdownRendererFancyOlBeginPrototype}%
1494 \ExplSyntaxOn
1495 \seq_gput_right:Nn
1496 \g_@@_renderers_seq
1497 { fancyOlBegin }
1498 \prop_gput:Nnn
1499 \g_@@_renderer_arities_prop

```

```

1500 { fancyO1Begin }
1501 { 2 }
1502 \ExplSyntaxOff

```

The `\markdownRendererFancyO1BeginTight` 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 `\markdownRendererFancyO1Begin` macro for the valid style values.

```

1503 \def\markdownRendererFancyO1BeginTight{%
1504 \markdownRendererFancyO1BeginTightPrototype}%
1505 \ExplSyntaxOn
1506 \seq_gput_right:Nn
1507 \g_@@_renderers_seq
1508 { fancyO1BeginTight }
1509 \prop_gput:Nnn
1510 \g_@@_renderer_arities_prop
1511 { fancyO1BeginTight }
1512 { 2 }
1513 \ExplSyntaxOff

```

The `\markdownRendererO1Item` 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.

```

1514 \def\markdownRendererO1Item{%
1515 \markdownRendererO1ItemPrototype}%
1516 \ExplSyntaxOn
1517 \seq_gput_right:Nn
1518 \g_@@_renderers_seq
1519 { olItem }
1520 \prop_gput:Nnn
1521 \g_@@_renderer_arities_prop
1522 { olItem }
1523 { 0 }
1524 \ExplSyntaxOff

```

The `\markdownRendererO1ItemEnd` 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.

```

1525 \def\markdownRendererO1ItemEnd{%
1526 \markdownRendererO1ItemEndPrototype}%
1527 \ExplSyntaxOn
1528 \seq_gput_right:Nn
1529 \g_@@_renderers_seq
1530 { olItemEnd }

```

```

1531 \prop_gput:Nnn
1532 \g_@@_renderer_arities_prop
1533 { olItemEnd }
1534 { 0 }
1535 \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.

```

1536 \def\markdownRendererOlItemWithNumber{%
1537 \markdownRendererOlItemWithNumberPrototype}%
1538 \ExplSyntaxOn
1539 \seq_gput_right:Nn
1540 \g_@@_renderers_seq
1541 { olItemWithNumber }
1542 \prop_gput:Nnn
1543 \g_@@_renderer_arities_prop
1544 { olItemWithNumber }
1545 { 1 }
1546 \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.

```

1547 \def\markdownRendererFancyOlItem{%
1548 \markdownRendererFancyOlItemPrototype}%
1549 \ExplSyntaxOn
1550 \seq_gput_right:Nn
1551 \g_@@_renderers_seq
1552 { fancyOlItem }
1553 \prop_gput:Nnn
1554 \g_@@_renderer_arities_prop
1555 { fancyOlItem }
1556 { 0 }
1557 \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.

```

1558 \def\markdownRendererFancyOlItemEnd{%
1559 \markdownRendererFancyOlItemEndPrototype}%
1560 \ExplSyntaxOn
1561 \seq_gput_right:Nn
1562 \g_@@_renderers_seq
1563 { fancyOlItemEnd }

```

```

1564 \prop_gput:Nnn
1565 \g_@@_renderer_arities_prop
1566 { fancyO1ItemEnd }
1567 { 0 }
1568 \ExplSyntaxOff

```

The `\markdownRendererFancyO1ItemWithNumber` 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.

```

1569 \def\markdownRendererFancyO1ItemWithNumber{%
1570 \markdownRendererFancyO1ItemWithNumberPrototype}%
1571 \ExplSyntaxOn
1572 \seq_gput_right:Nn
1573 \g_@@_renderers_seq
1574 { fancyO1ItemWithNumber }
1575 \prop_gput:Nnn
1576 \g_@@_renderer_arities_prop
1577 { fancyO1ItemWithNumber }
1578 { 1 }
1579 \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.

```

1580 \def\markdownRendererO1End{%
1581 \markdownRendererO1EndPrototype}%
1582 \ExplSyntaxOn
1583 \seq_gput_right:Nn
1584 \g_@@_renderers_seq
1585 { o1End }
1586 \prop_gput:Nnn
1587 \g_@@_renderer_arities_prop
1588 { o1End }
1589 { 0 }
1590 \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.

```

1591 \def\markdownRendererO1EndTight{%
1592 \markdownRendererO1EndTightPrototype}%
1593 \ExplSyntaxOn
1594 \seq_gput_right:Nn

```

```

1595 \g_@@_renderers_seq
1596 { olEndTight }
1597 \prop_gput:Nnn
1598 \g_@@_renderer_arities_prop
1599 { olEndTight }
1600 { 0 }
1601 \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.

```

1602 \def\markdownRendererFancyO1End{%
1603 \markdownRendererFancyO1EndPrototype}%
1604 \ExplSyntaxOn
1605 \seq_gput_right:Nn
1606 \g_@@_renderers_seq
1607 { fancyO1End }
1608 \prop_gput:Nnn
1609 \g_@@_renderer_arities_prop
1610 { fancyO1End }
1611 { 0 }
1612 \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.

```

1613 \def\markdownRendererFancyO1EndTight{%
1614 \markdownRendererFancyO1EndTightPrototype}%
1615 \ExplSyntaxOn
1616 \seq_gput_right:Nn
1617 \g_@@_renderers_seq
1618 { fancyO1EndTight }
1619 \prop_gput:Nnn
1620 \g_@@_renderer_arities_prop
1621 { fancyO1EndTight }
1622 { 0 }
1623 \ExplSyntaxOff

```

**2.2.3.24 Parenthesized Citations Renderer** 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.

```
1624 \def\markdownRendererCite{%
1625 \markdownRendererCitePrototype}%
1626 \ExplSyntaxOn
1627 \seq_gput_right:Nn
1628 \g_@@_renderers_seq
1629 { cite }
1630 \prop_gput:Nnn
1631 \g_@@_renderer_arities_prop
1632 { cite }
1633 { 1 }
1634 \ExplSyntaxOff
```

**2.2.3.25 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.

```
1635 \def\markdownRendererInputRawInline{%
1636 \markdownRendererInputRawInlinePrototype}%
1637 \ExplSyntaxOn
1638 \seq_gput_right:Nn
1639 \g_@@_renderers_seq
1640 { inputRawInline }
1641 \prop_gput:Nnn
1642 \g_@@_renderer_arities_prop
1643 { inputRawInline }
1644 { 2 }
1645 \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.

```
1646 \def\markdownRendererInputRawBlock{%
1647 \markdownRendererInputRawBlockPrototype}%
1648 \ExplSyntaxOn
1649 \seq_gput_right:Nn
1650 \g_@@_renderers_seq
1651 { inputRawBlock }
1652 \prop_gput:Nnn
1653 \g_@@_renderer_arities_prop
1654 { inputRawBlock }
1655 { 2 }
```

```
1656 \ExplSyntaxOff
```

**2.2.3.26 Special Character Renderers** The following macros replace any special plain T<sub>E</sub>X characters, including the active pipe character (|) of ConT<sub>E</sub>Xt, in the input text. These macros will only be produced, when the `hybrid` option is `false`.

```
1657 \def\markdownRendererLeftBrace{%
1658 \markdownRendererLeftBracePrototype}%
1659 \ExplSyntaxOn
1660 \seq_gput_right:Nn
1661 \g_@@_renderers_seq
1662 { leftBrace }
1663 \prop_gput:Nnn
1664 \g_@@_renderer_arities_prop
1665 { leftBrace }
1666 { 0 }
1667 \ExplSyntaxOff
1668 \def\markdownRendererRightBrace{%
1669 \markdownRendererRightBracePrototype}%
1670 \ExplSyntaxOn
1671 \seq_gput_right:Nn
1672 \g_@@_renderers_seq
1673 { rightBrace }
1674 \prop_gput:Nnn
1675 \g_@@_renderer_arities_prop
1676 { rightBrace }
1677 { 0 }
1678 \ExplSyntaxOff
1679 \def\markdownRendererDollarSign{%
1680 \markdownRendererDollarSignPrototype}%
1681 \ExplSyntaxOn
1682 \seq_gput_right:Nn
1683 \g_@@_renderers_seq
1684 { dollarSign }
1685 \prop_gput:Nnn
1686 \g_@@_renderer_arities_prop
1687 { dollarSign }
1688 { 0 }
1689 \ExplSyntaxOff
1690 \def\markdownRendererPercentSign{%
1691 \markdownRendererPercentSignPrototype}%
1692 \ExplSyntaxOn
1693 \seq_gput_right:Nn
1694 \g_@@_renderers_seq
1695 { percentSign }
1696 \prop_gput:Nnn
1697 \g_@@_renderer_arities_prop
```

```

1698 { percentSign }
1699 { 0 }
1700 \ExplSyntaxOff
1701 \def\markdownRendererAmpersand{%
1702 \markdownRendererAmpersandPrototype}%
1703 \ExplSyntaxOn
1704 \seq_gput_right:Nn
1705 \g_@@_renderers_seq
1706 { ampersand }
1707 \prop_gput:Nnn
1708 \g_@@_renderer_arities_prop
1709 { ampersand }
1710 { 0 }
1711 \ExplSyntaxOff
1712 \def\markdownRendererUnderscore{%
1713 \markdownRendererUnderscorePrototype}%
1714 \ExplSyntaxOn
1715 \seq_gput_right:Nn
1716 \g_@@_renderers_seq
1717 { underscore }
1718 \prop_gput:Nnn
1719 \g_@@_renderer_arities_prop
1720 { underscore }
1721 { 0 }
1722 \ExplSyntaxOff
1723 \def\markdownRendererHash{%
1724 \markdownRendererHashPrototype}%
1725 \ExplSyntaxOn
1726 \seq_gput_right:Nn
1727 \g_@@_renderers_seq
1728 { hash }
1729 \prop_gput:Nnn
1730 \g_@@_renderer_arities_prop
1731 { hash }
1732 { 0 }
1733 \ExplSyntaxOff
1734 \def\markdownRendererCircumflex{%
1735 \markdownRendererCircumflexPrototype}%
1736 \ExplSyntaxOn
1737 \seq_gput_right:Nn
1738 \g_@@_renderers_seq
1739 { circumflex }
1740 \prop_gput:Nnn
1741 \g_@@_renderer_arities_prop
1742 { circumflex }
1743 { 0 }
1744 \ExplSyntaxOff

```

```

1745 \def\markdownRendererBackslash{%
1746 \markdownRendererBackslashPrototype}%
1747 \ExplSyntaxOn
1748 \seq_gput_right:Nn
1749 \g_@@_renderers_seq
1750 { backslash }
1751 \prop_gput:Nnn
1752 \g_@@_renderer_arities_prop
1753 { backslash }
1754 { 0 }
1755 \ExplSyntaxOff
1756 \def\markdownRendererTilde{%
1757 \markdownRendererTildePrototype}%
1758 \ExplSyntaxOn
1759 \seq_gput_right:Nn
1760 \g_@@_renderers_seq
1761 { tilde }
1762 \prop_gput:Nnn
1763 \g_@@_renderer_arities_prop
1764 { tilde }
1765 { 0 }
1766 \ExplSyntaxOff
1767 \def\markdownRendererPipe{%
1768 \markdownRendererPipePrototype}%
1769 \ExplSyntaxOn
1770 \seq_gput_right:Nn
1771 \g_@@_renderers_seq
1772 { pipe }
1773 \prop_gput:Nnn
1774 \g_@@_renderer_arities_prop
1775 { pipe }
1776 { 0 }
1777 \ExplSyntaxOff

```

**2.2.3.27 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.

```

1778 \def\markdownRendererStrikeThrough{%
1779 \markdownRendererStrikeThroughPrototype}%
1780 \ExplSyntaxOn
1781 \seq_gput_right:Nn
1782 \g_@@_renderers_seq
1783 { strikeThrough }
1784 \prop_gput:Nnn
1785 \g_@@_renderer_arities_prop

```

```

1786 { strikeThrough }
1787 { 1 }
1788 \ExplSyntaxOff

```

**2.2.3.28 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.

```

1789 \def\markdownRendererSubscript{%
1790 \markdownRendererSubscriptPrototype}%
1791 \ExplSyntaxOn
1792 \seq_gput_right:Nn
1793 \g_@@_renderers_seq
1794 { subscript }
1795 \prop_gput:Nnn
1796 \g_@@_renderer_arities_prop
1797 { subscript }
1798 { 1 }

```

**2.2.3.29 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.

```

1799 \def\markdownRendererSuperscript{%
1800 \markdownRendererSuperscriptPrototype}%
1801 \ExplSyntaxOn
1802 \seq_gput_right:Nn
1803 \g_@@_renderers_seq
1804 { superscript }
1805 \prop_gput:Nnn
1806 \g_@@_renderer_arities_prop
1807 { superscript }
1808 { 1 }
1809 \ExplSyntaxOff

```

**2.2.3.30 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.

```

1810 \def\markdownRendererTable{%
1811 \markdownRendererTablePrototype}%
1812 \ExplSyntaxOn
1813 \seq_gput_right:Nn
1814 \g_@@_renderers_seq
1815 { table }
1816 \prop_gput:Nnn
1817 \g_@@_renderer_arities_prop
1818 { table }
1819 { 3 }
1820 \ExplSyntaxOff

```

**2.2.3.31 Text Citations Renderer** 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.

```

1821 \def\markdownRendererTextCite{%
1822 \markdownRendererTextCitePrototype}%
1823 \ExplSyntaxOn
1824 \seq_gput_right:Nn
1825 \g_@@_renderers_seq
1826 { textCite }
1827 \prop_gput:Nnn
1828 \g_@@_renderer_arities_prop
1829 { textCite }
1830 { 1 }
1831 \ExplSyntaxOff

```

**2.2.3.32 Thematic Break Renderer** The `\markdownRendererThematicBreak` macro represents a thematic break. The macro receives no arguments.

The `\markdownRendererHorizontalRule` and `\markdownRendererHorizontalRulePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

1832 \ExplSyntaxOn
1833 \cs_new:Npn
1834 \markdownRendererThematicBreak
1835 {
1836 \cs_if_exist:NTF
1837 \markdownRendererHorizontalRule
1838 {
1839 \markdownWarning
1840 {

```

```

1841 Horizontal~rule~renderer~has~been~deprecated,~
1842 to~be~removed~in~Markdown~3.0.0
1843 }
1844 \markdownRendererHorizontalRule
1845 }
1846 {
1847 \cs_if_exist:NTF
1848 \markdownRendererHorizontalRulePrototype
1849 {
1850 \markdownWarning
1851 {
1852 Horizontal~rule~renderer~prototype~has~been~deprecated,~
1853 to~be~removed~in~Markdown~3.0.0
1854 }
1855 \markdownRendererHorizontalRulePrototype
1856 }
1857 {
1858 \markdownRendererThematicBreakPrototype
1859 }
1860 }
1861 }
1862 \seq_gput_right:Nn
1863 \g_@@_renderers_seq
1864 { horizontalRule }
1865 \prop_gput:Nnn
1866 \g_@@_renderer_arities_prop
1867 { horizontalRule }
1868 { 0 }
1869 \seq_gput_right:Nn
1870 \g_@@_renderers_seq
1871 { thematicBreak }
1872 \prop_gput:Nnn
1873 \g_@@_renderer_arities_prop
1874 { thematicBreak }
1875 { 0 }
1876 \ExplSyntaxOff

```

**2.2.3.33 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.

```

1877 \def\markdownRendererTickedBox{%
1878 \markdownRendererTickedBoxPrototype}%
1879 \ExplSyntaxOn

```

```

1880 \seq_gput_right:Nn
1881 \g_@@_renderers_seq
1882 { tickedBox }
1883 \prop_gput:Nnn
1884 \g_@@_renderer_arities_prop
1885 { tickedBox }
1886 { 0 }
1887 \ExplSyntaxOff
1888 \def\markdownRendererHalfTickedBox{%
1889 \markdownRendererHalfTickedBoxPrototype}%
1890 \ExplSyntaxOn
1891 \seq_gput_right:Nn
1892 \g_@@_renderers_seq
1893 { halfTickedBox }
1894 \prop_gput:Nnn
1895 \g_@@_renderer_arities_prop
1896 { halfTickedBox }
1897 { 0 }
1898 \ExplSyntaxOff
1899 \def\markdownRendererUntickedBox{%
1900 \markdownRendererUntickedBoxPrototype}%
1901 \ExplSyntaxOn
1902 \seq_gput_right:Nn
1903 \g_@@_renderers_seq
1904 { untickedBox }
1905 \prop_gput:Nnn
1906 \g_@@_renderer_arities_prop
1907 { untickedBox }
1908 { 0 }
1909 \ExplSyntaxOff

```

**2.2.3.34 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.

```

1910 \def\markdownRendererJekyllDataBegin{%
1911 \markdownRendererJekyllDataBeginPrototype}%
1912 \ExplSyntaxOn
1913 \seq_gput_right:Nn
1914 \g_@@_renderers_seq
1915 { jekyllDataBegin }
1916 \prop_gput:Nnn
1917 \g_@@_renderer_arities_prop
1918 { jekyllDataBegin }
1919 { 0 }
1920 \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.

```
1921 \def\markdownRendererJekyllDataEnd{%
1922 \markdownRendererJekyllDataEndPrototype}%
1923 \ExplSyntaxOn
1924 \seq_gput_right:Nn
1925 \g_@@_renderers_seq
1926 { jekyllDataEnd }
1927 \prop_gput:Nnn
1928 \g_@@_renderer_arities_prop
1929 { jekyllDataEnd }
1930 { 0 }
1931 \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.

```
1932 \def\markdownRendererJekyllDataMappingBegin{%
1933 \markdownRendererJekyllDataMappingBeginPrototype}%
1934 \ExplSyntaxOn
1935 \seq_gput_right:Nn
1936 \g_@@_renderers_seq
1937 { jekyllDataMappingBegin }
1938 \prop_gput:Nnn
1939 \g_@@_renderer_arities_prop
1940 { jekyllDataMappingBegin }
1941 { 2 }
1942 \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.

```
1943 \def\markdownRendererJekyllDataMappingEnd{%
1944 \markdownRendererJekyllDataMappingEndPrototype}%
1945 \ExplSyntaxOn
1946 \seq_gput_right:Nn
1947 \g_@@_renderers_seq
1948 { jekyllDataMappingEnd }
1949 \prop_gput:Nnn
1950 \g_@@_renderer_arities_prop
1951 { jekyllDataMappingEnd }
1952 { 0 }
1953 \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.

```

1954 \def\markdownRendererJekyllDataSequenceBegin{%
1955 \markdownRendererJekyllDataSequenceBeginPrototype}%
1956 \ExplSyntaxOn
1957 \seq_gput_right:Nn
1958 \g_@@_renderers_seq
1959 { jekyllDataSequenceBegin }
1960 \prop_gput:Nnn
1961 \g_@@_renderer_arities_prop
1962 { jekyllDataSequenceBegin }
1963 { 2 }
1964 \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.

```

1965 \def\markdownRendererJekyllDataSequenceEnd{%
1966 \markdownRendererJekyllDataSequenceEndPrototype}%
1967 \ExplSyntaxOn
1968 \seq_gput_right:Nn
1969 \g_@@_renderers_seq
1970 { jekyllDataSequenceEnd }
1971 \prop_gput:Nnn
1972 \g_@@_renderer_arities_prop
1973 { jekyllDataSequenceEnd }
1974 { 0 }
1975 \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.

```

1976 \def\markdownRendererJekyllDataBoolean{%
1977 \markdownRendererJekyllDataBooleanPrototype}%
1978 \ExplSyntaxOn
1979 \seq_gput_right:Nn
1980 \g_@@_renderers_seq
1981 { jekyllDataBoolean }
1982 \prop_gput:Nnn
1983 \g_@@_renderer_arities_prop

```

```

1984 { jekyllDataBoolean }
1985 { 2 }
1986 \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.

```

1987 \def\markdownRendererJekyllDataNumber{%
1988 \markdownRendererJekyllDataNumberPrototype}%
1989 \ExplSyntaxOn
1990 \seq_gput_right:Nn
1991 \g_@@_renderers_seq
1992 { jekyllDataNumber }
1993 \prop_gput:Nnn
1994 \g_@@_renderer_arities_prop
1995 { jekyllDataNumber }
1996 { 2 }
1997 \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.

```

1998 \def\markdownRendererJekyllDataString{%
1999 \markdownRendererJekyllDataStringPrototype}%
2000 \ExplSyntaxOn
2001 \seq_gput_right:Nn
2002 \g_@@_renderers_seq
2003 { jekyllDataString }
2004 \prop_gput:Nnn
2005 \g_@@_renderer_arities_prop
2006 { jekyllDataString }
2007 { 2 }
2008 \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 2.2.4.1 for the description of the high-level `expl3` interface that you can also use to react to YAML metadata.

```

2009 \def\markdownRendererJekyllDataEmpty{%
2010 \markdownRendererJekyllDataEmptyPrototype}%

```

```

2011 \ExplSyntaxOn
2012 \seq_gput_right:Nn
2013 \g_@@_renderers_seq
2014 { jekyllDataEmpty }
2015 \prop_gput:Nnn
2016 \g_@@_renderer_arities_prop
2017 { jekyllDataEmpty }
2018 { 1 }
2019 \ExplSyntaxOff

```

## 2.2.4 Token Renderer Prototypes

**2.2.4.1 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.

```

2020 \ExplSyntaxOn
2021 \keys_define:nn
2022 { markdown/jekyllData }
2023 { }
2024 \ExplSyntaxOff

```

The following  $\TeX$  macros provide definitions for the token renderers (see Section 2.2.3) that have not been redefined by the user. These macros are intended to be redefined by macro package authors who wish to provide sensible default token renderers. They are also redefined by the  $\LaTeX$  and  $\ConTeXt$  implementations (see sections 3.3 and 3.4).

```

2025 \ExplSyntaxOn
2026 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
2027 {
2028 \seq_map_function:NN
2029 \g_@@_renderers_seq
2030 \@@_plaintex_define_renderer_prototype:n
2031 \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
2032 \let\markdownRendererBlockHtmlCommentBegin=\iffalse
2033 \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
2034 \let\markdownRendererBlockHtmlCommentEnd=\fi

```

The `\markdownRendererFootnote` and `\markdownRendererFootnotePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

2035 \cs_undefine:N \markdownRendererFootnote
2036 \cs_undefine:N \markdownRendererFootnotePrototype

```

The `\markdownRendererHorizontalRule` and `\markdownRendererHorizontalRulePrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```

2037 \cs_undefine:N \markdownRendererHorizontalRule
2038 \cs_undefine:N \markdownRendererHorizontalRulePrototype
2039 }

```

```

2040 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
2041 {
2042 \@@_renderer_prototype_tl_to_csname:nN
2043 { #1 }
2044 \l_tmpa_tl
2045 \prop_get:NnN
2046 \g_@@_renderer_arities_prop
2047 { #1 }
2048 \l_tmpb_tl
2049 \@@_plaintex_define_renderer_prototype:cV
2050 { \l_tmpa_tl }
2051 \l_tmpb_tl
2052 }
2053 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
2054 {
2055 \tl_set:Nn
2056 \l_tmpa_tl
2057 { \str_uppercase:n { #1 } }
2058 \tl_set:Nx
2059 #2
2060 {
2061 markdownRenderer
2062 \tl_head:f { \l_tmpa_tl }
2063 \tl_tail:n { #1 }
2064 Prototype
2065 }
2066 }
2067 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
2068 {
2069 \cs_generate_from_arg_count:NNnn
2070 #1
2071 \cs_set:Npn
2072 { #2 }
2073 { }
2074 }
2075 \cs_generate_variant:Nn
2076 \@@_plaintex_define_renderer_prototype:Nn
2077 { cV }
2078 \@@_plaintex_define_renderer_prototypes:
2079 \ExplSyntaxOff

```

## 2.2.5 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.

## 2.2.6 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*.

```
2080 \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.

```
2081 \let\markdownReadAndConvert\relax
```

```
2082 \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*.

```
2083 \catcode`\|=0\catcode`\=12%
```

```
2084 |gdef|markdownBegin{%
```

```
2085 |markdownReadAndConvert{\markdownEnd}%
```

```
2086 |{\markdownEnd}}%
```

```
2087 |endgroup
```

The macro is exposed in the interface, so that the user can create their own markdown environments. Due to the way the arguments are passed to Lua (see Section 3.2.6), the first argument may not contain the string `]]` (regardless of the category code of the bracket symbol (`]`)).

The `\markdownMode` macro specifies how the plain  $\TeX$  implementation interfaces with the Lua interface. The valid values and their meaning are as follows:

- `0` – Shell escape via the 18 output file stream
- `1` – Shell escape via the Lua `os.execute` method
- `2` – Direct Lua access
- `3` – The `lt3luabridge` Lua package

By defining the macro, the user can coerce the package to use a specific mode. If the user does not define the macro prior to loading the plain  $\TeX$  implementation, the correct value will be automatically detected. The outcome of changing the value of `\markdownMode` after the implementation has been loaded is undefined.

The `\markdownMode` macro has been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```

2088 \ExplSyntaxOn
2089 \cs_if_exist:NF
2090 \markdownMode
2091 {
2092 \file_if_exist:nTF
2093 { lt3luabridge.tex }
2094 {
2095 \cs_new:Npn
2096 \markdownMode
2097 { 3 }
2098 }
2099 {
2100 \cs_if_exist:NTF
2101 \directlua
2102 {
2103 \cs_new:Npn
2104 \markdownMode
2105 { 2 }
2106 }
2107 {
2108 \cs_new:Npn
2109 \markdownMode
2110 { 0 }
2111 }
2112 }
2113 }
2114 \ExplSyntaxOff

```

The `\markdownLuaRegisterIBCallback` and `\markdownLuaUnregisterIBCallback` macros have been deprecated and will be removed in Markdown 3.0.0:

```

2115 \def\markdownLuaRegisterIBCallback#1{\relax}%
2116 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

## 2.3 L<sup>A</sup>T<sub>E</sub>X Interface

The L<sup>A</sup>T<sub>E</sub>X interface provides L<sup>A</sup>T<sub>E</sub>X environments for the typesetting of markdown input from within L<sup>A</sup>T<sub>E</sub>X, facilities for setting Lua, plain T<sub>E</sub>X, and L<sup>A</sup>T<sub>E</sub>X options used during the conversion from markdown to plain T<sub>E</sub>X, and facilities for changing the way markdown tokens are rendered. The rest of the interface is inherited from the plain T<sub>E</sub>X interface (see Section 2.2).

The L<sup>A</sup>T<sub>E</sub>X implementation redefines the plain T<sub>E</sub>X logging macros (see Section 3.2.1) to use the L<sup>A</sup>T<sub>E</sub>X `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

2117 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
2118 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
2119 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
2120 \input markdown/markdown

```

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

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

where *<options>* are the L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2). Note that *<options>* inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) and `markdownRendererPrototypes` (see Section 2.3.2.6) keys. This limitation is due to the way L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub> parses package options.

### 2.3.1 Typesetting Markdown

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

The `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```

2121 \newenvironment{markdown}\relax\relax
2122 \newenvironment{markdown*}[1]\relax\relax

```

You may prepend your own code to the `\markdown` macro and append your own code to the `\endmarkdown` macro to produce special effects before and after the `markdown` L<sup>A</sup>T<sub>E</sub>X environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` L<sup>A</sup>T<sub>E</sub>X environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain T<sub>E</sub>X interface.

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

<code>\documentclass{article}</code>	<code>\documentclass{article}</code>
<code>\usepackage{markdown}</code>	<code>\usepackage{markdown}</code>
<code>\begin{document}</code>	<code>\begin{document}</code>
<code>% ...</code>	<code>% ...</code>
<code>\begin{markdown}</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 `\markdownInput` macro accepts a single mandatory parameter containing the filename of a markdown document and expands to the result of the conversion of the input markdown document to plain TeX. Unlike the `\markdownInput` macro provided by the plain TeX interface, this macro also accepts L<sup>A</sup>T<sub>E</sub>X interface options (see Section 2.3.2) as its optional argument. These options will only influence this markdown document.

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

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

### 2.3.2 Options

The L<sup>A</sup>T<sub>E</sub>X 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.

Except for the `plain` option described in Section 2.3.2.1, and the L<sup>A</sup>T<sub>E</sub>X themes described in Section 2.3.2.2, and the L<sup>A</sup>T<sub>E</sub>X setup snippets described in Section 2.3.2.3, L<sup>A</sup>T<sub>E</sub>X 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 2.2.3 and 2.2.4).

The L<sup>A</sup>T<sub>E</sub>X options may be specified when loading the L<sup>A</sup>T<sub>E</sub>X package, when using the `markdown*` L<sup>A</sup>T<sub>E</sub>X environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```
2123 \ExplSyntaxOn
2124 \cs_new:Nn
2125 \@@_setup:n
2126 {
2127 \keys_set:nn
2128 { markdown/latex-options }
2129 { #1 }
2130 }
2131 \let\markdownSetup=\@@_setup:n
2132 \ExplSyntaxOff
```

We may also store L<sup>A</sup>T<sub>E</sub>X options as *setup snippets* and invoke them later using the `\markdownSetupSnippet` macro. The `\markdownSetupSnippet` macro receives two arguments: the name of the setup snippet and the options to store:

```

2133 \newcommand\markdownSetupSnippet[2]{%
2134 \markdownIfSnippetExists{#1}%
2135 {%
2136 \markdownWarning
2137 {Redefined setup snippet \markdownLaTeXThemeName#1}%
2138 \csname markdownLaTeXSetupSnippet%
2139 \markdownLaTeXThemeName#1\endcsname={#2}%
2140 }{%
2141 \newtoks\next
2142 \next={#2}%
2143 \expandafter\let\csname markdownLaTeXSetupSnippet%
2144 \markdownLaTeXThemeName#1\endcsname=\next
2145 }%

```

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

```

2146 \newcommand\markdownIfSnippetExists[3]{%
2147 \ifundefined
2148 {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
2149 {#3}{#2}}%

```

See Section 2.3.2.2 for information on interactions between setup snippets and L<sup>A</sup>T<sub>E</sub>X themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of L<sup>A</sup>T<sub>E</sub>X options, we will maintain the `\g_@@_latex_options_seq` sequence.

```

2150 \ExplSyntaxOn
2151 \seq_new:N \g_@@_latex_options_seq

```

To enable the reflection of default L<sup>A</sup>T<sub>E</sub>X options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```

2152 \prop_new:N \g_@@_latex_option_types_prop
2153 \prop_new:N \g_@@_default_latex_options_prop
2154 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
2155 \seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
2156 \cs_new:Nn
2157 \@@_add_latex_option:nnn
2158 {
2159 \@@_add_option:Vnnn
2160 \c_@@_option_layer_latex_tl
2161 { #1 }
2162 { #2 }
2163 { #3 }
2164 }

```

**2.3.2.1 No default token renderer prototypes** Default token renderer prototypes require L<sup>A</sup>T<sub>E</sub>X packages that may clash with other packages used in a document.

Additionally, if we redefine token renderers and renderer prototypes ourselves, the default definitions will bring no benefit to us. Using the `plain` package option, we can keep the default definitions from the plain TeX implementation (see Section 3.2.2) and prevent the soft L<sup>A</sup>T<sub>E</sub>X prerequisites in Section 1.1.3 from being loaded: The `plain` option must be set before or when loading the package. Setting the option after loading the package will have no effect.

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

```
2165 \@@_add_latex_option:nnn
2166 { plain }
2167 { boolean }
2168 { false }
2169 \ExplSyntaxOff
```

**2.3.2.2 L<sup>A</sup>T<sub>E</sub>X themes** User-defined L<sup>A</sup>T<sub>E</sub>X themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to L<sup>A</sup>T<sub>E</sub>X packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The L<sup>A</sup>T<sub>E</sub>X option `theme=<theme name>` loads a L<sup>A</sup>T<sub>E</sub>X package (further referred to as a *theme*) named `markdowntheme<munged theme name>.sty`, 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, and a value is qualified if and only if it contains at least one forward slash. Themes are inspired by the Beamer L<sup>A</sup>T<sub>E</sub>X 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 intended for a single L<sup>A</sup>T<sub>E</sub>X document class or for a single L<sup>A</sup>T<sub>E</sub>X package. The preferred format of a theme name is `<theme author>/<target LATEX document class or package>/<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, because L<sup>A</sup>T<sub>E</sub>X packages are identified only by their filenames, not by their pathnames. [9] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a L<sup>A</sup>T<sub>E</sub>X package named `markdownthemewitiko_beamer_MU.sty`.

If the L<sup>A</sup>T<sub>E</sub>X option with key `theme` is (repeatedly) specified in the `\usepackage` macro, the loading of the theme(s) will be postponed in first-in-first-out order until after the Markdown L<sup>A</sup>T<sub>E</sub>X package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which

typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` L<sup>A</sup>T<sub>E</sub>X package, and finally the `markdownthemewitiko_dot.sty` L<sup>A</sup>T<sub>E</sub>X package:

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

```
2170 \newif\ifmarkdownLaTeXLoaded
2171 \markdownLaTeXLoadedfalse
2172 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
2173 \ExplSyntaxOn
2174 \tl_new:N \markdownLaTeXThemePackageName
2175 \cs_new:Nn
2176 \@@_set_latex_theme:n
2177 {
2178 \str_if_in:nnF
2179 { #1 }
2180 { / }
2181 {
2182 \markdownError
2183 { Won't~load~theme~with~unqualified~name~#1 }
2184 { Theme~names~must~contain~at~least~one~forward~slash }
2185 }
2186 \str_if_in:nnT
2187 { #1 }
2188 { _ }
2189 {
2190 \markdownError
2191 { Won't~load~theme~with~an~underscore~in~its~name~#1 }
2192 { Theme~names~must~not~contain~underscores~in~their~names }
2193 }
2194 \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
2195 \str_replace_all:Nnn
2196 \markdownLaTeXThemePackageName
2197 { / }
2198 { _ }
2199 \edef\markdownLaTeXThemePackageName{
2200 markdowntheme\markdownLaTeXThemePackageName}
2201 \expandafter\markdownLaTeXThemeLoad\expandafter{
2202 \markdownLaTeXThemePackageName}{#1/}
2203 }
2204 \keys_define:nn
2205 { markdown/latex-options }
```

```

2206 {
2207 theme .code:n = { \@@_set_latex_theme:n { #1 } },
2208 }
2209 \ExplSyntaxOff

```

The  $\LaTeX$  themes have a useful synergy with the setup snippets (see Section 2.3.2): To make it less likely that different themes will define setup snippets with the same name, we will prepend  $\langle theme\ name \rangle/$  before the snippet name and use the result as the snippet name. For example, if the `witiko/dot` theme defines the `product` setup snippet, the setup snippet will be available under the name `witiko/dot/product`. Due to limitations of  $\LaTeX$ , themes may not be loaded after the beginning of a  $\LaTeX$  document.

```

2210 \ExplSyntaxOn
2211 \@onlypreamble
2212 \@@_set_latex_theme:n
2213 \ExplSyntaxOff

```

Example 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[theme=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.

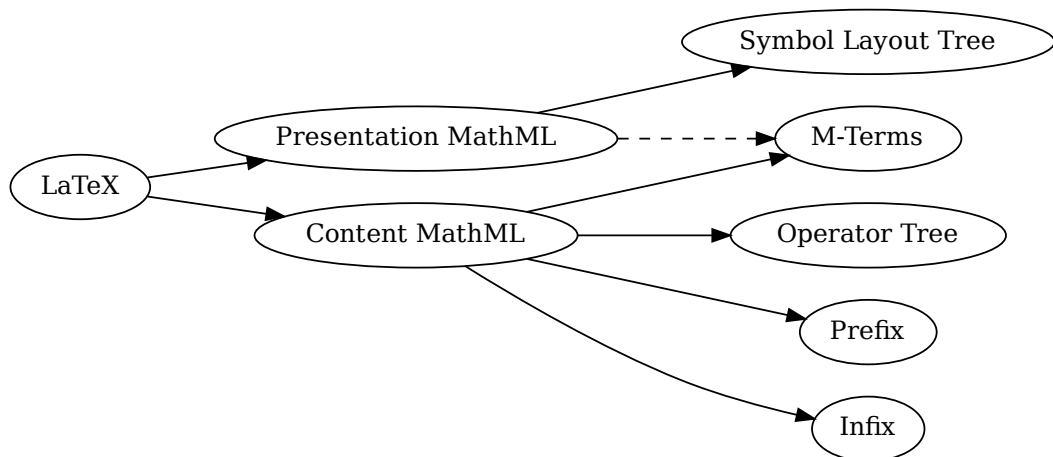


Figure 4: Various formats of mathematical formulae

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.

2214 \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[theme=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

```

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

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

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

```



Chapter 1

Introduction

1.1 Section
 1.1.1 Subsection
 Hello *Markdown!*

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

Table 1.1: Table

Figure 5: The banner of the Markdown package

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

2215 \ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%

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

```

\documentclass{article}
\usepackage[theme=witiko/tilde]{markdown}
\begin{document}

```

```

\begin{markdown}
Bartel~Leendert van~der~Waerden
\end{markdown}
\end{document}

```

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

```
2216 \ProvidesPackage{markdownthemewitiko_tilde}[2021/03/22]%
```

Please, see Section 3.3.2.1 for implementation details of the example themes.

2.3.2.3 L^AT_EX setup snippets The L^AT_EX option with key `snippet` invokes a snippet named $\langle value \rangle$:

```

2217 \ExplSyntaxOn
2218 \keys_define:nn
2219   { markdown/latex-options }
2220   {
2221     snippet .code:n = {
2222       \markdownIfSnippetExists{#1}
2223       {
2224         \expandafter\markdownSetup\expandafter{
2225           \the\csname markdownLaTeXSetupSnippet
2226             \markdownLaTeXThemeName#1\endcsname}
2227         }{
2228           \markdownError
2229             {Can't~invoke~setup~snippet~#1}
2230             {The~setup~snippet~is~undefined}
2231         }
2232       }
2233     }
2234 \ExplSyntaxOff

```

Here is how we can use setup snippets to store options and invoke them later:

```

\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*}
```

2.3.2.4 Plain T_EX Interface Options Here, we automatically define plain T_EX macros and the $\langle key \rangle = \langle value \rangle$ interface for the above L^AT_EX options.

```
2235 \ExplSyntaxOn  
2236 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:  
2237   {  
2238     \seq_map_inline:Nn  
2239       \g_@@_latex_options_seq  
2240       {  
2241         \@@_plain_tex_define_option_command:n  
2242         { ##1 }  
2243       }  
2244   }
```

Furthermore, we also define the $\langle key \rangle = \langle value \rangle$ interface for all option macros recognized by the Lua and plain T_EX interfaces.

```
2244   \seq_map_inline:Nn  
2245     \g_@@_option_layers_seq  
2246     {  
2247       \seq_map_inline:cn  
2248       { g_@@_ ##1 _options_seq }  
2249     }
```

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`.

```
2250     \@@_with_various_cases:nn  
2251     { #####1 }  
2252     {  
2253       \@@_latex_define_option_keyval:nnn  
2254       { ##1 }  
2255       { #####1 }  
2256       { #####1##1 }  
2257     }
```

```

2258     }
2259   }
2260 }
2261 \cs_new:Nn \@@_latex_define_option_keyval:nnn
2262 {
2263   \prop_get:cnN
2264     { g_@@_ #1 _option_types_prop }
2265     { #2 }
2266     \l_tmpa_tl
2267   \keys_define:nn
2268     { markdown/latex-options }
2269     {
2270       #3 .code:n = {
2271         \@@_set_option_value:nn
2272           { #2 }
2273           { ##1 }
2274       },
2275     }
2276   \str_if_eq:VVT
2277     \l_tmpa_tl
2278     \c_@@_option_type_boolean_tl
2279     {
2280       \keys_define:nn
2281         { markdown/latex-options }
2282         {
2283           #3 .default:n = { true },
2284         }
2285     }

```

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.

```

2286   \str_if_eq:VVT
2287     \l_tmpa_tl
2288     \c_@@_option_type_clist_tl
2289     {
2290       \tl_set:Nn
2291         \l_tmpa_tl
2292         { #3 }
2293       \tl_reverse:N
2294         \l_tmpa_tl
2295       \str_if_eq:enF
2296         {
2297           \tl_head:V
2298           \l_tmpa_tl

```

```

2299     }
2300     { s }
2301     {
2302         \msg_error:nnn
2303         { @@ }
2304         { malformed-name-for-clist-option }
2305         { #3 }
2306     }
2307     \tl_set:Nx
2308     \l_tmpa_tl
2309     {
2310         \tl_tail:V
2311         \l_tmpa_tl
2312     }
2313     \tl_reverse:N
2314     \l_tmpa_tl
2315     \tl_put_right:Nn
2316     \l_tmpa_tl
2317     {
2318         .code:n = {
2319             \@@_get_option_value:nN
2320             { #2 }
2321             \l_tmpa_tl
2322             \clist_set:NV
2323             \l_tmpa_clist
2324             { \l_tmpa_tl, { ##1 } }
2325             \@@_set_option_value:nV
2326             { #2 }
2327             \l_tmpa_clist
2328         }
2329     }
2330     \keys_define:nV
2331     { markdown/latex-options }
2332     \l_tmpa_tl
2333 }
2334 }
2335 \cs_generate_variant:Nn
2336 \clist_set:Nn
2337 { NV }
2338 \cs_generate_variant:Nn
2339 \keys_define:nn
2340 { nV }
2341 \cs_generate_variant:Nn
2342 \@@_set_option_value:nn
2343 { nV }
2344 \prg_generate_conditional_variant:Nnn
2345 \str_if_eq:nn

```

```

2346 { en }
2347 { F }
2348 \msg_new:nnn
2349 { @@ }
2350 { malformed-name-for-clist-option }
2351 {
2352   Clist~option~name~#1~does~not~end~with~-s.
2353 }
2354 \@@_latex_define_option_commands_and_keyvals:
2355 \ExplSyntaxOff

```

The `finalizeCache` and `frozenCache` plain \TeX options are exposed through \LaTeX options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizcache` and `frozencache` package options with similar semantics, the `Markdown` package also recognizes these as aliases and recognizes them as document class options. By passing `finalizcache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

```

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

```

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

```

2356 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
2357 \DeclareOption{frozencache}{\markdownSetup{frozenCache}}

```

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

```

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

```

2.3.2.5 Plain \TeX Markdown Token Renderers The \LaTeX interface recognizes an option with the `renderers` key, whose value must be a list of options that map directly to the markdown token renderer macros exposed by the plain \TeX interface (see Section 2.2.3).

```

2358 \ExplSyntaxOn
2359 \cs_new:Nn \@@_latex_define_renderers:
2360 {
2361   \seq_map_function:NN
2362     \g_@@_renderers_seq
2363     \@@_latex_define_renderer:n
2364 }
2365 \cs_new:Nn \@@_latex_define_renderer:n
2366 {
2367   \@@_renderer_tl_to_csname:nN
2368     { #1 }
2369     \l_tmpa_tl
2370   \prop_get:NnN
2371     \g_@@_renderer_arities_prop
2372     { #1 }
2373     \l_tmpb_tl
2374   \@@_latex_define_renderer:ncV
2375     { #1 }
2376     { \l_tmpa_tl }
2377     \l_tmpb_tl
2378 }
2379 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2380 {
2381   \tl_set:Nn
2382     \l_tmpa_tl
2383     { \str_uppercase:n { #1 } }
2384   \tl_set:Nx
2385     #2
2386     {
2387       markdownRenderer
2388       \tl_head:f { \l_tmpa_tl }
2389       \tl_tail:n { #1 }
2390     }
2391 }
2392 \cs_new:Nn \@@_latex_define_renderer:nNn
2393 {
2394   \@@_with_various_cases:nn
2395     { #1 }
2396     {
2397       \keys_define:nn
2398         { markdown/latex-options/renderers }
2399         {
2400           ##1 .code:n = {
2401             \cs_generate_from_arg_count:NNnn
2402               #2
2403               \cs_set:Npn
2404                 { #3 }

```

```

2405             { ####1 }
2406         },
2407     }
2408 }
2409 }
2410 \cs_generate_variant:Nn
2411   \@@_latex_define_renderer:nNn
2412   { ncV }
2413 \ExplSyntaxOff

```

The following example L^AT_EX code showcases a possible configuration of the `\markdownRendererLink` and `\markdownRendererEmphasis` markdown token renderers.

```

\markdownSetup{
  renderers = {
    link = {#4},           % Render links as the link title.
    emphasis = {\emph{#1}}, % Render emphasized text via \emph`.
  }
}

```

2.3.2.6 Plain T_EX Markdown Token Renderer Prototypes The L^AT_EX interface recognizes an option with the `rendererPrototypes` key, whose value must be a list of options that map directly to the markdown token renderer prototype macros exposed by the plain T_EX interface (see Section 2.2.4).

```

2414 \ExplSyntaxOn
2415 \cs_new:Nn \@@_latex_define_renderer_prototypes:
2416   {
2417     \seq_map_function:NN
2418       \g_@@_renderers_seq
2419       \@@_latex_define_renderer_prototype:n
2420   }
2421 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2422   {
2423     \@@_renderer_prototype_tl_to_csname:nN
2424     { #1 }
2425     \l_tmpa_tl
2426     \prop_get:NnN
2427       \g_@@_renderer_arities_prop
2428       { #1 }
2429     \l_tmpb_tl
2430     \@@_latex_define_renderer_prototype:ncV
2431     { #1 }
2432     { \l_tmpa_tl }
2433     \l_tmpb_tl

```

```

2434 }
2435 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2436 {
2437   \@@_with_various_cases:nn
2438   { #1 }
2439   {
2440     \keys_define:nn
2441     { markdown/latex-options/renderer-prototypes }
2442     {
2443       ##1 .code:n = {
2444         \cs_generate_from_arg_count:NNnn
2445         #2
2446         \cs_set:Npn
2447         { #3 }
2448         { #####1 }
2449       },
2450     }
2451   }
2452 }
2453 \cs_generate_variant:Nn
2454 \@@_latex_define_renderer_prototype:nNn
2455 { ncV }
2456 \ExplSyntaxOff

```

The following example L^AT_EX code showcases a possible configuration of the `\markdownRendererImagePrototype` and `\markdownRendererCodeSpanPrototype` markdown token renderer prototypes.

```

\markdownSetup{
  rendererPrototypes = {
    image = {\includegraphics{#2}},
    codeSpan = {\texttt{#1}}, % Render inline code via '\texttt'.
  }
}

```

2.4 ConT_EXt Interface

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

```

2457 \writestatus{loading}{ConTEXt User Module / markdown}%
2458 \startmodule[markdown]
2459 \unprotect

```

The ConT_EXt implementation redefines the plain T_EX logging macros (see Section 3.2.1) to use the ConT_EXt `\writestatus` macro.

```
2460 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2461 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
2462 \def\dospecials{\do\ \do\\\do{\do\}\do\$\do\&%
2463   \do\#\do\~\do\_ \do\% \do\~}%
2464 \input markdown/markdown
```

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

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

It is expected that the special plain T_EX 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` command.

```
2465 \let\startmarkdown\relax
2466 \let\stopmarkdown\relax
2467 \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 T_EX interface.

The following example ConT_EXt 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 T_EX. Unlike the `\markdownInput` macro

provided by the plain \TeX interface, this macro also accepts Con \TeX t 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 Con \TeX t options are represented by a comma-delimited list of $\langle key \rangle = \langle value \rangle$ pairs. For boolean options, the $= \langle value \rangle$ part is optional, and $\langle key \rangle$ will be interpreted as $\langle key \rangle = \text{true}$ (or, equivalently, $\langle key \rangle = \text{yes}$) if the $= \langle value \rangle$ part has been omitted.

Con \TeX t options map directly to the options recognized by the plain \TeX interface (see Section 2.2.2).

The Con \TeX t options may be specified when using the `\inputmarkdown` macro (see Section 2.4), or via the `\setupmarkdown` macro. The `\setupmarkdown` macro receives the options to set up as its only argument:

```

2468 \ExplSyntaxOn
2469 \cs_new:Nn
2470   \@@_setup:n
2471   {
2472     \keys_set:n
2473       { markdown/context-options }
2474       { #1 }
2475   }
2476 \long\def\setupmarkdown[#1]
2477   {
2478     \@@_setup:n
2479       { #1 }
2480   }
2481 \ExplSyntaxOff

```

2.4.2.1 Con \TeX t Interface Options We define the $\langle key \rangle = \langle value \rangle$ interface for all option macros recognized by the Lua and plain \TeX interfaces.

```

2482 \ExplSyntaxOn
2483 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2484   {
2485     \seq_map_inline:Nn
2486       \g_@@_option_layers_seq
2487       {

```

```

2488     \seq_map_inline:cn
2489     { g_@@_ ##1 _options_seq }
2490     {

```

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.

```

2491         \@@_with_various_cases:nn
2492         { #####1 }
2493         {
2494             \@@_context_define_option_keyval:nnn
2495             { ##1 }
2496             { #####1 }
2497             { #####1 }
2498         }
2499     }
2500 }
2501 }

```

Furthermore, we also accept caseless variants of options in line with the style of ConTeXt.

```

2502 \cs_new:Nn \@@_caseless:N
2503 {
2504     \regex_replace_all:nnN
2505     { ([a-z])([A-Z]) }
2506     { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
2507     #1
2508     \tl_set:Nx
2509     #1
2510     { #1 }
2511 }
2512 \seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
2513 \cs_new:Nn \@@_context_define_option_keyval:nnn
2514 {
2515     \prop_get:cnN
2516     { g_@@_ #1 _option_types_prop }
2517     { #2 }
2518     \l_tmpa_tl
2519     \keys_define:nn
2520     { markdown/context-options }
2521     {
2522         #3 .code:n = {
2523             \tl_set:Nx
2524             \l_tmpa_tl
2525             {
2526                 \str_case:nnF

```

```

2527         { ##1 }
2528         {
2529             { yes } { true }
2530             { no } { false }
2531         }
2532         { ##1 }
2533     }
2534     \@@_set_option_value:nV
2535     { #2 }
2536     \l_tmpa_tl
2537 },
2538 }
2539 \str_if_eq:VVT
2540 \l_tmpa_tl
2541 \c_@@_option_type_boolean_tl
2542 {
2543     \keys_define:nn
2544     { markdown/context-options }
2545     {
2546         #3 .default:n = { true },
2547     }
2548 }
2549 }
2550 \cs_generate_variant:Nn
2551 \@@_set_option_value:nn
2552 { nV }
2553 \@@_context_define_option_commands_and_keyvals:
2554 \ExplSyntaxOff

```

3 Implementation

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

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

3.1 Lua Implementation

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

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

```
2555 local upper, gsub, format, length =
2556   string.upper, string.gsub, string.format, string.len
2557 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2558   lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2559   lpeg.Cmt, lpeg.Cc, lpeg.Ct, lpeg.B, lpeg.Cs, 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.

```
2560 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.

```
2561 function util.err(msg, exit_code)
2562   io.stderr:write("markdown.lua: " .. msg .. "\n")
2563   os.exit(exit_code or 1)
2564 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.

```
2565 function util.cache(dir, string, salt, transform, suffix)
2566   local digest = md5.sumhexa(string .. (salt or ""))
2567   local name = util.pathname(dir, digest .. suffix)
2568   local file = io.open(name, "r")
2569   if file == nil then -- If no cache entry exists, then create a new one.
2570     file = assert(io.open(name, "w"),
2571       [[Could not open file ]] .. name .. [[ for writing]])
2572     local result = string
2573     if transform ~= nil then
2574       result = transform(result)
2575     end
2576     assert(file:write(result))
2577     assert(file:close())
2578   end
2579   return name
2580 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.

```
2581 function util.cache_verbatim(dir, string)
2582   string = string:gsub('[\r\n%s]*$', '')
2583   local name = util.cache(dir, string, nil, nil, ".verbatim")
2584   return name
2585 end
```

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

```
2586 function util.table_copy(t)
2587   local u = { }
2588   for k, v in pairs(t) do u[k] = v end
2589   return setmetatable(u, getmetatable(t))
2590 end
```

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

```
2591 function util.encode_json_string(s)
2592   s = s:gsub([[\\]], [[\\]])
2593   s = s:gsub([[\"]], [[\"]])
2594   return [["]] .. s .. [["]]
2595 end
```

The `util.lookup_files` method looks up files with filename `f` and returns its path. If the `kpathsea` library is available, it will search for files not only in the current working directory but also in the `TEX` directory structure. Further options for `kpathsea` can be specified in table `options`. [1, Section 10.7.4]

```
2596 util.lookup_files = (function()
2597   local ran_ok, kpse = pcall(require, "kpse")
2598   if ran_ok then
2599     kpse.set_program_name("luatex")
2600   else
2601     kpse = { lookup = function(f, _) return f end }
2602   end
2603
2604   local function lookup_files(f, options)
2605     return kpse.lookup(f, options)
2606   end
2607
2608   return lookup_files
2609 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].

```
2610 function util.expand_tabs_in_line(s, tabstop)
```

```

2611 local tab = tabstop or 4
2612 local corr = 0
2613 return (s:gsub("()\t", function(p)
2614         local sp = tab - (p - 1 + corr) % tab
2615         corr = corr - 1 + sp
2616         return string.rep(" ", sp)
2617     end))
2618 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.

```

2619 function util.walk(t, f)
2620     local typ = type(t)
2621     if typ == "string" then
2622         f(t)
2623     elseif typ == "table" then
2624         local i = 1
2625         local n
2626         n = t[i]
2627         while n do
2628             util.walk(n, f)
2629             i = i + 1
2630             n = t[i]
2631         end
2632     elseif typ == "function" then
2633         local ok, val = pcall(t)
2634         if ok then
2635             util.walk(val, f)
2636         end
2637     else
2638         f(tostring(t))
2639     end
2640 end

```

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

```

2641 function util.flatten(ary)
2642     local new = {}
2643     for _,v in ipairs(ary) do
2644         if type(v) == "table" then
2645             for _,w in ipairs(util.flatten(v)) do
2646                 new[#new + 1] = w
2647             end
2648         else
2649             new[#new + 1] = v
2650         end
2651     end
2652 end

```

```

2650     end
2651   end
2652   return new
2653 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.

```

2654 function util.rope_to_string(rope)
2655   local buffer = {}
2656   util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2657   return table.concat(buffer)
2658 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.

```

2659 function util.rope_last(rope)
2660   if #rope == 0 then
2661     return nil
2662   else
2663     local l = rope[#rope]
2664     if type(l) == "table" then
2665       return util.rope_last(l)
2666     else
2667       return l
2668     end
2669   end
2670 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 $1 \leq i \leq \#ary$.

```

2671 function util.intersperse(ary, x)
2672   local new = {}
2673   local l = #ary
2674   for i,v in ipairs(ary) do
2675     local n = #new
2676     new[n + 1] = v
2677     if i ~= l then
2678       new[n + 2] = x
2679     end
2680   end
2681   return new
2682 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 $1 \leq i \leq \#ary$.

```

2683 function util.map(ary, f)
2684   local new = {}

```

```

2685   for i,v in ipairs(ary) do
2686     new[i] = f(v)
2687   end
2688   return new
2689 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.

```

2690 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

2691   local char_escapes_list = ""
2692   for i,_ in pairs(char_escapes) do
2693     char_escapes_list = char_escapes_list .. i
2694   end

```

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

```

2695   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.

```

2696   if string_escapes then
2697     for k,v in pairs(string_escapes) do
2698       escapable = P(k) / v + escapable
2699     end
2700   end

```

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

```

2701   local escape_string = Cs((escapable + any)^0)

```

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

```

2702   return function(s)
2703     return lpeg.match(escape_string, s)
2704   end
2705 end

```


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

```
2706 function util.pathname(dir, file)
2707   if #dir == 0 then
2708     return file
2709   else
2710     return dir .. "/" .. file
2711   end
2712 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.

```
2713 local entities = {}
2714
2715 local character_entities = {
2716   ["Tab"] = 9,
2717   ["NewLine"] = 10,
2718   ["excl"] = 33,
2719   ["quot"] = 34,
2720   ["QUOT"] = 34,
2721   ["num"] = 35,
2722   ["dollar"] = 36,
2723   ["percent"] = 37,
2724   ["amp"] = 38,
2725   ["AMP"] = 38,
2726   ["apos"] = 39,
2727   ["lpar"] = 40,
2728   ["rpar"] = 41,
2729   ["ast"] = 42,
2730   ["midast"] = 42,
2731   ["plus"] = 43,
2732   ["comma"] = 44,
2733   ["period"] = 46,
2734   ["sol"] = 47,
2735   ["colon"] = 58,
2736   ["semi"] = 59,
2737   ["lt"] = 60,
2738   ["LT"] = 60,
2739   ["equals"] = 61,
2740   ["gt"] = 62,
2741   ["GT"] = 62,
2742   ["quest"] = 63,
2743   ["commat"] = 64,
```

2744 ["lsqb"] = 91,
2745 ["lbrack"] = 91,
2746 ["bsol"] = 92,
2747 ["rsqb"] = 93,
2748 ["rbrack"] = 93,
2749 ["Hat"] = 94,
2750 ["lowbar"] = 95,
2751 ["grave"] = 96,
2752 ["DiacriticalGrave"] = 96,
2753 ["lcub"] = 123,
2754 ["lbrace"] = 123,
2755 ["verbar"] = 124,
2756 ["vert"] = 124,
2757 ["VerticalLine"] = 124,
2758 ["rcub"] = 125,
2759 ["rbrace"] = 125,
2760 ["nbsp"] = 160,
2761 ["NonBreakingSpace"] = 160,
2762 ["iexcl"] = 161,
2763 ["cent"] = 162,
2764 ["pound"] = 163,
2765 ["curren"] = 164,
2766 ["yen"] = 165,
2767 ["brvbar"] = 166,
2768 ["sect"] = 167,
2769 ["Dot"] = 168,
2770 ["die"] = 168,
2771 ["DoubleDot"] = 168,
2772 ["uml"] = 168,
2773 ["copy"] = 169,
2774 ["COPY"] = 169,
2775 ["ordf"] = 170,
2776 ["laquo"] = 171,
2777 ["not"] = 172,
2778 ["shy"] = 173,
2779 ["reg"] = 174,
2780 ["circledR"] = 174,
2781 ["REG"] = 174,
2782 ["macr"] = 175,
2783 ["OverBar"] = 175,
2784 ["strns"] = 175,
2785 ["deg"] = 176,
2786 ["plusmn"] = 177,
2787 ["pm"] = 177,
2788 ["PlusMinus"] = 177,
2789 ["sup2"] = 178,
2790 ["sup3"] = 179,

2791 ["acute"] = 180,
2792 ["DiacriticalAcute"] = 180,
2793 ["micro"] = 181,
2794 ["para"] = 182,
2795 ["middot"] = 183,
2796 ["centerdot"] = 183,
2797 ["CenterDot"] = 183,
2798 ["cedil"] = 184,
2799 ["Cedilla"] = 184,
2800 ["sup1"] = 185,
2801 ["ordm"] = 186,
2802 ["raquo"] = 187,
2803 ["frac14"] = 188,
2804 ["frac12"] = 189,
2805 ["half"] = 189,
2806 ["frac34"] = 190,
2807 ["iquest"] = 191,
2808 ["Agrave"] = 192,
2809 ["Aacute"] = 193,
2810 ["Acirc"] = 194,
2811 ["Atilde"] = 195,
2812 ["Auml"] = 196,
2813 ["Aring"] = 197,
2814 ["AElig"] = 198,
2815 ["Ccedil"] = 199,
2816 ["Egrave"] = 200,
2817 ["Eacute"] = 201,
2818 ["Ecirc"] = 202,
2819 ["Euml"] = 203,
2820 ["Igrave"] = 204,
2821 ["Iacute"] = 205,
2822 ["Icirc"] = 206,
2823 ["Iuml"] = 207,
2824 ["ETH"] = 208,
2825 ["Ntilde"] = 209,
2826 ["Ograve"] = 210,
2827 ["Oacute"] = 211,
2828 ["Ocirc"] = 212,
2829 ["Otilde"] = 213,
2830 ["Ouml"] = 214,
2831 ["times"] = 215,
2832 ["Oslash"] = 216,
2833 ["Ugrave"] = 217,
2834 ["Uacute"] = 218,
2835 ["Ucirc"] = 219,
2836 ["Uuml"] = 220,
2837 ["Yacute"] = 221,

2838 ["THORN"] = 222,
2839 ["szlig"] = 223,
2840 ["agrave"] = 224,
2841 ["acute"] = 225,
2842 ["acirc"] = 226,
2843 ["atilde"] = 227,
2844 ["auml"] = 228,
2845 ["aring"] = 229,
2846 ["aelig"] = 230,
2847 ["cedil"] = 231,
2848 ["egrave"] = 232,
2849 ["eacute"] = 233,
2850 ["ecirc"] = 234,
2851 ["euml"] = 235,
2852 ["igrave"] = 236,
2853 ["iacute"] = 237,
2854 ["icirc"] = 238,
2855 ["iuml"] = 239,
2856 ["eth"] = 240,
2857 ["ntilde"] = 241,
2858 ["ograve"] = 242,
2859 ["oacute"] = 243,
2860 ["ocirc"] = 244,
2861 ["otilde"] = 245,
2862 ["ouml"] = 246,
2863 ["divide"] = 247,
2864 ["div"] = 247,
2865 ["oslash"] = 248,
2866 ["ugrave"] = 249,
2867 ["uacute"] = 250,
2868 ["ucirc"] = 251,
2869 ["uuml"] = 252,
2870 ["yacute"] = 253,
2871 ["thorn"] = 254,
2872 ["yuml"] = 255,
2873 ["Amacr"] = 256,
2874 ["amacr"] = 257,
2875 ["Abreve"] = 258,
2876 ["abreve"] = 259,
2877 ["Aogon"] = 260,
2878 ["aogon"] = 261,
2879 ["Cacute"] = 262,
2880 ["cacute"] = 263,
2881 ["Ccirc"] = 264,
2882 ["ccirc"] = 265,
2883 ["Cdot"] = 266,
2884 ["cdot"] = 267,

2885 ["Ccaron"] = 268,
2886 ["ccaron"] = 269,
2887 ["Dcaron"] = 270,
2888 ["dcaron"] = 271,
2889 ["Dstrok"] = 272,
2890 ["dstrok"] = 273,
2891 ["Emacr"] = 274,
2892 ["emacr"] = 275,
2893 ["Edot"] = 278,
2894 ["edot"] = 279,
2895 ["Eogon"] = 280,
2896 ["eogon"] = 281,
2897 ["Ecaron"] = 282,
2898 ["ecaron"] = 283,
2899 ["Gcirc"] = 284,
2900 ["gcirc"] = 285,
2901 ["Gbreve"] = 286,
2902 ["gbreve"] = 287,
2903 ["Gdot"] = 288,
2904 ["gdot"] = 289,
2905 ["Gcedil"] = 290,
2906 ["Hcirc"] = 292,
2907 ["hcirc"] = 293,
2908 ["Hstrok"] = 294,
2909 ["hstrok"] = 295,
2910 ["Itilde"] = 296,
2911 ["itilde"] = 297,
2912 ["Imacr"] = 298,
2913 ["imacr"] = 299,
2914 ["Iogon"] = 302,
2915 ["iogon"] = 303,
2916 ["Idot"] = 304,
2917 ["imath"] = 305,
2918 ["inodot"] = 305,
2919 ["IJlig"] = 306,
2920 ["ijlig"] = 307,
2921 ["Jcirc"] = 308,
2922 ["jcirc"] = 309,
2923 ["Kcedil"] = 310,
2924 ["kcedil"] = 311,
2925 ["kgreen"] = 312,
2926 ["Lacute"] = 313,
2927 ["lacute"] = 314,
2928 ["Lcedil"] = 315,
2929 ["lcedil"] = 316,
2930 ["Lcaron"] = 317,
2931 ["lcaron"] = 318,

2932 ["Lmidot"] = 319,
2933 ["lmidot"] = 320,
2934 ["Lstrok"] = 321,
2935 ["lstrok"] = 322,
2936 ["Nacute"] = 323,
2937 ["nacute"] = 324,
2938 ["Ncedil"] = 325,
2939 ["ncedil"] = 326,
2940 ["Ncaron"] = 327,
2941 ["ncaron"] = 328,
2942 ["napos"] = 329,
2943 ["ENG"] = 330,
2944 ["eng"] = 331,
2945 ["Omacr"] = 332,
2946 ["omacr"] = 333,
2947 ["Odblac"] = 336,
2948 ["odblac"] = 337,
2949 ["OElig"] = 338,
2950 ["oelig"] = 339,
2951 ["Racute"] = 340,
2952 ["racute"] = 341,
2953 ["Rcedil"] = 342,
2954 ["rcedil"] = 343,
2955 ["Rcaron"] = 344,
2956 ["rcaron"] = 345,
2957 ["Sacute"] = 346,
2958 ["sacute"] = 347,
2959 ["Scirc"] = 348,
2960 ["scirc"] = 349,
2961 ["Scedil"] = 350,
2962 ["scedil"] = 351,
2963 ["Scaron"] = 352,
2964 ["scaron"] = 353,
2965 ["Tcedil"] = 354,
2966 ["tcedil"] = 355,
2967 ["Tcaron"] = 356,
2968 ["tcaron"] = 357,
2969 ["Tstrok"] = 358,
2970 ["tstrok"] = 359,
2971 ["Utilde"] = 360,
2972 ["utilde"] = 361,
2973 ["Umacr"] = 362,
2974 ["umacr"] = 363,
2975 ["Ubreve"] = 364,
2976 ["ubreve"] = 365,
2977 ["Uring"] = 366,
2978 ["uring"] = 367,

2979 ["Udblac"] = 368,
 2980 ["udblac"] = 369,
 2981 ["Uogon"] = 370,
 2982 ["uogon"] = 371,
 2983 ["Wcirc"] = 372,
 2984 ["wcirc"] = 373,
 2985 ["Ycirc"] = 374,
 2986 ["ycirc"] = 375,
 2987 ["Yuml"] = 376,
 2988 ["Zacute"] = 377,
 2989 ["zacute"] = 378,
 2990 ["Zdot"] = 379,
 2991 ["zdot"] = 380,
 2992 ["Zcaron"] = 381,
 2993 ["zcaron"] = 382,
 2994 ["fnof"] = 402,
 2995 ["imped"] = 437,
 2996 ["gacute"] = 501,
 2997 ["jmath"] = 567,
 2998 ["circ"] = 710,
 2999 ["caron"] = 711,
 3000 ["Hacek"] = 711,
 3001 ["breve"] = 728,
 3002 ["Breve"] = 728,
 3003 ["dot"] = 729,
 3004 ["DiacriticalDot"] = 729,
 3005 ["ring"] = 730,
 3006 ["ogon"] = 731,
 3007 ["tilde"] = 732,
 3008 ["DiacriticalTilde"] = 732,
 3009 ["dblac"] = 733,
 3010 ["DiacriticalDoubleAcute"] = 733,
 3011 ["DownBreve"] = 785,
 3012 ["UnderBar"] = 818,
 3013 ["Alpha"] = 913,
 3014 ["Beta"] = 914,
 3015 ["Gamma"] = 915,
 3016 ["Delta"] = 916,
 3017 ["Epsilon"] = 917,
 3018 ["Zeta"] = 918,
 3019 ["Eta"] = 919,
 3020 ["Theta"] = 920,
 3021 ["Iota"] = 921,
 3022 ["Kappa"] = 922,
 3023 ["Lambda"] = 923,
 3024 ["Mu"] = 924,
 3025 ["Nu"] = 925,

3026 ["Xi"] = 926,
3027 ["Omicron"] = 927,
3028 ["Pi"] = 928,
3029 ["Rho"] = 929,
3030 ["Sigma"] = 931,
3031 ["Tau"] = 932,
3032 ["Upsilon"] = 933,
3033 ["Phi"] = 934,
3034 ["Chi"] = 935,
3035 ["Psi"] = 936,
3036 ["Omega"] = 937,
3037 ["alpha"] = 945,
3038 ["beta"] = 946,
3039 ["gamma"] = 947,
3040 ["delta"] = 948,
3041 ["epsiv"] = 949,
3042 ["varepsilon"] = 949,
3043 ["epsilon"] = 949,
3044 ["zeta"] = 950,
3045 ["eta"] = 951,
3046 ["theta"] = 952,
3047 ["iota"] = 953,
3048 ["kappa"] = 954,
3049 ["lambda"] = 955,
3050 ["mu"] = 956,
3051 ["nu"] = 957,
3052 ["xi"] = 958,
3053 ["omicron"] = 959,
3054 ["pi"] = 960,
3055 ["rho"] = 961,
3056 ["sigmav"] = 962,
3057 ["varsigma"] = 962,
3058 ["sigmaf"] = 962,
3059 ["sigma"] = 963,
3060 ["tau"] = 964,
3061 ["upsi"] = 965,
3062 ["upsilon"] = 965,
3063 ["phi"] = 966,
3064 ["phiv"] = 966,
3065 ["varphi"] = 966,
3066 ["chi"] = 967,
3067 ["psi"] = 968,
3068 ["omega"] = 969,
3069 ["thetav"] = 977,
3070 ["vartheta"] = 977,
3071 ["thetasym"] = 977,
3072 ["Upsi"] = 978,

3073 ["upsih"] = 978,
3074 ["straightphi"] = 981,
3075 ["piv"] = 982,
3076 ["varpi"] = 982,
3077 ["Gammad"] = 988,
3078 ["gammad"] = 989,
3079 ["digamma"] = 989,
3080 ["kappav"] = 1008,
3081 ["varkappa"] = 1008,
3082 ["rhov"] = 1009,
3083 ["varrho"] = 1009,
3084 ["epsi"] = 1013,
3085 ["straightepsilon"] = 1013,
3086 ["bepsi"] = 1014,
3087 ["backepsilon"] = 1014,
3088 ["IOcy"] = 1025,
3089 ["DJcy"] = 1026,
3090 ["GJcy"] = 1027,
3091 ["Jukcy"] = 1028,
3092 ["DScy"] = 1029,
3093 ["Iukcy"] = 1030,
3094 ["YIcy"] = 1031,
3095 ["Jsercy"] = 1032,
3096 ["LJcy"] = 1033,
3097 ["NJcy"] = 1034,
3098 ["TSHcy"] = 1035,
3099 ["KJcy"] = 1036,
3100 ["Ubrcy"] = 1038,
3101 ["DZcy"] = 1039,
3102 ["Acy"] = 1040,
3103 ["Bcy"] = 1041,
3104 ["Vcy"] = 1042,
3105 ["Gcy"] = 1043,
3106 ["Dcy"] = 1044,
3107 ["IEcy"] = 1045,
3108 ["ZHcy"] = 1046,
3109 ["Zcy"] = 1047,
3110 ["Icy"] = 1048,
3111 ["Jcy"] = 1049,
3112 ["Kcy"] = 1050,
3113 ["Lcy"] = 1051,
3114 ["Mcy"] = 1052,
3115 ["Ncy"] = 1053,
3116 ["Ocy"] = 1054,
3117 ["Pcy"] = 1055,
3118 ["Rcy"] = 1056,
3119 ["Scy"] = 1057,

3120 ["Tcy"] = 1058,
3121 ["Ucy"] = 1059,
3122 ["Fcy"] = 1060,
3123 ["KHcy"] = 1061,
3124 ["TScy"] = 1062,
3125 ["CHcy"] = 1063,
3126 ["SHcy"] = 1064,
3127 ["SHCHcy"] = 1065,
3128 ["HARDcy"] = 1066,
3129 ["Ycy"] = 1067,
3130 ["SOFTcy"] = 1068,
3131 ["Ecy"] = 1069,
3132 ["YUcy"] = 1070,
3133 ["YAcy"] = 1071,
3134 ["acy"] = 1072,
3135 ["bcy"] = 1073,
3136 ["vcy"] = 1074,
3137 ["gcy"] = 1075,
3138 ["dcy"] = 1076,
3139 ["iecy"] = 1077,
3140 ["zhcy"] = 1078,
3141 ["zcy"] = 1079,
3142 ["icy"] = 1080,
3143 ["jcy"] = 1081,
3144 ["kcy"] = 1082,
3145 ["lcy"] = 1083,
3146 ["mcy"] = 1084,
3147 ["ncy"] = 1085,
3148 ["ocy"] = 1086,
3149 ["pcy"] = 1087,
3150 ["rcy"] = 1088,
3151 ["scy"] = 1089,
3152 ["tcy"] = 1090,
3153 ["ucy"] = 1091,
3154 ["fcy"] = 1092,
3155 ["khcy"] = 1093,
3156 ["tscy"] = 1094,
3157 ["chcy"] = 1095,
3158 ["shcy"] = 1096,
3159 ["shchcy"] = 1097,
3160 ["hardcy"] = 1098,
3161 ["ycy"] = 1099,
3162 ["softcy"] = 1100,
3163 ["ecy"] = 1101,
3164 ["yucy"] = 1102,
3165 ["yacy"] = 1103,
3166 ["iocy"] = 1105,

3167 ["djcy"] = 1106,
3168 ["gjcy"] = 1107,
3169 ["jukcy"] = 1108,
3170 ["dscy"] = 1109,
3171 ["iukcy"] = 1110,
3172 ["yicy"] = 1111,
3173 ["jsercy"] = 1112,
3174 ["ljcy"] = 1113,
3175 ["njcy"] = 1114,
3176 ["tshcy"] = 1115,
3177 ["kjcy"] = 1116,
3178 ["ubrcy"] = 1118,
3179 ["dzcy"] = 1119,
3180 ["ensp"] = 8194,
3181 ["emsp"] = 8195,
3182 ["emsp13"] = 8196,
3183 ["emsp14"] = 8197,
3184 ["numsp"] = 8199,
3185 ["puncsp"] = 8200,
3186 ["thinsp"] = 8201,
3187 ["ThinSpace"] = 8201,
3188 ["hairsp"] = 8202,
3189 ["VeryThinSpace"] = 8202,
3190 ["ZeroWidthSpace"] = 8203,
3191 ["NegativeVeryThinSpace"] = 8203,
3192 ["NegativeThinSpace"] = 8203,
3193 ["NegativeMediumSpace"] = 8203,
3194 ["NegativeThickSpace"] = 8203,
3195 ["zwnj"] = 8204,
3196 ["zwj"] = 8205,
3197 ["lrm"] = 8206,
3198 ["rlm"] = 8207,
3199 ["hyphen"] = 8208,
3200 ["dash"] = 8208,
3201 ["ndash"] = 8211,
3202 ["mdash"] = 8212,
3203 ["horbar"] = 8213,
3204 ["Verbar"] = 8214,
3205 ["Vert"] = 8214,
3206 ["lsquo"] = 8216,
3207 ["OpenCurlyQuote"] = 8216,
3208 ["rsquo"] = 8217,
3209 ["rsquor"] = 8217,
3210 ["CloseCurlyQuote"] = 8217,
3211 ["lsquor"] = 8218,
3212 ["sbquo"] = 8218,
3213 ["ldquo"] = 8220,

3214 ["OpenCurlyDoubleQuote"] = 8220,
3215 ["rdquo"] = 8221,
3216 ["rdquor"] = 8221,
3217 ["CloseCurlyDoubleQuote"] = 8221,
3218 ["ldquo"] = 8222,
3219 ["bdquo"] = 8222,
3220 ["dagger"] = 8224,
3221 ["Dagger"] = 8225,
3222 ["ddagger"] = 8225,
3223 ["bull"] = 8226,
3224 ["bullet"] = 8226,
3225 ["nldr"] = 8229,
3226 ["hellip"] = 8230,
3227 ["mldr"] = 8230,
3228 ["permil"] = 8240,
3229 ["pertenk"] = 8241,
3230 ["prime"] = 8242,
3231 ["Prime"] = 8243,
3232 ["tprime"] = 8244,
3233 ["bprime"] = 8245,
3234 ["backprime"] = 8245,
3235 ["lsaquo"] = 8249,
3236 ["rsaquo"] = 8250,
3237 ["oline"] = 8254,
3238 ["caret"] = 8257,
3239 ["hybull"] = 8259,
3240 ["frasl"] = 8260,
3241 ["bsemi"] = 8271,
3242 ["qprime"] = 8279,
3243 ["MediumSpace"] = 8287,
3244 ["NoBreak"] = 8288,
3245 ["ApplyFunction"] = 8289,
3246 ["af"] = 8289,
3247 ["InvisibleTimes"] = 8290,
3248 ["it"] = 8290,
3249 ["InvisibleComma"] = 8291,
3250 ["ic"] = 8291,
3251 ["euro"] = 8364,
3252 ["tdot"] = 8411,
3253 ["TripleDot"] = 8411,
3254 ["DotDot"] = 8412,
3255 ["Copf"] = 8450,
3256 ["complexes"] = 8450,
3257 ["incare"] = 8453,
3258 ["gscr"] = 8458,
3259 ["hamilt"] = 8459,
3260 ["HilbertSpace"] = 8459,

3261 ["Hscr"] = 8459,
3262 ["Hfr"] = 8460,
3263 ["Poincareplane"] = 8460,
3264 ["quaternions"] = 8461,
3265 ["Hopf"] = 8461,
3266 ["planckh"] = 8462,
3267 ["planck"] = 8463,
3268 ["hbar"] = 8463,
3269 ["plankv"] = 8463,
3270 ["hslash"] = 8463,
3271 ["Iscr"] = 8464,
3272 ["imagline"] = 8464,
3273 ["image"] = 8465,
3274 ["Im"] = 8465,
3275 ["imagpart"] = 8465,
3276 ["Ifr"] = 8465,
3277 ["Lscr"] = 8466,
3278 ["lagran"] = 8466,
3279 ["Laplacetrif"] = 8466,
3280 ["ell"] = 8467,
3281 ["Nopf"] = 8469,
3282 ["naturals"] = 8469,
3283 ["numero"] = 8470,
3284 ["copysr"] = 8471,
3285 ["weierp"] = 8472,
3286 ["wp"] = 8472,
3287 ["Popf"] = 8473,
3288 ["primes"] = 8473,
3289 ["rationals"] = 8474,
3290 ["Qopf"] = 8474,
3291 ["Rscr"] = 8475,
3292 ["realine"] = 8475,
3293 ["real"] = 8476,
3294 ["Re"] = 8476,
3295 ["realpart"] = 8476,
3296 ["Rfr"] = 8476,
3297 ["reals"] = 8477,
3298 ["Ropf"] = 8477,
3299 ["rx"] = 8478,
3300 ["trade"] = 8482,
3301 ["TRADE"] = 8482,
3302 ["integers"] = 8484,
3303 ["Zopf"] = 8484,
3304 ["ohm"] = 8486,
3305 ["mho"] = 8487,
3306 ["Zfr"] = 8488,
3307 ["zeetrf"] = 8488,

3308 ["iiota"] = 8489,
3309 ["angst"] = 8491,
3310 ["bernou"] = 8492,
3311 ["Bernoullis"] = 8492,
3312 ["Bscr"] = 8492,
3313 ["Cfr"] = 8493,
3314 ["Cayleys"] = 8493,
3315 ["escr"] = 8495,
3316 ["Escr"] = 8496,
3317 ["expectation"] = 8496,
3318 ["Fscr"] = 8497,
3319 ["Fouriertrf"] = 8497,
3320 ["phmmat"] = 8499,
3321 ["Mellintrf"] = 8499,
3322 ["Mscr"] = 8499,
3323 ["order"] = 8500,
3324 ["orderof"] = 8500,
3325 ["oscr"] = 8500,
3326 ["alefsym"] = 8501,
3327 ["aleph"] = 8501,
3328 ["beth"] = 8502,
3329 ["gimel"] = 8503,
3330 ["daleth"] = 8504,
3331 ["CapitalDifferentialD"] = 8517,
3332 ["DD"] = 8517,
3333 ["DifferentialD"] = 8518,
3334 ["dd"] = 8518,
3335 ["ExponentialE"] = 8519,
3336 ["exponentiale"] = 8519,
3337 ["ee"] = 8519,
3338 ["ImaginaryI"] = 8520,
3339 ["ii"] = 8520,
3340 ["frac13"] = 8531,
3341 ["frac23"] = 8532,
3342 ["frac15"] = 8533,
3343 ["frac25"] = 8534,
3344 ["frac35"] = 8535,
3345 ["frac45"] = 8536,
3346 ["frac16"] = 8537,
3347 ["frac56"] = 8538,
3348 ["frac18"] = 8539,
3349 ["frac38"] = 8540,
3350 ["frac58"] = 8541,
3351 ["frac78"] = 8542,
3352 ["larr"] = 8592,
3353 ["leftarrow"] = 8592,
3354 ["LeftArrow"] = 8592,

3355 ["slarr"] = 8592,
3356 ["ShortLeftArrow"] = 8592,
3357 ["uarr"] = 8593,
3358 ["uparrow"] = 8593,
3359 ["UpArrow"] = 8593,
3360 ["ShortUpArrow"] = 8593,
3361 ["rarr"] = 8594,
3362 ["rightarrow"] = 8594,
3363 ["RightArrow"] = 8594,
3364 ["srarr"] = 8594,
3365 ["ShortRightArrow"] = 8594,
3366 ["darr"] = 8595,
3367 ["downarrow"] = 8595,
3368 ["DownArrow"] = 8595,
3369 ["ShortDownArrow"] = 8595,
3370 ["harr"] = 8596,
3371 ["leftrightarrow"] = 8596,
3372 ["LeftRightArrow"] = 8596,
3373 ["varr"] = 8597,
3374 ["updownarrow"] = 8597,
3375 ["UpDownArrow"] = 8597,
3376 ["nwarr"] = 8598,
3377 ["UpperLeftArrow"] = 8598,
3378 ["nwarrow"] = 8598,
3379 ["nearr"] = 8599,
3380 ["UpperRightArrow"] = 8599,
3381 ["nearrow"] = 8599,
3382 ["searr"] = 8600,
3383 ["searrow"] = 8600,
3384 ["LowerRightArrow"] = 8600,
3385 ["swarr"] = 8601,
3386 ["swarrow"] = 8601,
3387 ["LowerLeftArrow"] = 8601,
3388 ["nlarr"] = 8602,
3389 ["nleftarrow"] = 8602,
3390 ["nrarr"] = 8603,
3391 ["nrightarrow"] = 8603,
3392 ["rarrw"] = 8605,
3393 ["rightsquigarrow"] = 8605,
3394 ["Larr"] = 8606,
3395 ["twoheadleftarrow"] = 8606,
3396 ["Uarr"] = 8607,
3397 ["Rarr"] = 8608,
3398 ["twoheadrightarrow"] = 8608,
3399 ["Darr"] = 8609,
3400 ["larrtl"] = 8610,
3401 ["leftarrowtail"] = 8610,

3402 ["rarrtl"] = 8611,
3403 ["rightarrowtail"] = 8611,
3404 ["LeftTeeArrow"] = 8612,
3405 ["mapstoleft"] = 8612,
3406 ["UpTeeArrow"] = 8613,
3407 ["mapstoup"] = 8613,
3408 ["map"] = 8614,
3409 ["RightTeeArrow"] = 8614,
3410 ["mapsto"] = 8614,
3411 ["DownTeeArrow"] = 8615,
3412 ["mapstodown"] = 8615,
3413 ["larrhk"] = 8617,
3414 ["hookleftarrow"] = 8617,
3415 ["rarrhk"] = 8618,
3416 ["hookrightarrow"] = 8618,
3417 ["larrlp"] = 8619,
3418 ["looparrowleft"] = 8619,
3419 ["rarrlp"] = 8620,
3420 ["looparrowright"] = 8620,
3421 ["harrw"] = 8621,
3422 ["leftrightsquigarrow"] = 8621,
3423 ["nharr"] = 8622,
3424 ["nletrightarrow"] = 8622,
3425 ["lsh"] = 8624,
3426 ["Lsh"] = 8624,
3427 ["rsh"] = 8625,
3428 ["Rsh"] = 8625,
3429 ["ldsh"] = 8626,
3430 ["rdsh"] = 8627,
3431 ["crarr"] = 8629,
3432 ["cularr"] = 8630,
3433 ["curvearrowleft"] = 8630,
3434 ["curarr"] = 8631,
3435 ["curvearrowright"] = 8631,
3436 ["olarr"] = 8634,
3437 ["circlearrowleft"] = 8634,
3438 ["orarr"] = 8635,
3439 ["circlearrowright"] = 8635,
3440 ["lharu"] = 8636,
3441 ["LeftVector"] = 8636,
3442 ["leftharpoonup"] = 8636,
3443 ["lhard"] = 8637,
3444 ["leftharpoondown"] = 8637,
3445 ["DownLeftVector"] = 8637,
3446 ["uharr"] = 8638,
3447 ["upharpoonright"] = 8638,
3448 ["RightUpVector"] = 8638,

3449 ["uharl"] = 8639,
 3450 ["upharpoonleft"] = 8639,
 3451 ["LeftUpVector"] = 8639,
 3452 ["rharu"] = 8640,
 3453 ["RightVector"] = 8640,
 3454 ["rightharpoonup"] = 8640,
 3455 ["rhard"] = 8641,
 3456 ["rightharpoondown"] = 8641,
 3457 ["DownRightVector"] = 8641,
 3458 ["dharr"] = 8642,
 3459 ["RightDownVector"] = 8642,
 3460 ["downharpoonright"] = 8642,
 3461 ["dharl"] = 8643,
 3462 ["LeftDownVector"] = 8643,
 3463 ["downharpoonleft"] = 8643,
 3464 ["rlarr"] = 8644,
 3465 ["rightleftarrows"] = 8644,
 3466 ["RightArrowLeftArrow"] = 8644,
 3467 ["udarr"] = 8645,
 3468 ["UpArrowDownArrow"] = 8645,
 3469 ["lrarr"] = 8646,
 3470 ["leftrightarrows"] = 8646,
 3471 ["LeftArrowRightArrow"] = 8646,
 3472 ["llarr"] = 8647,
 3473 ["leftleftarrows"] = 8647,
 3474 ["uuarr"] = 8648,
 3475 ["upuparrows"] = 8648,
 3476 ["rrarr"] = 8649,
 3477 ["rightrightarrows"] = 8649,
 3478 ["ddarr"] = 8650,
 3479 ["downdownarrows"] = 8650,
 3480 ["lrhar"] = 8651,
 3481 ["ReverseEquilibrium"] = 8651,
 3482 ["leftrightharpoons"] = 8651,
 3483 ["rlhar"] = 8652,
 3484 ["rightleftharpoons"] = 8652,
 3485 ["Equilibrium"] = 8652,
 3486 ["nlArr"] = 8653,
 3487 ["nLeftarrow"] = 8653,
 3488 ["nhArr"] = 8654,
 3489 ["nLeftrightarrow"] = 8654,
 3490 ["nrArr"] = 8655,
 3491 ["nRightarrow"] = 8655,
 3492 ["lArr"] = 8656,
 3493 ["Leftarrow"] = 8656,
 3494 ["DoubleLeftArrow"] = 8656,
 3495 ["uArr"] = 8657,

3496 ["Uparrow"] = 8657,
3497 ["DoubleUpArrow"] = 8657,
3498 ["rArr"] = 8658,
3499 ["Rightarrow"] = 8658,
3500 ["Implies"] = 8658,
3501 ["DoubleRightArrow"] = 8658,
3502 ["dArr"] = 8659,
3503 ["Downarrow"] = 8659,
3504 ["DoubleDownArrow"] = 8659,
3505 ["hArr"] = 8660,
3506 ["Leftrightarrow"] = 8660,
3507 ["DoubleLeftRightArrow"] = 8660,
3508 ["iff"] = 8660,
3509 ["vArr"] = 8661,
3510 ["Updownarrow"] = 8661,
3511 ["DoubleUpDownArrow"] = 8661,
3512 ["nwArr"] = 8662,
3513 ["neArr"] = 8663,
3514 ["seArr"] = 8664,
3515 ["swArr"] = 8665,
3516 ["lAarr"] = 8666,
3517 ["Lleftarrow"] = 8666,
3518 ["rAarr"] = 8667,
3519 ["Rrightarrow"] = 8667,
3520 ["zigrarr"] = 8669,
3521 ["larrb"] = 8676,
3522 ["LeftArrowBar"] = 8676,
3523 ["rarrb"] = 8677,
3524 ["RightArrowBar"] = 8677,
3525 ["duarr"] = 8693,
3526 ["DownArrowUpArrow"] = 8693,
3527 ["loarr"] = 8701,
3528 ["roarr"] = 8702,
3529 ["hoarr"] = 8703,
3530 ["forall"] = 8704,
3531 ["ForAll"] = 8704,
3532 ["comp"] = 8705,
3533 ["complement"] = 8705,
3534 ["part"] = 8706,
3535 ["PartialD"] = 8706,
3536 ["exist"] = 8707,
3537 ["Exists"] = 8707,
3538 ["nexist"] = 8708,
3539 ["NotExists"] = 8708,
3540 ["nexists"] = 8708,
3541 ["empty"] = 8709,
3542 ["emptyset"] = 8709,

3543 ["emptyv"] = 8709,
3544 ["varnothing"] = 8709,
3545 ["nabla"] = 8711,
3546 ["Del"] = 8711,
3547 ["isin"] = 8712,
3548 ["isinv"] = 8712,
3549 ["Element"] = 8712,
3550 ["in"] = 8712,
3551 ["notin"] = 8713,
3552 ["NotElement"] = 8713,
3553 ["notinva"] = 8713,
3554 ["niv"] = 8715,
3555 ["ReverseElement"] = 8715,
3556 ["ni"] = 8715,
3557 ["SuchThat"] = 8715,
3558 ["notni"] = 8716,
3559 ["notniva"] = 8716,
3560 ["NotReverseElement"] = 8716,
3561 ["prod"] = 8719,
3562 ["Product"] = 8719,
3563 ["coprod"] = 8720,
3564 ["Coproduct"] = 8720,
3565 ["sum"] = 8721,
3566 ["Sum"] = 8721,
3567 ["minus"] = 8722,
3568 ["mnplus"] = 8723,
3569 ["mp"] = 8723,
3570 ["MinusPlus"] = 8723,
3571 ["plusdo"] = 8724,
3572 ["dotplus"] = 8724,
3573 ["setmn"] = 8726,
3574 ["setminus"] = 8726,
3575 ["Backslash"] = 8726,
3576 ["ssetmn"] = 8726,
3577 ["smallsetminus"] = 8726,
3578 ["lowast"] = 8727,
3579 ["compfn"] = 8728,
3580 ["SmallCircle"] = 8728,
3581 ["radic"] = 8730,
3582 ["Sqrt"] = 8730,
3583 ["prop"] = 8733,
3584 ["propto"] = 8733,
3585 ["Proportional"] = 8733,
3586 ["vprop"] = 8733,
3587 ["varpropto"] = 8733,
3588 ["infin"] = 8734,
3589 ["angrt"] = 8735,

3590 ["ang"] = 8736,
3591 ["angle"] = 8736,
3592 ["angmsd"] = 8737,
3593 ["measuredangle"] = 8737,
3594 ["angsph"] = 8738,
3595 ["mid"] = 8739,
3596 ["VerticalBar"] = 8739,
3597 ["smid"] = 8739,
3598 ["shortmid"] = 8739,
3599 ["nmid"] = 8740,
3600 ["NotVerticalBar"] = 8740,
3601 ["nsmid"] = 8740,
3602 ["nshortmid"] = 8740,
3603 ["par"] = 8741,
3604 ["parallel"] = 8741,
3605 ["DoubleVerticalBar"] = 8741,
3606 ["spar"] = 8741,
3607 ["shortparallel"] = 8741,
3608 ["npar"] = 8742,
3609 ["nparallel"] = 8742,
3610 ["NotDoubleVerticalBar"] = 8742,
3611 ["nspar"] = 8742,
3612 ["nshortparallel"] = 8742,
3613 ["and"] = 8743,
3614 ["wedge"] = 8743,
3615 ["or"] = 8744,
3616 ["vee"] = 8744,
3617 ["cap"] = 8745,
3618 ["cup"] = 8746,
3619 ["int"] = 8747,
3620 ["Integral"] = 8747,
3621 ["Int"] = 8748,
3622 ["tint"] = 8749,
3623 ["iiint"] = 8749,
3624 ["conint"] = 8750,
3625 ["oint"] = 8750,
3626 ["ContourIntegral"] = 8750,
3627 ["Conint"] = 8751,
3628 ["DoubleContourIntegral"] = 8751,
3629 ["Cconint"] = 8752,
3630 ["cwint"] = 8753,
3631 ["cwconint"] = 8754,
3632 ["ClockwiseContourIntegral"] = 8754,
3633 ["awconint"] = 8755,
3634 ["CounterClockwiseContourIntegral"] = 8755,
3635 ["there4"] = 8756,
3636 ["therefore"] = 8756,

3637 ["Therefore"] = 8756,
3638 ["becaus"] = 8757,
3639 ["because"] = 8757,
3640 ["Because"] = 8757,
3641 ["ratio"] = 8758,
3642 ["Colon"] = 8759,
3643 ["Proportion"] = 8759,
3644 ["minusd"] = 8760,
3645 ["dotminus"] = 8760,
3646 ["mDDot"] = 8762,
3647 ["homtht"] = 8763,
3648 ["sim"] = 8764,
3649 ["Tilde"] = 8764,
3650 ["thksim"] = 8764,
3651 ["thicksim"] = 8764,
3652 ["bsim"] = 8765,
3653 ["backsim"] = 8765,
3654 ["ac"] = 8766,
3655 ["mstpos"] = 8766,
3656 ["acd"] = 8767,
3657 ["wreath"] = 8768,
3658 ["VerticalTilde"] = 8768,
3659 ["wr"] = 8768,
3660 ["nsim"] = 8769,
3661 ["NotTilde"] = 8769,
3662 ["esim"] = 8770,
3663 ["EqualTilde"] = 8770,
3664 ["eqsim"] = 8770,
3665 ["sime"] = 8771,
3666 ["TildeEqual"] = 8771,
3667 ["simeq"] = 8771,
3668 ["nsime"] = 8772,
3669 ["nsimeq"] = 8772,
3670 ["NotTildeEqual"] = 8772,
3671 ["cong"] = 8773,
3672 ["TildeFullEqual"] = 8773,
3673 ["simne"] = 8774,
3674 ["ncong"] = 8775,
3675 ["NotTildeFullEqual"] = 8775,
3676 ["asyp"] = 8776,
3677 ["ap"] = 8776,
3678 ["TildeTilde"] = 8776,
3679 ["approx"] = 8776,
3680 ["thkap"] = 8776,
3681 ["thickapprox"] = 8776,
3682 ["nap"] = 8777,
3683 ["NotTildeTilde"] = 8777,

3684 ["napprox"] = 8777,
 3685 ["ape"] = 8778,
 3686 ["approxpeq"] = 8778,
 3687 ["apid"] = 8779,
 3688 ["bcong"] = 8780,
 3689 ["backcong"] = 8780,
 3690 ["asympeq"] = 8781,
 3691 ["CupCap"] = 8781,
 3692 ["bump"] = 8782,
 3693 ["HumpDownHump"] = 8782,
 3694 ["Bumpeq"] = 8782,
 3695 ["bumpe"] = 8783,
 3696 ["HumpEqual"] = 8783,
 3697 ["bumpeq"] = 8783,
 3698 ["esdot"] = 8784,
 3699 ["DotEqual"] = 8784,
 3700 ["doteq"] = 8784,
 3701 ["eDot"] = 8785,
 3702 ["doteqdot"] = 8785,
 3703 ["efDot"] = 8786,
 3704 ["fallingdotseq"] = 8786,
 3705 ["erDot"] = 8787,
 3706 ["risingdotseq"] = 8787,
 3707 ["colone"] = 8788,
 3708 ["coloneq"] = 8788,
 3709 ["Assign"] = 8788,
 3710 ["ecolon"] = 8789,
 3711 ["eqcolon"] = 8789,
 3712 ["ecir"] = 8790,
 3713 ["eqcirc"] = 8790,
 3714 ["cire"] = 8791,
 3715 ["circeq"] = 8791,
 3716 ["wedgeq"] = 8793,
 3717 ["veeeq"] = 8794,
 3718 ["trie"] = 8796,
 3719 ["triangleq"] = 8796,
 3720 ["equest"] = 8799,
 3721 ["questeq"] = 8799,
 3722 ["ne"] = 8800,
 3723 ["NotEqual"] = 8800,
 3724 ["equiv"] = 8801,
 3725 ["Congruent"] = 8801,
 3726 ["nequiv"] = 8802,
 3727 ["NotCongruent"] = 8802,
 3728 ["le"] = 8804,
 3729 ["leq"] = 8804,
 3730 ["ge"] = 8805,

3731 ["GreaterEqual"] = 8805,
3732 ["geq"] = 8805,
3733 ["lE"] = 8806,
3734 ["LessFullEqual"] = 8806,
3735 ["leqq"] = 8806,
3736 ["gE"] = 8807,
3737 ["GreaterFullEqual"] = 8807,
3738 ["geqq"] = 8807,
3739 ["lnE"] = 8808,
3740 ["lneqq"] = 8808,
3741 ["gnE"] = 8809,
3742 ["gneqq"] = 8809,
3743 ["Lt"] = 8810,
3744 ["NestedLessLess"] = 8810,
3745 ["ll"] = 8810,
3746 ["Gt"] = 8811,
3747 ["NestedGreaterGreater"] = 8811,
3748 ["gg"] = 8811,
3749 ["twixt"] = 8812,
3750 ["between"] = 8812,
3751 ["NotCupCap"] = 8813,
3752 ["nlt"] = 8814,
3753 ["NotLess"] = 8814,
3754 ["nless"] = 8814,
3755 ["ngt"] = 8815,
3756 ["NotGreater"] = 8815,
3757 ["ngtr"] = 8815,
3758 ["nle"] = 8816,
3759 ["NotLessEqual"] = 8816,
3760 ["nleq"] = 8816,
3761 ["nge"] = 8817,
3762 ["NotGreaterEqual"] = 8817,
3763 ["ngeq"] = 8817,
3764 ["lsim"] = 8818,
3765 ["LessTilde"] = 8818,
3766 ["lesssim"] = 8818,
3767 ["gsim"] = 8819,
3768 ["gtrsim"] = 8819,
3769 ["GreaterTilde"] = 8819,
3770 ["nlsim"] = 8820,
3771 ["NotLessTilde"] = 8820,
3772 ["ngsim"] = 8821,
3773 ["NotGreaterTilde"] = 8821,
3774 ["lg"] = 8822,
3775 ["lessgtr"] = 8822,
3776 ["LessGreater"] = 8822,
3777 ["gl"] = 8823,

3778 ["gtrless"] = 8823,
3779 ["GreaterLess"] = 8823,
3780 ["ntlg"] = 8824,
3781 ["NotLessGreater"] = 8824,
3782 ["ntgl"] = 8825,
3783 ["NotGreaterLess"] = 8825,
3784 ["pr"] = 8826,
3785 ["Precedes"] = 8826,
3786 ["prec"] = 8826,
3787 ["sc"] = 8827,
3788 ["Succeeds"] = 8827,
3789 ["succ"] = 8827,
3790 ["prcue"] = 8828,
3791 ["PrecedesSlantEqual"] = 8828,
3792 ["preccurlyeq"] = 8828,
3793 ["sccue"] = 8829,
3794 ["SucceedsSlantEqual"] = 8829,
3795 ["succurlyeq"] = 8829,
3796 ["prsim"] = 8830,
3797 ["precsim"] = 8830,
3798 ["PrecedesTilde"] = 8830,
3799 ["scsim"] = 8831,
3800 ["succsim"] = 8831,
3801 ["SucceedsTilde"] = 8831,
3802 ["npr"] = 8832,
3803 ["nprec"] = 8832,
3804 ["NotPrecedes"] = 8832,
3805 ["nsc"] = 8833,
3806 ["nsucc"] = 8833,
3807 ["NotSucceeds"] = 8833,
3808 ["sub"] = 8834,
3809 ["subset"] = 8834,
3810 ["sup"] = 8835,
3811 ["supset"] = 8835,
3812 ["Superset"] = 8835,
3813 ["nsub"] = 8836,
3814 ["nsup"] = 8837,
3815 ["sube"] = 8838,
3816 ["SubsetEqual"] = 8838,
3817 ["subseteq"] = 8838,
3818 ["supe"] = 8839,
3819 ["supseteq"] = 8839,
3820 ["SupersetEqual"] = 8839,
3821 ["nsube"] = 8840,
3822 ["nsubseteq"] = 8840,
3823 ["NotSubsetEqual"] = 8840,
3824 ["nsupe"] = 8841,

3825 ["nsupseteq"] = 8841,
3826 ["NotSupersetEqual"] = 8841,
3827 ["subne"] = 8842,
3828 ["subsetneq"] = 8842,
3829 ["supne"] = 8843,
3830 ["supsetneq"] = 8843,
3831 ["cupdot"] = 8845,
3832 ["uplus"] = 8846,
3833 ["UnionPlus"] = 8846,
3834 ["sqsub"] = 8847,
3835 ["SquareSubset"] = 8847,
3836 ["sqsubset"] = 8847,
3837 ["sqsup"] = 8848,
3838 ["SquareSuperset"] = 8848,
3839 ["sqsupset"] = 8848,
3840 ["sqsube"] = 8849,
3841 ["SquareSubsetEqual"] = 8849,
3842 ["sqsubseteq"] = 8849,
3843 ["sqsupe"] = 8850,
3844 ["SquareSupersetEqual"] = 8850,
3845 ["sqsupseteq"] = 8850,
3846 ["sqcap"] = 8851,
3847 ["SquareIntersection"] = 8851,
3848 ["sqcup"] = 8852,
3849 ["SquareUnion"] = 8852,
3850 ["oplus"] = 8853,
3851 ["CirclePlus"] = 8853,
3852 ["ominus"] = 8854,
3853 ["CircleMinus"] = 8854,
3854 ["otimes"] = 8855,
3855 ["CircleTimes"] = 8855,
3856 ["osol"] = 8856,
3857 ["odot"] = 8857,
3858 ["CircleDot"] = 8857,
3859 ["ocir"] = 8858,
3860 ["circledcirc"] = 8858,
3861 ["oast"] = 8859,
3862 ["circledast"] = 8859,
3863 ["odash"] = 8861,
3864 ["circleddash"] = 8861,
3865 ["plusb"] = 8862,
3866 ["boxplus"] = 8862,
3867 ["minusb"] = 8863,
3868 ["boxminus"] = 8863,
3869 ["timesb"] = 8864,
3870 ["boxtimes"] = 8864,
3871 ["sdotb"] = 8865,

3872 ["dotsquare"] = 8865,
3873 ["vdash"] = 8866,
3874 ["RightTee"] = 8866,
3875 ["dashv"] = 8867,
3876 ["LeftTee"] = 8867,
3877 ["top"] = 8868,
3878 ["DownTee"] = 8868,
3879 ["bottom"] = 8869,
3880 ["bot"] = 8869,
3881 ["perp"] = 8869,
3882 ["UpTee"] = 8869,
3883 ["models"] = 8871,
3884 ["vDash"] = 8872,
3885 ["DoubleRightTee"] = 8872,
3886 ["Vdash"] = 8873,
3887 ["Vvdash"] = 8874,
3888 ["VDash"] = 8875,
3889 ["nvdash"] = 8876,
3890 ["nvDash"] = 8877,
3891 ["nVdash"] = 8878,
3892 ["nVDash"] = 8879,
3893 ["prurel"] = 8880,
3894 ["vltri"] = 8882,
3895 ["vartriangleleft"] = 8882,
3896 ["LeftTriangle"] = 8882,
3897 ["vrtri"] = 8883,
3898 ["vartriangleright"] = 8883,
3899 ["RightTriangle"] = 8883,
3900 ["ltrie"] = 8884,
3901 ["trianglelefteq"] = 8884,
3902 ["LeftTriangleEqual"] = 8884,
3903 ["rtrie"] = 8885,
3904 ["trianglerighteq"] = 8885,
3905 ["RightTriangleEqual"] = 8885,
3906 ["origof"] = 8886,
3907 ["imof"] = 8887,
3908 ["mumap"] = 8888,
3909 ["multimap"] = 8888,
3910 ["hercon"] = 8889,
3911 ["intcal"] = 8890,
3912 ["intercal"] = 8890,
3913 ["veebar"] = 8891,
3914 ["barvee"] = 8893,
3915 ["angrtvb"] = 8894,
3916 ["lrtri"] = 8895,
3917 ["xwedge"] = 8896,
3918 ["Wedge"] = 8896,

3919 ["bigwedge"] = 8896,
3920 ["xvee"] = 8897,
3921 ["Vee"] = 8897,
3922 ["bigvee"] = 8897,
3923 ["xcap"] = 8898,
3924 ["Intersection"] = 8898,
3925 ["bigcap"] = 8898,
3926 ["xcup"] = 8899,
3927 ["Union"] = 8899,
3928 ["bigcup"] = 8899,
3929 ["diam"] = 8900,
3930 ["diamond"] = 8900,
3931 ["Diamond"] = 8900,
3932 ["sdot"] = 8901,
3933 ["sstarf"] = 8902,
3934 ["Star"] = 8902,
3935 ["divonx"] = 8903,
3936 ["divideontimes"] = 8903,
3937 ["bowtie"] = 8904,
3938 ["ltimes"] = 8905,
3939 ["rtimes"] = 8906,
3940 ["lthree"] = 8907,
3941 ["leftthreetimes"] = 8907,
3942 ["rthree"] = 8908,
3943 ["rightthreetimes"] = 8908,
3944 ["bsime"] = 8909,
3945 ["backsimeq"] = 8909,
3946 ["cuvee"] = 8910,
3947 ["curlyvee"] = 8910,
3948 ["cuwed"] = 8911,
3949 ["curlywedge"] = 8911,
3950 ["Sub"] = 8912,
3951 ["Subset"] = 8912,
3952 ["Sup"] = 8913,
3953 ["Supset"] = 8913,
3954 ["Cap"] = 8914,
3955 ["Cup"] = 8915,
3956 ["fork"] = 8916,
3957 ["pitchfork"] = 8916,
3958 ["epar"] = 8917,
3959 ["ltdot"] = 8918,
3960 ["lessdot"] = 8918,
3961 ["gtdot"] = 8919,
3962 ["gtrdot"] = 8919,
3963 ["Ll"] = 8920,
3964 ["Gg"] = 8921,
3965 ["ggg"] = 8921,

3966 ["leg"] = 8922,
 3967 ["LessEqualGreater"] = 8922,
 3968 ["lesseqgtr"] = 8922,
 3969 ["gel"] = 8923,
 3970 ["gtreqless"] = 8923,
 3971 ["GreaterEqualLess"] = 8923,
 3972 ["cuepr"] = 8926,
 3973 ["curlyeqprec"] = 8926,
 3974 ["cuesc"] = 8927,
 3975 ["curlyeqsucc"] = 8927,
 3976 ["nprcue"] = 8928,
 3977 ["NotPrecedesSlantEqual"] = 8928,
 3978 ["nsccue"] = 8929,
 3979 ["NotSucceedsSlantEqual"] = 8929,
 3980 ["nsqsube"] = 8930,
 3981 ["NotSquareSubsetEqual"] = 8930,
 3982 ["nsqsupe"] = 8931,
 3983 ["NotSquareSupersetEqual"] = 8931,
 3984 ["lnsim"] = 8934,
 3985 ["gnsim"] = 8935,
 3986 ["prnsim"] = 8936,
 3987 ["precnsim"] = 8936,
 3988 ["scnsim"] = 8937,
 3989 ["succnsim"] = 8937,
 3990 ["nltri"] = 8938,
 3991 ["ntriangleleft"] = 8938,
 3992 ["NotLeftTriangle"] = 8938,
 3993 ["nrtri"] = 8939,
 3994 ["ntriangleright"] = 8939,
 3995 ["NotRightTriangle"] = 8939,
 3996 ["nltrie"] = 8940,
 3997 ["ntrianglelefteq"] = 8940,
 3998 ["NotLeftTriangleEqual"] = 8940,
 3999 ["nrtrie"] = 8941,
 4000 ["ntrianglerighteq"] = 8941,
 4001 ["NotRightTriangleEqual"] = 8941,
 4002 ["vellip"] = 8942,
 4003 ["ctdot"] = 8943,
 4004 ["utdot"] = 8944,
 4005 ["dtdot"] = 8945,
 4006 ["disin"] = 8946,
 4007 ["isinsv"] = 8947,
 4008 ["isins"] = 8948,
 4009 ["isindot"] = 8949,
 4010 ["notinvc"] = 8950,
 4011 ["notinvb"] = 8951,
 4012 ["isinE"] = 8953,

4013 ["nisd"] = 8954,
4014 ["xnis"] = 8955,
4015 ["nis"] = 8956,
4016 ["notnivc"] = 8957,
4017 ["notnivb"] = 8958,
4018 ["barwed"] = 8965,
4019 ["barwedge"] = 8965,
4020 ["Barwed"] = 8966,
4021 ["doublebarwedge"] = 8966,
4022 ["lceil"] = 8968,
4023 ["LeftCeiling"] = 8968,
4024 ["rceil"] = 8969,
4025 ["RightCeiling"] = 8969,
4026 ["lfloor"] = 8970,
4027 ["LeftFloor"] = 8970,
4028 ["rfloor"] = 8971,
4029 ["RightFloor"] = 8971,
4030 ["drcrop"] = 8972,
4031 ["dlcrop"] = 8973,
4032 ["urcrop"] = 8974,
4033 ["ulcrop"] = 8975,
4034 ["bnot"] = 8976,
4035 ["proflin"] = 8978,
4036 ["profsurf"] = 8979,
4037 ["telrec"] = 8981,
4038 ["target"] = 8982,
4039 ["ulcorn"] = 8988,
4040 ["ulcorner"] = 8988,
4041 ["urcorn"] = 8989,
4042 ["urcorner"] = 8989,
4043 ["dlcorn"] = 8990,
4044 ["llcorner"] = 8990,
4045 ["drcorn"] = 8991,
4046 ["lrcorn"] = 8991,
4047 ["frown"] = 8994,
4048 ["sfrown"] = 8994,
4049 ["smile"] = 8995,
4050 ["ssmile"] = 8995,
4051 ["cylcty"] = 9005,
4052 ["profalar"] = 9006,
4053 ["topbot"] = 9014,
4054 ["ovbar"] = 9021,
4055 ["solbar"] = 9023,
4056 ["angzarr"] = 9084,
4057 ["lmoust"] = 9136,
4058 ["lmoustache"] = 9136,
4059 ["rmoust"] = 9137,

4060 ["rmoustache"] = 9137,
4061 ["tbrk"] = 9140,
4062 ["OverBracket"] = 9140,
4063 ["bbrk"] = 9141,
4064 ["UnderBracket"] = 9141,
4065 ["bbrktbrk"] = 9142,
4066 ["OverParenthesis"] = 9180,
4067 ["UnderParenthesis"] = 9181,
4068 ["OverBrace"] = 9182,
4069 ["UnderBrace"] = 9183,
4070 ["trpezium"] = 9186,
4071 ["elinters"] = 9191,
4072 ["blank"] = 9251,
4073 ["oS"] = 9416,
4074 ["circledS"] = 9416,
4075 ["boxh"] = 9472,
4076 ["HorizontalLine"] = 9472,
4077 ["boxv"] = 9474,
4078 ["boxdr"] = 9484,
4079 ["boxdl"] = 9488,
4080 ["boxur"] = 9492,
4081 ["boxul"] = 9496,
4082 ["boxvr"] = 9500,
4083 ["boxvl"] = 9508,
4084 ["boxhd"] = 9516,
4085 ["boxhu"] = 9524,
4086 ["boxvh"] = 9532,
4087 ["boxH"] = 9552,
4088 ["boxV"] = 9553,
4089 ["boxdR"] = 9554,
4090 ["boxDr"] = 9555,
4091 ["boxDR"] = 9556,
4092 ["boxdL"] = 9557,
4093 ["boxDl"] = 9558,
4094 ["boxDL"] = 9559,
4095 ["boxuR"] = 9560,
4096 ["boxUr"] = 9561,
4097 ["boxUR"] = 9562,
4098 ["boxuL"] = 9563,
4099 ["boxUl"] = 9564,
4100 ["boxUL"] = 9565,
4101 ["boxvR"] = 9566,
4102 ["boxVr"] = 9567,
4103 ["boxVR"] = 9568,
4104 ["boxvL"] = 9569,
4105 ["boxVl"] = 9570,
4106 ["boxVL"] = 9571,

4107 ["boxHd"] = 9572,
4108 ["boxhD"] = 9573,
4109 ["boxHD"] = 9574,
4110 ["boxHu"] = 9575,
4111 ["boxhU"] = 9576,
4112 ["boxHU"] = 9577,
4113 ["boxvH"] = 9578,
4114 ["boxVh"] = 9579,
4115 ["boxVH"] = 9580,
4116 ["uhblk"] = 9600,
4117 ["lhblk"] = 9604,
4118 ["block"] = 9608,
4119 ["blk14"] = 9617,
4120 ["blk12"] = 9618,
4121 ["blk34"] = 9619,
4122 ["squ"] = 9633,
4123 ["square"] = 9633,
4124 ["Square"] = 9633,
4125 ["sqf"] = 9642,
4126 ["squarf"] = 9642,
4127 ["blacksquare"] = 9642,
4128 ["FilledVerySmallSquare"] = 9642,
4129 ["EmptyVerySmallSquare"] = 9643,
4130 ["rect"] = 9645,
4131 ["marker"] = 9646,
4132 ["fltns"] = 9649,
4133 ["xutri"] = 9651,
4134 ["bigtriangleup"] = 9651,
4135 ["utrif"] = 9652,
4136 ["blacktriangle"] = 9652,
4137 ["utri"] = 9653,
4138 ["triangle"] = 9653,
4139 ["rtrif"] = 9656,
4140 ["blacktriangleright"] = 9656,
4141 ["rtri"] = 9657,
4142 ["triangleright"] = 9657,
4143 ["xdtri"] = 9661,
4144 ["bigtriangledown"] = 9661,
4145 ["dtrif"] = 9662,
4146 ["blacktriangledown"] = 9662,
4147 ["dtri"] = 9663,
4148 ["triangledown"] = 9663,
4149 ["ltrif"] = 9666,
4150 ["blacktriangleleft"] = 9666,
4151 ["ltri"] = 9667,
4152 ["triangleleft"] = 9667,
4153 ["loz"] = 9674,

4154 ["lozenge"] = 9674,
4155 ["cir"] = 9675,
4156 ["tridot"] = 9708,
4157 ["xcirc"] = 9711,
4158 ["bigcirc"] = 9711,
4159 ["ultri"] = 9720,
4160 ["urtri"] = 9721,
4161 ["lltri"] = 9722,
4162 ["EmptySmallSquare"] = 9723,
4163 ["FilledSmallSquare"] = 9724,
4164 ["starf"] = 9733,
4165 ["bigstar"] = 9733,
4166 ["star"] = 9734,
4167 ["phone"] = 9742,
4168 ["female"] = 9792,
4169 ["male"] = 9794,
4170 ["spades"] = 9824,
4171 ["spadesuit"] = 9824,
4172 ["clubs"] = 9827,
4173 ["clubsuit"] = 9827,
4174 ["hearts"] = 9829,
4175 ["heartsuit"] = 9829,
4176 ["diams"] = 9830,
4177 ["diamondsuit"] = 9830,
4178 ["sung"] = 9834,
4179 ["flat"] = 9837,
4180 ["natur"] = 9838,
4181 ["natural"] = 9838,
4182 ["sharp"] = 9839,
4183 ["check"] = 10003,
4184 ["checkmark"] = 10003,
4185 ["cross"] = 10007,
4186 ["malt"] = 10016,
4187 ["maltese"] = 10016,
4188 ["sext"] = 10038,
4189 ["VerticalSeparator"] = 10072,
4190 ["lbrk"] = 10098,
4191 ["rbrk"] = 10099,
4192 ["lobrk"] = 10214,
4193 ["LeftDoubleBracket"] = 10214,
4194 ["robrk"] = 10215,
4195 ["RightDoubleBracket"] = 10215,
4196 ["lang"] = 10216,
4197 ["LeftAngleBracket"] = 10216,
4198 ["langle"] = 10216,
4199 ["rang"] = 10217,
4200 ["RightAngleBracket"] = 10217,

4201 ["rangle"] = 10217,
 4202 ["Lang"] = 10218,
 4203 ["Rang"] = 10219,
 4204 ["loang"] = 10220,
 4205 ["roang"] = 10221,
 4206 ["xlarr"] = 10229,
 4207 ["longleftarrow"] = 10229,
 4208 ["LongLeftArrow"] = 10229,
 4209 ["xrarr"] = 10230,
 4210 ["longrightarrow"] = 10230,
 4211 ["LongRightArrow"] = 10230,
 4212 ["xharr"] = 10231,
 4213 ["longleftrightarrow"] = 10231,
 4214 ["LongLeftRightArrow"] = 10231,
 4215 ["xlArr"] = 10232,
 4216 ["Longleftarrow"] = 10232,
 4217 ["DoubleLongLeftArrow"] = 10232,
 4218 ["xrArr"] = 10233,
 4219 ["Longrightarrow"] = 10233,
 4220 ["DoubleLongRightArrow"] = 10233,
 4221 ["xhArr"] = 10234,
 4222 ["Longleftrightarrow"] = 10234,
 4223 ["DoubleLongLeftRightArrow"] = 10234,
 4224 ["xmap"] = 10236,
 4225 ["longmapsto"] = 10236,
 4226 ["dzigrarr"] = 10239,
 4227 ["nvlArr"] = 10498,
 4228 ["nvrArr"] = 10499,
 4229 ["nvHarr"] = 10500,
 4230 ["Map"] = 10501,
 4231 ["lbarr"] = 10508,
 4232 ["rbarr"] = 10509,
 4233 ["bkarow"] = 10509,
 4234 ["lBarr"] = 10510,
 4235 ["rBarr"] = 10511,
 4236 ["dbkarow"] = 10511,
 4237 ["RBarr"] = 10512,
 4238 ["drbkarow"] = 10512,
 4239 ["DDottrahd"] = 10513,
 4240 ["UpArrowBar"] = 10514,
 4241 ["DownArrowBar"] = 10515,
 4242 ["Rarrtl"] = 10518,
 4243 ["latail"] = 10521,
 4244 ["ratail"] = 10522,
 4245 ["lAtail"] = 10523,
 4246 ["rAtail"] = 10524,
 4247 ["larrfs"] = 10525,

4248 ["rarrfs"] = 10526,
4249 ["larrbfs"] = 10527,
4250 ["rarrbfs"] = 10528,
4251 ["nwarhk"] = 10531,
4252 ["nearhk"] = 10532,
4253 ["searhk"] = 10533,
4254 ["hksearow"] = 10533,
4255 ["swarhk"] = 10534,
4256 ["hkswarow"] = 10534,
4257 ["nwnear"] = 10535,
4258 ["nesear"] = 10536,
4259 ["toea"] = 10536,
4260 ["seswar"] = 10537,
4261 ["tosa"] = 10537,
4262 ["swnwar"] = 10538,
4263 ["rarrc"] = 10547,
4264 ["cudarr"] = 10549,
4265 ["ldca"] = 10550,
4266 ["rdca"] = 10551,
4267 ["cudarrl"] = 10552,
4268 ["larrpl"] = 10553,
4269 ["curarrm"] = 10556,
4270 ["cularrp"] = 10557,
4271 ["rarrpl"] = 10565,
4272 ["harrcir"] = 10568,
4273 ["Uarrocir"] = 10569,
4274 ["lurdshar"] = 10570,
4275 ["ldrushar"] = 10571,
4276 ["LeftRightVector"] = 10574,
4277 ["RightUpDownVector"] = 10575,
4278 ["DownLeftRightVector"] = 10576,
4279 ["LeftUpDownVector"] = 10577,
4280 ["LeftVectorBar"] = 10578,
4281 ["RightVectorBar"] = 10579,
4282 ["RightUpVectorBar"] = 10580,
4283 ["RightDownVectorBar"] = 10581,
4284 ["DownLeftVectorBar"] = 10582,
4285 ["DownRightVectorBar"] = 10583,
4286 ["LeftUpVectorBar"] = 10584,
4287 ["LeftDownVectorBar"] = 10585,
4288 ["LeftTeeVector"] = 10586,
4289 ["RightTeeVector"] = 10587,
4290 ["RightUpTeeVector"] = 10588,
4291 ["RightDownTeeVector"] = 10589,
4292 ["DownLeftTeeVector"] = 10590,
4293 ["DownRightTeeVector"] = 10591,
4294 ["LeftUpTeeVector"] = 10592,

4295 ["LeftDownTeeVector"] = 10593,
4296 ["lHar"] = 10594,
4297 ["uHar"] = 10595,
4298 ["rHar"] = 10596,
4299 ["dHar"] = 10597,
4300 ["luruhar"] = 10598,
4301 ["ldrdhar"] = 10599,
4302 ["ruluhar"] = 10600,
4303 ["rdldhar"] = 10601,
4304 ["lharul"] = 10602,
4305 ["llhard"] = 10603,
4306 ["rharul"] = 10604,
4307 ["lrhard"] = 10605,
4308 ["udhar"] = 10606,
4309 ["UpEquilibrium"] = 10606,
4310 ["duhar"] = 10607,
4311 ["ReverseUpEquilibrium"] = 10607,
4312 ["RoundImplies"] = 10608,
4313 ["erarr"] = 10609,
4314 ["simrarr"] = 10610,
4315 ["larrsim"] = 10611,
4316 ["rarrsim"] = 10612,
4317 ["rarrap"] = 10613,
4318 ["ltlarr"] = 10614,
4319 ["gtrarr"] = 10616,
4320 ["subrarr"] = 10617,
4321 ["suplarr"] = 10619,
4322 ["lfisht"] = 10620,
4323 ["rfisht"] = 10621,
4324 ["ufisht"] = 10622,
4325 ["dfisht"] = 10623,
4326 ["lopar"] = 10629,
4327 ["ropar"] = 10630,
4328 ["lbrke"] = 10635,
4329 ["rbrke"] = 10636,
4330 ["lbrkslu"] = 10637,
4331 ["rbrksld"] = 10638,
4332 ["lbrksld"] = 10639,
4333 ["rbrkslu"] = 10640,
4334 ["langd"] = 10641,
4335 ["rangd"] = 10642,
4336 ["lparlt"] = 10643,
4337 ["rpargt"] = 10644,
4338 ["gtlPar"] = 10645,
4339 ["ltrPar"] = 10646,
4340 ["vzigzag"] = 10650,
4341 ["vangrt"] = 10652,

4342 ["angrtvbd"] = 10653,
4343 ["ange"] = 10660,
4344 ["range"] = 10661,
4345 ["dwangle"] = 10662,
4346 ["uwangle"] = 10663,
4347 ["angmsdaa"] = 10664,
4348 ["angmsdab"] = 10665,
4349 ["angmsdac"] = 10666,
4350 ["angmsdad"] = 10667,
4351 ["angmsdae"] = 10668,
4352 ["angmsdaf"] = 10669,
4353 ["angmsdag"] = 10670,
4354 ["angmsdah"] = 10671,
4355 ["bemptyv"] = 10672,
4356 ["demptyv"] = 10673,
4357 ["cemptyv"] = 10674,
4358 ["raemptyv"] = 10675,
4359 ["laemptyv"] = 10676,
4360 ["ohbar"] = 10677,
4361 ["omid"] = 10678,
4362 ["opar"] = 10679,
4363 ["operp"] = 10681,
4364 ["olcross"] = 10683,
4365 ["odsold"] = 10684,
4366 ["olcir"] = 10686,
4367 ["ofcir"] = 10687,
4368 ["olt"] = 10688,
4369 ["ogt"] = 10689,
4370 ["cirscir"] = 10690,
4371 ["cirE"] = 10691,
4372 ["solb"] = 10692,
4373 ["bsolb"] = 10693,
4374 ["boxbox"] = 10697,
4375 ["trisb"] = 10701,
4376 ["rtriltri"] = 10702,
4377 ["LeftTriangleBar"] = 10703,
4378 ["RightTriangleBar"] = 10704,
4379 ["race"] = 10714,
4380 ["iinfin"] = 10716,
4381 ["infintie"] = 10717,
4382 ["nvinfin"] = 10718,
4383 ["eparsl"] = 10723,
4384 ["smeparsl"] = 10724,
4385 ["eqvparsl"] = 10725,
4386 ["lozf"] = 10731,
4387 ["blacklozenge"] = 10731,
4388 ["RuleDelayed"] = 10740,

4389 ["dsol"] = 10742,
4390 ["xodot"] = 10752,
4391 ["bigodot"] = 10752,
4392 ["xoplus"] = 10753,
4393 ["bigoplus"] = 10753,
4394 ["xotime"] = 10754,
4395 ["bigotimes"] = 10754,
4396 ["xuplus"] = 10756,
4397 ["biguplus"] = 10756,
4398 ["xsqcup"] = 10758,
4399 ["bigsqcup"] = 10758,
4400 ["qint"] = 10764,
4401 ["iiiint"] = 10764,
4402 ["fpartint"] = 10765,
4403 ["cirfnint"] = 10768,
4404 ["awint"] = 10769,
4405 ["rppolint"] = 10770,
4406 ["scpolint"] = 10771,
4407 ["npolint"] = 10772,
4408 ["pointint"] = 10773,
4409 ["quatint"] = 10774,
4410 ["intlarhk"] = 10775,
4411 ["pluscir"] = 10786,
4412 ["plusacir"] = 10787,
4413 ["simplus"] = 10788,
4414 ["plusdu"] = 10789,
4415 ["plussim"] = 10790,
4416 ["plustwo"] = 10791,
4417 ["mcomma"] = 10793,
4418 ["minusdu"] = 10794,
4419 ["loplus"] = 10797,
4420 ["roplus"] = 10798,
4421 ["Cross"] = 10799,
4422 ["timesd"] = 10800,
4423 ["timesbar"] = 10801,
4424 ["smashp"] = 10803,
4425 ["lotimes"] = 10804,
4426 ["rotimes"] = 10805,
4427 ["otimesas"] = 10806,
4428 ["Otimes"] = 10807,
4429 ["odiv"] = 10808,
4430 ["triplus"] = 10809,
4431 ["triminus"] = 10810,
4432 ["tritime"] = 10811,
4433 ["iproduct"] = 10812,
4434 ["intprod"] = 10812,
4435 ["amalg"] = 10815,

4436 ["capdot"] = 10816,
 4437 ["ncup"] = 10818,
 4438 ["ncap"] = 10819,
 4439 ["capand"] = 10820,
 4440 ["cupor"] = 10821,
 4441 ["cupcap"] = 10822,
 4442 ["capcup"] = 10823,
 4443 ["cupbrcap"] = 10824,
 4444 ["capbrcup"] = 10825,
 4445 ["cupcup"] = 10826,
 4446 ["capcap"] = 10827,
 4447 ["ccups"] = 10828,
 4448 ["ccaps"] = 10829,
 4449 ["ccupssm"] = 10832,
 4450 ["And"] = 10835,
 4451 ["Or"] = 10836,
 4452 ["andand"] = 10837,
 4453 ["oror"] = 10838,
 4454 ["orslope"] = 10839,
 4455 ["andslope"] = 10840,
 4456 ["andv"] = 10842,
 4457 ["orv"] = 10843,
 4458 ["andd"] = 10844,
 4459 ["ord"] = 10845,
 4460 ["wedbar"] = 10847,
 4461 ["sdote"] = 10854,
 4462 ["simdot"] = 10858,
 4463 ["congdot"] = 10861,
 4464 ["easter"] = 10862,
 4465 ["apacir"] = 10863,
 4466 ["apE"] = 10864,
 4467 ["eplus"] = 10865,
 4468 ["pluse"] = 10866,
 4469 ["Esim"] = 10867,
 4470 ["Colone"] = 10868,
 4471 ["Equal"] = 10869,
 4472 ["eDDot"] = 10871,
 4473 ["ddotseq"] = 10871,
 4474 ["equivDD"] = 10872,
 4475 ["ltcir"] = 10873,
 4476 ["gtcir"] = 10874,
 4477 ["ltquest"] = 10875,
 4478 ["gtquest"] = 10876,
 4479 ["les"] = 10877,
 4480 ["LessSlantEqual"] = 10877,
 4481 ["leqslant"] = 10877,
 4482 ["ges"] = 10878,

4483 ["GreaterSlantEqual"] = 10878,
4484 ["geqslant"] = 10878,
4485 ["lesdot"] = 10879,
4486 ["gesdot"] = 10880,
4487 ["lesdoto"] = 10881,
4488 ["gesdoto"] = 10882,
4489 ["lesdotor"] = 10883,
4490 ["gesdoto1"] = 10884,
4491 ["lap"] = 10885,
4492 ["lessapprox"] = 10885,
4493 ["gap"] = 10886,
4494 ["gtrapprox"] = 10886,
4495 ["lne"] = 10887,
4496 ["lneq"] = 10887,
4497 ["gne"] = 10888,
4498 ["gneq"] = 10888,
4499 ["lnap"] = 10889,
4500 ["lnapprox"] = 10889,
4501 ["gnap"] = 10890,
4502 ["gnapprox"] = 10890,
4503 ["lEg"] = 10891,
4504 ["lesseqgtr"] = 10891,
4505 ["gE1"] = 10892,
4506 ["gtreqqlless"] = 10892,
4507 ["lsime"] = 10893,
4508 ["gsime"] = 10894,
4509 ["lsimg"] = 10895,
4510 ["gsiml"] = 10896,
4511 ["lgE"] = 10897,
4512 ["glE"] = 10898,
4513 ["lesges"] = 10899,
4514 ["gesles"] = 10900,
4515 ["els"] = 10901,
4516 ["eqslantless"] = 10901,
4517 ["egs"] = 10902,
4518 ["eqslantgtr"] = 10902,
4519 ["elsdot"] = 10903,
4520 ["egsdot"] = 10904,
4521 ["el"] = 10905,
4522 ["eg"] = 10906,
4523 ["siml"] = 10909,
4524 ["simg"] = 10910,
4525 ["simlE"] = 10911,
4526 ["simgE"] = 10912,
4527 ["LessLess"] = 10913,
4528 ["GreaterGreater"] = 10914,
4529 ["glj"] = 10916,

4530 ["gla"] = 10917,
 4531 ["ltcc"] = 10918,
 4532 ["gtcc"] = 10919,
 4533 ["lescc"] = 10920,
 4534 ["gescc"] = 10921,
 4535 ["smt"] = 10922,
 4536 ["lat"] = 10923,
 4537 ["smte"] = 10924,
 4538 ["late"] = 10925,
 4539 ["bumpE"] = 10926,
 4540 ["pre"] = 10927,
 4541 ["preceq"] = 10927,
 4542 ["PrecedesEqual"] = 10927,
 4543 ["sce"] = 10928,
 4544 ["succeq"] = 10928,
 4545 ["SucceedsEqual"] = 10928,
 4546 ["prE"] = 10931,
 4547 ["scE"] = 10932,
 4548 ["prnE"] = 10933,
 4549 ["precneqq"] = 10933,
 4550 ["scnE"] = 10934,
 4551 ["succneqq"] = 10934,
 4552 ["prap"] = 10935,
 4553 ["precapprox"] = 10935,
 4554 ["scap"] = 10936,
 4555 ["succapprox"] = 10936,
 4556 ["prnap"] = 10937,
 4557 ["precnapprox"] = 10937,
 4558 ["scnap"] = 10938,
 4559 ["succnapprox"] = 10938,
 4560 ["Pr"] = 10939,
 4561 ["Sc"] = 10940,
 4562 ["subdot"] = 10941,
 4563 ["supdot"] = 10942,
 4564 ["subplus"] = 10943,
 4565 ["supplus"] = 10944,
 4566 ["submult"] = 10945,
 4567 ["supmult"] = 10946,
 4568 ["subedot"] = 10947,
 4569 ["supedot"] = 10948,
 4570 ["subE"] = 10949,
 4571 ["subseteqq"] = 10949,
 4572 ["supE"] = 10950,
 4573 ["supseteqq"] = 10950,
 4574 ["subsim"] = 10951,
 4575 ["supsim"] = 10952,
 4576 ["subnE"] = 10955,

4577 ["subsetneqq"] = 10955,
 4578 ["supnE"] = 10956,
 4579 ["supsetneqq"] = 10956,
 4580 ["csub"] = 10959,
 4581 ["csup"] = 10960,
 4582 ["csube"] = 10961,
 4583 ["csupe"] = 10962,
 4584 ["subsup"] = 10963,
 4585 ["supsub"] = 10964,
 4586 ["subsub"] = 10965,
 4587 ["supsup"] = 10966,
 4588 ["suphsub"] = 10967,
 4589 ["supdsub"] = 10968,
 4590 ["forkv"] = 10969,
 4591 ["topfork"] = 10970,
 4592 ["mlcp"] = 10971,
 4593 ["Dashv"] = 10980,
 4594 ["DoubleLeftTee"] = 10980,
 4595 ["Vdashl"] = 10982,
 4596 ["Barv"] = 10983,
 4597 ["vBar"] = 10984,
 4598 ["vBarv"] = 10985,
 4599 ["Vbar"] = 10987,
 4600 ["Not"] = 10988,
 4601 ["bNot"] = 10989,
 4602 ["rnmid"] = 10990,
 4603 ["cirmid"] = 10991,
 4604 ["midcir"] = 10992,
 4605 ["topcir"] = 10993,
 4606 ["nhpar"] = 10994,
 4607 ["parsim"] = 10995,
 4608 ["parsl"] = 11005,
 4609 ["fflig"] = 64256,
 4610 ["filig"] = 64257,
 4611 ["fllig"] = 64258,
 4612 ["ffilig"] = 64259,
 4613 ["fflilig"] = 64260,
 4614 ["Ascr"] = 119964,
 4615 ["Cscr"] = 119966,
 4616 ["Dscr"] = 119967,
 4617 ["Gscr"] = 119970,
 4618 ["Jscr"] = 119973,
 4619 ["Kscr"] = 119974,
 4620 ["Nscr"] = 119977,
 4621 ["Oscr"] = 119978,
 4622 ["Pscr"] = 119979,
 4623 ["Qscr"] = 119980,

4624 ["Sscr"] = 119982,
4625 ["Tscr"] = 119983,
4626 ["Uscr"] = 119984,
4627 ["Vscr"] = 119985,
4628 ["Wscr"] = 119986,
4629 ["Xscr"] = 119987,
4630 ["Yscr"] = 119988,
4631 ["Zscr"] = 119989,
4632 ["ascr"] = 119990,
4633 ["bscr"] = 119991,
4634 ["cscr"] = 119992,
4635 ["dscr"] = 119993,
4636 ["fscr"] = 119995,
4637 ["hscr"] = 119997,
4638 ["iscr"] = 119998,
4639 ["jscr"] = 119999,
4640 ["kscr"] = 120000,
4641 ["lscr"] = 120001,
4642 ["mscr"] = 120002,
4643 ["nscr"] = 120003,
4644 ["pscr"] = 120005,
4645 ["qscr"] = 120006,
4646 ["rscr"] = 120007,
4647 ["sscr"] = 120008,
4648 ["tscr"] = 120009,
4649 ["uscr"] = 120010,
4650 ["vscr"] = 120011,
4651 ["wscr"] = 120012,
4652 ["xscr"] = 120013,
4653 ["yscr"] = 120014,
4654 ["zscr"] = 120015,
4655 ["Afr"] = 120068,
4656 ["Bfr"] = 120069,
4657 ["Dfr"] = 120071,
4658 ["Efr"] = 120072,
4659 ["Ffr"] = 120073,
4660 ["Gfr"] = 120074,
4661 ["Jfr"] = 120077,
4662 ["Kfr"] = 120078,
4663 ["Lfr"] = 120079,
4664 ["Mfr"] = 120080,
4665 ["Nfr"] = 120081,
4666 ["Ofr"] = 120082,
4667 ["Pfr"] = 120083,
4668 ["Qfr"] = 120084,
4669 ["Sfr"] = 120086,
4670 ["Tfr"] = 120087,

4671 ["Ufr"] = 120088,
4672 ["Vfr"] = 120089,
4673 ["Wfr"] = 120090,
4674 ["Xfr"] = 120091,
4675 ["Yfr"] = 120092,
4676 ["afr"] = 120094,
4677 ["bfr"] = 120095,
4678 ["cfr"] = 120096,
4679 ["dfr"] = 120097,
4680 ["efr"] = 120098,
4681 ["ffr"] = 120099,
4682 ["gfr"] = 120100,
4683 ["hfr"] = 120101,
4684 ["ifr"] = 120102,
4685 ["jfr"] = 120103,
4686 ["kfr"] = 120104,
4687 ["lfr"] = 120105,
4688 ["mfr"] = 120106,
4689 ["nfr"] = 120107,
4690 ["ofr"] = 120108,
4691 ["pfr"] = 120109,
4692 ["qfr"] = 120110,
4693 ["rfr"] = 120111,
4694 ["sfr"] = 120112,
4695 ["tfr"] = 120113,
4696 ["ufr"] = 120114,
4697 ["vfr"] = 120115,
4698 ["wfr"] = 120116,
4699 ["xfr"] = 120117,
4700 ["yfr"] = 120118,
4701 ["zfr"] = 120119,
4702 ["Aopf"] = 120120,
4703 ["Bopf"] = 120121,
4704 ["Dopf"] = 120123,
4705 ["Eopf"] = 120124,
4706 ["Fopf"] = 120125,
4707 ["Gopf"] = 120126,
4708 ["Iopf"] = 120128,
4709 ["Jopf"] = 120129,
4710 ["Kopf"] = 120130,
4711 ["Lopf"] = 120131,
4712 ["Mopf"] = 120132,
4713 ["Oopf"] = 120134,
4714 ["Sopf"] = 120138,
4715 ["Topf"] = 120139,
4716 ["Uopf"] = 120140,
4717 ["Vopf"] = 120141,

```

4718 ["Wopf"] = 120142,
4719 ["Xopf"] = 120143,
4720 ["Yopf"] = 120144,
4721 ["aopf"] = 120146,
4722 ["bopf"] = 120147,
4723 ["copf"] = 120148,
4724 ["dopf"] = 120149,
4725 ["eopf"] = 120150,
4726 ["fopf"] = 120151,
4727 ["gopf"] = 120152,
4728 ["hopf"] = 120153,
4729 ["iopf"] = 120154,
4730 ["jopf"] = 120155,
4731 ["kopf"] = 120156,
4732 ["lopf"] = 120157,
4733 ["mopf"] = 120158,
4734 ["nopf"] = 120159,
4735 ["oopf"] = 120160,
4736 ["popf"] = 120161,
4737 ["qopf"] = 120162,
4738 ["ropf"] = 120163,
4739 ["sopf"] = 120164,
4740 ["topf"] = 120165,
4741 ["uopf"] = 120166,
4742 ["vopf"] = 120167,
4743 ["wopf"] = 120168,
4744 ["xopf"] = 120169,
4745 ["yopf"] = 120170,
4746 ["zopf"] = 120171,
4747 }

```

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

```

4748 function entities.dec_entity(s)
4749   return unicode.utf8.char(tonumber(s))
4750 end

```

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

```

4751 function entities.hex_entity(s)
4752   return unicode.utf8.char(tonumber("0x"..s))
4753 end

```

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

```

4754 function entities.char_entity(s)
4755   local n = character_entities[s]
4756   if n == nil then

```

```

4757     return "&" .. s .. ";"
4758 end
4759 return unicode.utf8.char(n)
4760 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 Lunamark Lua module.

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

```

4761 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.

```

4762 function M.writer.new(options)
4763     local self = {}

```

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

```

4764     self.options = options

```

Parse the `slice` option and define `writer->slice_begin`, `writer->slice_end`, and `writer->is_writing`. The `writer->is_writing` member variable is mutable.

```

4765     local slice_specifiers = {}
4766     for specifier in options.slice:gmatch("[^%s]+") do
4767         table.insert(slice_specifiers, specifier)
4768     end
4769
4770     if #slice_specifiers == 2 then
4771         self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4772         local slice_begin_type = self.slice_begin:sub(1, 1)
4773         if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4774             self.slice_begin = "^" .. self.slice_begin
4775         end
4776         local slice_end_type = self.slice_end:sub(1, 1)

```

```

4777     if slice_end_type ~= "^" and slice_end_type ~= "$" then
4778         self.slice_end = "$" .. self.slice_end
4779     end
4780 elseif #slice_specifiers == 1 then
4781     self.slice_begin = "^" .. slice_specifiers[1]
4782     self.slice_end = "$" .. slice_specifiers[1]
4783 end
4784
4785 if self.slice_begin == "^" and self.slice_end ~= "^" then
4786     self.is_writing = true
4787 else
4788     self.is_writing = false
4789 end

```

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

```

4790 self.suffix = ".tex"

```

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

```

4791 self.space = " "

```

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

```

4792 self.nbsp = "\\markdownRendererNbsp{}"

```

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

```

4793 function self.plain(s)
4794     return s
4795 end

```

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

```

4796 function self.paragraph(s)
4797     if not self.is_writing then return "" end
4798     return s
4799 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.

```

4800 function self.pack(name)
4801     return [[\input ]] .. name .. [[\relax]]
4802 end

```

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

```

4803 function self.interblocksep()
4804     if not self.is_writing then return "" end
4805     return "\\markdownRendererInterblockSeparator\n{}"
4806 end

```

Define `writer->linebreak` as the output format of a forced line break.

```

4807 self.linebreak = "\\markdownRendererLineBreak\n{}"

```

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

```
4808 self.ellipsis = "\\markdownRendererEllipsis{}
```

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

```
4809 function self.thematic_break()
4810   if not self.is_writing then return "" end
4811   return "\\markdownRendererThematicBreak{}"
4812 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.

```
4813 self.escaped_uri_chars = {
4814   ["{"] = "\\markdownRendererLeftBrace{}",
4815   ["}"] = "\\markdownRendererRightBrace{}",
4816   ["\\"] = "\\markdownRendererBackslash{}",
4817 }
4818 self.escaped_minimal_strings = {
4819   ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4820   ["☒"] = "\\markdownRendererTickedBox{}",
4821   ["◻"] = "\\markdownRendererHalfTickedBox{}",
4822   ["□"] = "\\markdownRendererUntickedBox{}",
4823 }
```

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 for typeset content.

```
4824 self.escaped_chars = {
4825   ["{"] = "\\markdownRendererLeftBrace{}",
4826   ["}"] = "\\markdownRendererRightBrace{}",
4827   ["%"] = "\\markdownRendererPercentSign{}",
4828   ["\\"] = "\\markdownRendererBackslash{}",
4829   ["#"] = "\\markdownRendererHash{}",
4830   ["$"] = "\\markdownRendererDollarSign{}",
4831   ["&"] = "\\markdownRendererAmpersand{}",
4832   ["_"] = "\\markdownRendererUnderscore{}",
4833   ["^"] = "\\markdownRendererCircumflex{}",
4834   ["~"] = "\\markdownRendererTilde{}",
4835   ["|"] = "\\markdownRendererPipe{}",
4836 }
```

Use the `writer->escaped_chars`, `writer->escaped_uri_chars`, and `writer->escaped_minimal_strings` tables to create the `writer->escape`, `writer->escape_uri`, and `writer->escape_minimal` escaper functions.

```
4837 self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
4838 self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
4839 self.escape_minimal = util.escaper({}, self.escaped_minimal_strings)
```

Define `writer->string` as a function that will transform an input plain text span `s` to the output format and `writer->uri` as a function that will transform an input URI `u` to the output format. If the `hybrid` option is enabled, use the `writer->escape_minimal`. Otherwise, use the `writer->escape`, and `writer->escape_uri` functions.

```
4840 if options.hybrid then
4841   self.string = self.escape_minimal
4842   self.uri = self.escape_minimal
4843 else
4844   self.string = self.escape
4845   self.uri = self.escape_uri
4846 end
```

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

```
4847 function self.code(s)
4848   return {"\\markdownRendererCodeSpan{" ,self.escape(s),"}"}
4849 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, and `tit` to the title of the link.

```
4850 function self.link(lab,src,tit)
4851   return {"\\markdownRendererLink{" ,lab,"} ",
4852           "{" ,self.escape(src),"} ",
4853           "{" ,self.uri(src),"} ",
4854           "{" ,self.string(tit or ""),""}
4855 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, and `tit` to the title of the image.

```
4856 function self.image(lab,src,tit)
4857   return {"\\markdownRendererImage{" ,lab,"} ",
4858           "{" ,self.string(src),"} ",
4859           "{" ,self.uri(src),"} ",
4860           "{" ,self.string(tit or ""),""}
4861 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.

```
4862 function self.bulletlist(items,tight)
4863   if not self.is_writing then return "" end
4864   local buffer = {}
4865   for _,item in ipairs(items) do
4866     buffer[#buffer + 1] = self.bulletitem(item)
```



```

4867     end
4868     local contents = util.intersperse(buffer, "\n")
4869     if tight and options.tightLists then
4870         return {"\\markdownRendererUlBeginTight\n", contents,
4871             "\n\\markdownRendererUlEndTight "}
4872     else
4873         return {"\\markdownRendererUlBegin\n", contents,
4874             "\n\\markdownRendererUlEnd "}
4875     end
4876 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.

```

4877 function self.bulletitem(s)
4878     return {"\\markdownRendererUlItem ", s,
4879         "\\markdownRendererUlItemEnd "}
4880 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.

```

4881 function self.orderedlist(items, tight, startnum)
4882     if not self.is_writing then return "" end
4883     local buffer = {}
4884     local num = startnum
4885     for _, item in ipairs(items) do
4886         buffer[#buffer + 1] = self.ordereditem(item, num)
4887         if num ~= nil then
4888             num = num + 1
4889         end
4890     end
4891     local contents = util.intersperse(buffer, "\n")
4892     if tight and options.tightLists then
4893         return {"\\markdownRendererOlBeginTight\n", contents,
4894             "\n\\markdownRendererOlEndTight "}
4895     else
4896         return {"\\markdownRendererOlBegin\n", contents,
4897             "\n\\markdownRendererOlEnd "}
4898     end
4899 end

```

Define `writer->ordereditem` as a function that will transform an input ordered list item to the output format, where `s` is the text of the list item. If the optional parameter `num` is present, it is the number of the list item.

```

4900 function self.ordereditem(s, num)
4901     if num ~= nil then
4902         return {"\\markdownRendererOlItemWithNumber{" , num, "}", s,

```

```

4903             "\\markdownRendererO1ItemEnd "}
4904     else
4905         return {"\\markdownRendererO1Item ",s,
4906             "\\markdownRendererO1ItemEnd "}
4907     end
4908 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.

```

4909 function self.inline_html_comment(contents)
4910     return {"\\markdownRendererInlineHtmlComment{",contents,""}
4911 end

```

Define `writer->block_html_comment` as a function that will transform the contents of a block HTML comment, to the output format, where `contents` are the contents of the HTML comment.

```

4912 function self.block_html_comment(contents)
4913     if not self.is_writing then return "" end
4914     return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
4915         "\n\\markdownRendererBlockHtmlCommentEnd "}
4916 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.

```

4917 function self.inline_html_tag(contents)
4918     return {"\\markdownRendererInlineHtmlTag{",self.string(contents),""}
4919 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.

```

4920 function self.block_html_element(s)
4921     if not self.is_writing then return "" end
4922     local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4923     return {"\\markdownRendererInputBlockHtmlElement{",name,""}
4924 end

```

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

```

4925 function self.emphasis(s)
4926     return {"\\markdownRendererEmphasis{",s,""}
4927 end

```

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

```

4928 function self.checkbox(f)

```

```

4929     if f == 1.0 then
4930         return "☒ "
4931     elseif f == 0.0 then
4932         return "□ "
4933     else
4934         return "☐ "
4935     end
4936 end

```

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

```

4937 function self.strong(s)
4938     return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
4939 end

```

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

```

4940 function self.blockquote(s)
4941     if #util.ropetostring(s) == 0 then return "" end
4942     return {"\\markdownRendererBlockQuoteBegin\n",s,
4943           "\n\\markdownRendererBlockQuoteEnd "}
4944 end

```

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

```

4945 function self.verbatim(s)
4946     if not self.is_writing then return "" end
4947     local name = util.cache_verbatim(options.cacheDir, s)
4948     return {"\\markdownRendererInputVerbatim{" ,name,"}"}
4949 end

```

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

```

4950 function self.document(d)
4951     local active_attributes = self.active_attributes
4952     local buf = {"\\markdownRendererDocumentBegin\n", d}
4953
4954     -- pop attributes for sections that have ended
4955     if options.headerAttributes and self.is_writing then
4956         while #active_attributes > 0 do
4957             local attributes = active_attributes[#active_attributes]
4958             if #attributes > 0 then
4959                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4960             end
4961             table.remove(active_attributes, #active_attributes)
4962         end
4963     end
4964 end

```

```

4965     table.insert(buf, "\\markdownRendererDocumentEnd")
4966
4967     return buf
4968 end

```

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

```

4969 function self.attributes(attr)
4970     local buf = {}
4971
4972     table.sort(attr)
4973     local key, value
4974     for i = 1, #attr do
4975         if attr[i]:sub(1, 1) == "#" then
4976             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
4977                                 attr[i]:sub(2), "}"}))
4978         elseif attr[i]:sub(1, 1) == "." then
4979             table.insert(buf, {"\\markdownRendererAttributeClassName{" ,
4980                                 attr[i]:sub(2), "}"}))
4981         else
4982             key, value = attr[i]:match("(^[^= ]+)%s*=%s*(.*)")
4983             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
4984                                 key, "}{" , value, "}"}))
4985         end
4986     end
4987
4988     return buf
4989 end

```

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

```

4990 self.active_attributes = {}

```

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

```

4991 function self.heading(s, level, attributes)
4992     attributes = attributes or {}
4993     for i = 1, #attributes do
4994         attributes[attributes[i]] = true
4995     end
4996
4997     local active_attributes = self.active_attributes
4998     local slice_begin_type = self.slice_begin:sub(1, 1)
4999     local slice_begin_identifier = self.slice_begin:sub(2) or ""
5000     local slice_end_type = self.slice_end:sub(1, 1)
5001     local slice_end_identifier = self.slice_end:sub(2) or ""
5002
5003     local buf = {}

```

```

5004
5005 -- push empty attributes for implied sections
5006 while #active_attributes < level-1 do
5007   table.insert(active_attributes, {})
5008 end
5009
5010 -- pop attributes for sections that have ended
5011 while #active_attributes >= level do
5012   local active_identifiers = active_attributes[#active_attributes]
5013   -- tear down all active attributes at slice end
5014   if active_identifiers["#" .. slice_end_identifier] ~= nil
5015     and slice_end_type == "$" then
5016     for header_level = #active_attributes, 1, -1 do
5017       if options.headerAttributes and #active_attributes[header_level] > 0 then
5018         table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
5019       end
5020     end
5021     self.is_writing = false
5022   end
5023   table.remove(active_attributes, #active_attributes)
5024   if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
5025     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
5026   end
5027   -- apply all active attributes at slice beginning
5028   if active_identifiers["#" .. slice_begin_identifier] ~= nil
5029     and slice_begin_type == "$" then
5030     for header_level = 1, #active_attributes do
5031       if options.headerAttributes and #active_attributes[header_level] > 0 then
5032         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
5033       end
5034     end
5035     self.is_writing = true
5036   end
5037 end
5038
5039 -- tear down all active attributes at slice end
5040 if attributes["#" .. slice_end_identifier] ~= nil
5041   and slice_end_type == "^" then
5042   for header_level = #active_attributes, 1, -1 do
5043     if options.headerAttributes and #active_attributes[header_level] > 0 then
5044       table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
5045     end
5046   end
5047   self.is_writing = false
5048 end
5049
5050 -- push attributes for the new section

```

```

5051     table.insert(active_attributes, attributes)
5052     if self.is_writing and options.headerAttributes and #attributes > 0 then
5053         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
5054     end
5055
5056     -- apply all active attributes at slice beginning
5057     if attributes["#" .. slice_begin_identifiser] ~= nil
5058         and slice_begin_type == "^" then
5059         for header_level = 1, #active_attributes do
5060             if options.headerAttributes and #active_attributes[header_level] > 0 then
5061                 table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
5062             end
5063         end
5064         self.is_writing = true
5065     end
5066
5067     if self.is_writing then
5068         table.insert(buf, self.attributes(attributes))
5069     end
5070
5071     local cmd
5072     level = level + options.shiftHeadings
5073     if level <= 1 then
5074         cmd = "\\markdownRendererHeadingOne"
5075     elseif level == 2 then
5076         cmd = "\\markdownRendererHeadingTwo"
5077     elseif level == 3 then
5078         cmd = "\\markdownRendererHeadingThree"
5079     elseif level == 4 then
5080         cmd = "\\markdownRendererHeadingFour"
5081     elseif level == 5 then
5082         cmd = "\\markdownRendererHeadingFive"
5083     elseif level >= 6 then
5084         cmd = "\\markdownRendererHeadingSix"
5085     else
5086         cmd = ""
5087     end
5088     if self.is_writing then
5089         table.insert(buf, {cmd, "{", s, "}"})
5090     end
5091
5092     return buf
5093 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.

```

5094     function self.get_state()

```

```

5095     return {
5096         is_writing=self.is_writing,
5097         active_attributes={table.unpack(self.active_attributes)},
5098     }
5099 end

```

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

```

5100 function self.set_state(s)
5101     local previous_state = self.get_state()
5102     for key, value in pairs(s) do
5103         self[key] = value
5104     end
5105     return previous_state
5106 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`.

```

5107 function self.defer_call(f)
5108     local previous_state = self.get_state()
5109     return function(...)
5110         local state = self.set_state(previous_state)
5111         local return_value = f(...)
5112         self.set_state(state)
5113         return return_value
5114     end
5115 end
5116
5117 return self
5118 end

```

3.1.4 Parsers

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

```

5119 local parsers = {}

```

3.1.4.1 Basic Parsers

```

5120 parsers.percent = P("%")
5121 parsers.at = P("@")
5122 parsers.comma = P(",")
5123 parsers.asterisk = P("*")
5124 parsers.dash = P("-")
5125 parsers.plus = P("+")
5126 parsers.underscore = P("_")

```

```

5127 parsers.period           = P(".")
5128 parsers.hash             = P("#")
5129 parsers.ampersand        = P("&")
5130 parsers.backtick         = P("`")
5131 parsers.less              = P("<")
5132 parsers.more              = P(">")
5133 parsers.space             = P(" ")
5134 parsers.squote            = P("'")
5135 parsers.dquote            = P('"')
5136 parsers.lparent           = P("(")
5137 parsers.rparent           = P(")")
5138 parsers.lbracket          = P("[")
5139 parsers.rbracket          = P("]")
5140 parsers.lbrace             = P("{")
5141 parsers.rbrace             = P("}")
5142 parsers.circumflex        = P("^")
5143 parsers.slash             = P("/")
5144 parsers.equal              = P("=")
5145 parsers.colon              = P(":")
5146 parsers.semicolon         = P(";")
5147 parsers.exclamation       = P("!")
5148 parsers.pipe              = P("|")
5149 parsers.tilde              = P("~")
5150 parsers.backslash         = P("\\")
5151 parsers.tab                 = P("\t")
5152 parsers.newline            = P("\n")
5153 parsers.tightblocksep     = P("\001")
5154
5155 parsers.digit              = R("09")
5156 parsers.hexdigit           = R("09", "af", "AF")
5157 parsers.letter             = R("AZ", "az")
5158 parsers.alphanumeric      = R("AZ", "az", "09")
5159 parsers.keyword           = parsers.letter
5160                          * parsers.alphanumeric^0
5161 parsers.internal_punctuation = S(";,?.?")
5162
5163 parsers.doubleasterisks    = P("**")
5164 parsers.doubleunderscores  = P("__")
5165 parsers.doubletildes       = P("~~")
5166 parsers.fourspace         = P("    ")
5167
5168 parsers.any                = P(1)
5169 parsers.succeed            = P(true)
5170 parsers.fail                = P(false)
5171
5172 parsers.escapable          = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
5173 parsers.anyescaped         = parsers.backslash / " " * parsers.escapable

```



```

5174                                     + parsers.any
5175
5176 parsers.spacechar                    = S("\t ")
5177 parsers.spacing                      = S(" \n\r\t")
5178 parsers.nospacechar                  = parsers.any - parsers.spacing
5179 parsers.optionalspace                 = parsers.spacechar^0
5180
5181 parsers.normalchar                    = parsers.any - (V("SpecialChar")
5182                                     + parsers.spacing
5183                                     + parsers.tightblocksep)
5184 parsers.eof                           = -parsers.any
5185 parsers.nonindentpace                 = parsers.space^-3 * - parsers.spacechar
5186 parsers.indent                        = parsers.space^-3 * parsers.tab
5187                                     + parsers.fourspace / ""
5188 parsers.linechar                      = P(1 - parsers.newline)
5189
5190 parsers.blankline                     = parsers.optionalspace
5191                                     * parsers.newline / "\n"
5192 parsers.blanklines                    = parsers.blankline^0
5193 parsers.skipblanklines                 = (parsers.optionalspace * parsers.newline)^0
5194 parsers.indentedline                  = parsers.indent / ""
5195                                     * C(parsers.linechar^1 * parsers.newline^-
5196                                     1)
5196 parsers.optionallyindentedline        = parsers.indent^-1 / ""
5197                                     * C(parsers.linechar^1 * parsers.newline^-
5198                                     1)
5198 parsers.sp                             = parsers.spacing^0
5199 parsers.spnl                           = parsers.optionalspace
5200                                     * (parsers.newline * parsers.optionalspace)^-
5201                                     1
5201 parsers.line                           = parsers.linechar^0 * parsers.newline
5202 parsers.nonemptyline                   = parsers.line - parsers.blankline

```

The `parsers.commented_line^1` parser recognizes the regular language of T_EX comments, see an equivalent finite automaton in Figure 6.

```

5203 parsers.commented_line_letter          = parsers.linechar
5204                                     + parsers.newline
5205                                     - parsers.backslash
5206                                     - parsers.percent
5207 parsers.commented_line                 = Cg(Cc(""), "backslashes")
5208                                     * ((#(parsers.commented_line_letter
5209                                     - parsers.newline)
5210                                     * Cb("backslashes")
5211                                     * Cs(parsers.commented_line_letter
5212                                     - parsers.newline)^1 -- initial
5213                                     * Cg(Cc(""), "backslashes"))
5214                                     + #(parsers.backslash * parsers.backslash)

```

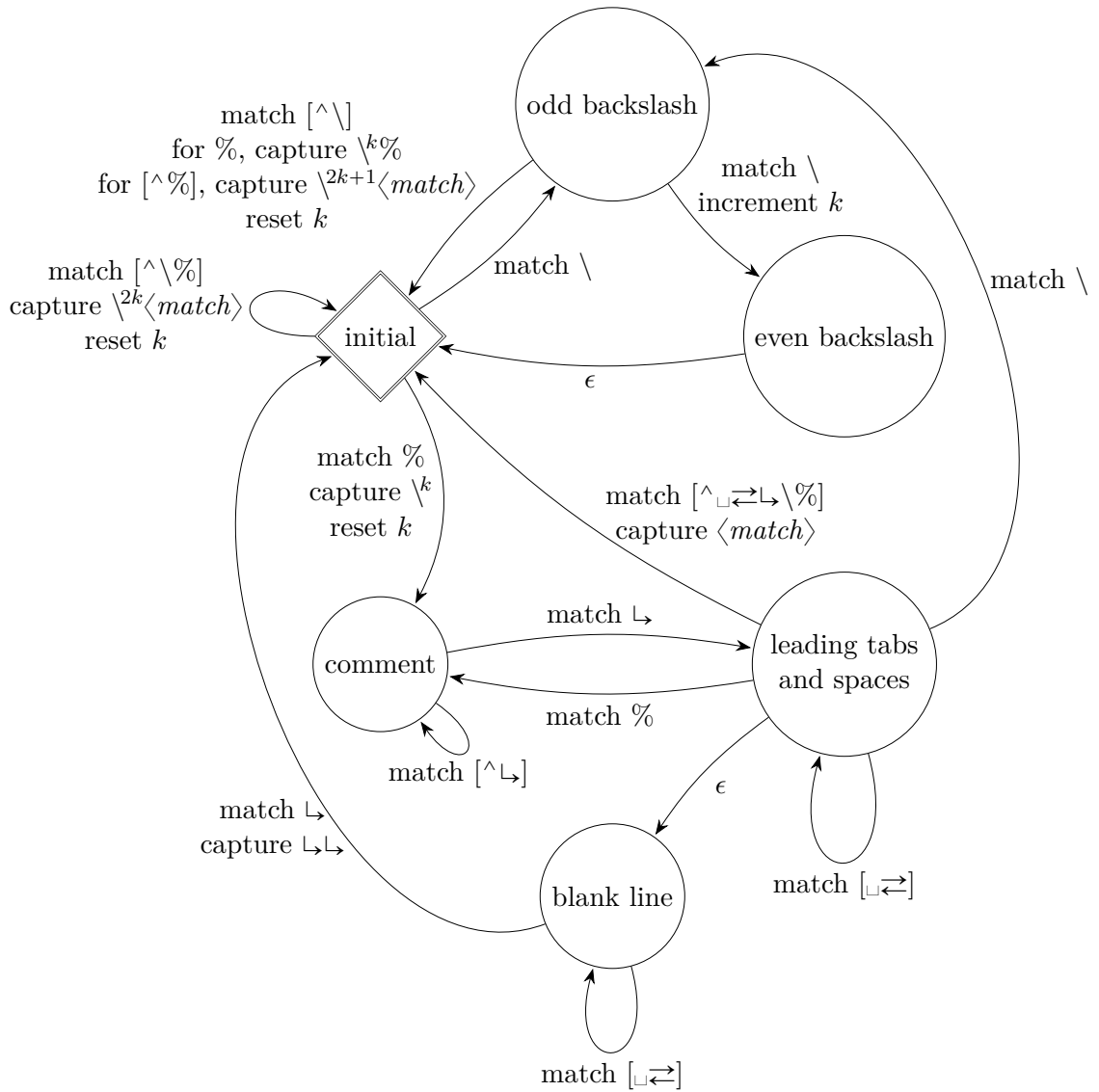


Figure 6: A pushdown automaton that recognizes TeX comments

```

5215 * Cg((parsers.backslash -- even backslash
5216 * parsers.backslash)^1, "backslashes")
5217 + (parsers.backslash
5218 * (#parsers.percent
5219 * Cb("backslashes")
5220 / function(backslashes)
5221 return string.rep("\\", #backslashes / 2)
5222 end
5223 * C(parsers.percent)
5224 + #parsers.commented_line_letter
5225 * Cb("backslashes")
5226 * Cc("\\")
5227 * C(parsers.commented_line_letter))
5228 * Cg(Cc(""), "backslashes"))^0
5229 * (#parsers.percent
5230 * Cb("backslashes")
5231 / function(backslashes)
5232 return string.rep("\\", #backslashes / 2)
5233 end
5234 * ((parsers.percent -- comment
5235 * parsers.line
5236 * #parsers.blankline) -- blank line
5237 / "\n"
5238 + parsers.percent -- comment
5239 * parsers.line
5240 * parsers.optionalspace) -- leading tabs and space
5241 + #parsers.newline)
5242 * Cb("backslashes")
5243 * C(parsers.newline))
5244
5245 parsers.chunk = parsers.line * (parsers.optionallyindentedline
5246 - parsers.blankline)^0
5247
5248 parsers.attribute_key_char = parsers.alphanumeric + S("_-")
5249 parsers.attribute_key = (parsers.attribute_key_char
5250 - parsers.dash - parsers.digit)
5251 * parsers.attribute_key_char^0
5252 parsers.attribute_value = ( (parsers.dquote / "")
5253 * (parsers.anyescaped - parsers.dquote)^0
5254 * (parsers.dquote / ""))
5255 + ( parsers.anyescaped - parsers.dquote - parsers.rbrack
5256 - parsers.space)^0
5257
5258 parsers.attribute = (parsers.dash * Cc(".unnumbered"))
5259 + C((parsers.hash + parsers.period)
5260 * parsers.attribute_key)
5261 + Cs( parsers.attribute_key

```

```

5262             * parsers.optionalspace * parsers.equal * parsers.optionalspace
5263             * parsers.attribute_value)
5264 parsers.attributes = parsers.lbrace
5265             * parsers.optionalspace
5266             * parsers.attribute
5267             * (parsers.spacechar^1
5268             * parsers.attribute)^0
5269             * parsers.optionalspace
5270             * parsers.rbrace
5271
5272 -- block followed by 0 or more optionally
5273 -- indented blocks with first line indented.
5274 parsers.indented_blocks = function(bl)
5275   return Cs( bl
5276             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
5277             * (parsers.blankline^1 + parsers.eof) )
5278 end

```

3.1.4.2 Parsers Used for Markdown Lists

```

5279 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
5280
5281 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
5282                  * (parsers.tab + parsers.space^-
5283                    3)
5284                  + parsers.space * parsers.bulletchar * #parsers.spacing
5285                  * (parsers.tab + parsers.space^-2)
5286                  + parsers.space * parsers.space * parsers.bulletchar
5287                  * #parsers.spacing
5288                  * (parsers.tab + parsers.space^-1)
5289                  + parsers.space * parsers.space * parsers.space
5290                  * parsers.bulletchar * #parsers.spacing
5291                  )
5292 local function tickbox(interior)
5293   return parsers.optionalspace * parsers.lbracket
5294         * interior * parsers.rbracket * parsers.spacechar^1
5295 end
5296
5297 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
5298 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
5299 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
5300

```

3.1.4.3 Parsers Used for Markdown Code Spans

```

5301 parsers.openticks = Cg(parsers.backtick^1, "ticks")
5302

```

```

5303 local function captures_equal_length(_,i,a,b)
5304   return #a == #b and i
5305 end
5306
5307 parsers.closeticks = parsers.space^-1
5308                   * Cmt(C(parsers.backtick^1)
5309                       * Cb("ticks"), captures_equal_length)
5310
5311 parsers.intickschar = (parsers.any - S(" \n\r`"))
5312                   + (parsers.newline * -parsers.blankline)
5313                   + (parsers.space - parsers.closeticks)
5314                   + (parsers.backtick^1 - parsers.closeticks)
5315
5316 parsers.inticks    = parsers.openticks * parsers.space^-1
5317                   * C(parsers.intickschar^0) * parsers.closeticks

```

3.1.4.4 Parsers Used for Fenced Code Blocks

```

5318 local function captures_geq_length(_,i,a,b)
5319   return #a >= #b and i
5320 end
5321
5322 parsers.tilde_infostring
5323                   = C((parsers.linechar
5324                       - (parsers.spacechar^1 * parsers.newline))^0)
5325                   * parsers.optionalspace
5326                   * (parsers.newline + parsers.eof)
5327
5328 parsers.backtick_infostring
5329                   = C((parsers.linechar
5330                       - (parsers.backtick
5331                           + parsers.spacechar^1 * parsers.newline))^0)
5332                   * parsers.optionalspace
5333                   * (parsers.newline + parsers.eof)
5334
5335 local fenceindent
5336 parsers.fencehead = function(char, infostring)
5337   return          C(parsers.nonindentpace) / function(s) fenceindent = #s end
5338                   * Cg(char^3, "fencelength")
5339                   * parsers.optionalspace * infostring
5340 end
5341
5342 parsers.fencehead_with_attributes
5343                   = function(char)
5344   return          C(parsers.nonindentpace) / function(s) fenceindent = #s end
5345                   * Cg(char^3, "fencelength")
5346                   * parsers.optionalspace * Ct(parsers.attributes)

```

```

5347             * parsers.optionalspace * (parsers.newline + parsers.eof)
5348 end
5349
5350 parsers.fencetail = function(char)
5351   return
5352     parsers.nonindentospace
5353     * Cmt(C(char~3) * Cb("fencelength"), captures_geq_length)
5354     * parsers.optionalspace * (parsers.newline + parsers.eof)
5355     + parsers.eof
5356 end
5357
5358 parsers.fencedline = function(char)
5359   return
5360     C(parsers.line - parsers.fencetail(char))
5361 / function(s)
5362   local i = 1
5363   local remaining = fenceindent
5364   while true do
5365     local c = s:sub(i, i)
5366     if c == " " and remaining > 0 then
5367       remaining = remaining - 1
5368       i = i + 1
5369     elseif c == "\t" and remaining > 3 then
5370       remaining = remaining - 4
5371       i = i + 1
5372     else
5373       break
5374     end
5375   end
5376   return s:sub(i)
5377 end
5378 end

```

3.1.4.5 Parsers Used for Markdown Tags and Links

```

5377 parsers.leader    = parsers.space^-3
5378
5379 -- content in balanced brackets, parentheses, or quotes:
5380 parsers.bracketed = P{ parsers.lbracket
5381   * (( parsers.backslash / "\"" * parsers.rbracket
5382     + parsers.any - (parsers.lbracket
5383       + parsers.rbracket
5384       + parsers.blankline^2)
5385     ) + V(1))^0
5386   * parsers.rbracket }
5387
5388 parsers.inparens  = P{ parsers.lparent
5389   * ((parsers.anyescaped - (parsers.lparent
5390     + parsers.rparent

```

```

5391                                     + parsers.blankline^2)
5392         ) + V(1))^0
5393         * parsers.rparent }
5394
5395 parsers.squoted = P{ parsers.squote * parsers.alphanumeric
5396         * ((parsers.anyescaped - (parsers.squote
5397         + parsers.blankline^2)
5398         ) + V(1))^0
5399         * parsers.squote }
5400
5401 parsers.dquoted = P{ parsers.dquote * parsers.alphanumeric
5402         * ((parsers.anyescaped - (parsers.dquote
5403         + parsers.blankline^2)
5404         ) + V(1))^0
5405         * parsers.dquote }
5406
5407 -- bracketed tag for markdown links, allowing nested brackets:
5408 parsers.tag = parsers.lbracket
5409         * Cs((parsers.alphanumeric^1
5410         + parsers.bracketed
5411         + parsers.inticks
5412         + ( parsers.backslash / "\"" * parsers.rbracket
5413         + parsers.any
5414         - (parsers.rbracket + parsers.blankline^2)))^0)
5415         * parsers.rbracket
5416
5417 -- url for markdown links, allowing nested brackets:
5418 parsers.url = parsers.less * Cs((parsers.anyescaped
5419         - parsers.more)^0)
5420         * parsers.more
5421         + Cs((parsers.inparens + (parsers.anyescaped
5422         - parsers.spacing
5423         - parsers.rparent))^1)
5424
5425 -- quoted text, possibly with nested quotes:
5426 parsers.title_s = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5427         + parsers.squoted)^0)
5428         * parsers.squote
5429
5430 parsers.title_d = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5431         + parsers.dquoted)^0)
5432         * parsers.dquote
5433
5434 parsers.title_p = parsers.lparent
5435         * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5436         * parsers.rparent
5437

```

```

5438 parsers.title      = parsers.title_d + parsers.title_s + parsers.title_p
5439
5440 parsers.optionaltitle
5441                   = parsers.spnl * parsers.title * parsers.spacechar^0
5442                   + Cc("")

```

3.1.4.6 Parsers Used for HTML

```

5443 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5444 parsers.keyword_exact = function(s)
5445   local parser = P(0)
5446   for i=1,#s do
5447     local c = s:sub(i,i)
5448     local m = c .. upper(c)
5449     parser = parser * S(m)
5450   end
5451   return parser
5452 end
5453
5454 parsers.block_keyword =
5455   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5456   parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5457   parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5458   parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5459   parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5460   parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5461   parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5462   parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5463   parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5464   parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5465   parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5466   parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5467   parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5468   parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5469   parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5470   parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5471   parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5472   parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5473
5474 -- There is no reason to support bad html, so we expect quoted attributes
5475 parsers.htmlattributevalue
5476                   = parsers.squote * (parsers.any - (parsers.blankline
5477                   + parsers.squote))^0
5478                   * parsers.squote
5479                   + parsers.dquote * (parsers.any - (parsers.blankline
5480                   + parsers.dquote))^0
5481                   * parsers.dquote

```



```

5482
5483 parsers.htmlattribute      = parsers.spacing^1
5484                             * (parsers.alphanumeric + S("_-"))^1
5485                             * parsers.sp * parsers.equal * parsers.sp
5486                             * parsers.htmlattributevalue
5487
5488 parsers.htmlcomment         = P("<!--")
5489                             * parsers.optionalspace
5490                             * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5491                             * parsers.optionalspace
5492                             * P("-->")
5493
5494 parsers.htmlinstruction      = P("<?") * (parsers.any - P("?>"))^0 * P("?>")
5495
5496 parsers.openelt_any         = parsers.less * parsers.keyword * parsers.htmlattribute^0
5497                             * parsers.sp * parsers.more
5498
5499 parsers.openelt_exact       = function(s)
5500     return parsers.less * parsers.sp * parsers.keyword_exact(s)
5501         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5502 end
5503
5504 parsers.openelt_block       = parsers.sp * parsers.block_keyword
5505                             * parsers.htmlattribute^0 * parsers.sp * parsers.more
5506
5507 parsers.closeelt_any        = parsers.less * parsers.sp * parsers.slash
5508                             * parsers.keyword * parsers.sp * parsers.more
5509
5510 parsers.closeelt_exact      = function(s)
5511     return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5512         * parsers.sp * parsers.more
5513 end
5514
5515 parsers.emptyelt_any        = parsers.less * parsers.sp * parsers.keyword
5516                             * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5517                             * parsers.more
5518
5519 parsers.emptyelt_block      = parsers.less * parsers.sp * parsers.block_keyword
5520                             * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5521                             * parsers.more
5522
5523 parsers.displaytext         = (parsers.any - parsers.less)^1
5524
5525 -- return content between two matched HTML tags
5526 parsers.in_matched          = function(s)
5527     return { parsers.openelt_exact(s)
5528         * (V(1) + parsers.displaytext

```

```

5529         + (parsers.less - parsers.closeelt_exact(s))^0
5530     * parsers.closeelt_exact(s) }
5531 end
5532
5533 local function parse_matched_tags(s,pos)
5534     local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5535     return lpeg.match(parsers.in_matched(t),s,pos-1)
5536 end
5537
5538 parsers.in_matched_block_tags = parsers.less
5539                               * Cmt(#parsers.openelt_block, parse_matched_tags)
5540

```

3.1.4.7 Parsers Used for HTML Entities

```

5541 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5542                   * C(parsers.hexdigit^1) * parsers.semicolon
5543 parsers.decentity = parsers.ampersand * parsers.hash
5544                   * C(parsers.digit^1) * parsers.semicolon
5545 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5546                   * parsers.semicolon

```

3.1.4.8 Helpers for References

```

5547 -- parse a reference definition: [foo]: /bar "title"
5548 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon
5549                                 * parsers.spacechar^0 * parsers.url
5550                                 * parsers.optionaltitle * parsers.blankline^1

```

3.1.4.9 Inline Elements

```

5551 parsers.Inline      = V("Inline")
5552 parsers.IndentedInline = V("IndentedInline")
5553
5554 -- parse many p between starter and ender
5555 parsers.between = function(p, starter, ender)
5556     local ender2 = B(parsers.nonspacechar) * ender
5557     return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5558 end
5559
5560 parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more

```

3.1.4.10 Block Elements

```

5561 parsers.TildeFencedCode
5562     = parsers.fencehead(parsers.tilde,
5563                         parsers.tilde_infostring)
5564     * Cs(parsers.fencedline(parsers.tilde)^0)
5565     * parsers.fencetail(parsers.tilde)

```

```

5566
5567 parsers.BacktickFencedCode
5568     = parsers.fencehead(parsers.backtick,
5569                         parsers.backtick_infostring)
5570     * Cs(parsers.fencedline(parsers.backtick)^0)
5571     * parsers.fencetail(parsers.backtick)
5572
5573 parsers.lineof = function(c)
5574     return (parsers.leader * (P(c) * parsers.optionalspace)^3
5575           * (parsers.newline * parsers.blankline^1
5576             + parsers.newline^-1 * parsers.eof))
5577 end

```

3.1.4.11 Headings

```

5578 -- parse Atx heading start and return level
5579 parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
5580                       * -parsers.hash / length
5581
5582 -- parse setext header ending and return level
5583 parsers.heading_level = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5584
5585 local function strip_atx_end(s)
5586     return s:gsub("#%s*\n$", "")
5587 end

```

3.1.5 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>`.

```

5588 M.reader = {}
5589 function M.reader.new(writer, options)
5590     local self = {}

```

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

```

5591     self.writer = writer
5592     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.

```
5593 self.parsers = {}
5594 (function(parsers)
5595   setmetatable(self.parsers, {
5596     __index = function (_, key)
5597       return parsers[key]
5598     end
5599   })
5600 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.

```
5601 local parsers = self.parsers
```

3.1.5.1 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.

```
5602 function self.normalize_tag(tag)
5603   return string.lower(
5604     gsub(util.ropo_to_string(tag), "[ \\n\\r\\t]+", " "))
5605 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.

```
5606 local function iterlines(s, f)
5607   local rope = lpeg.match(Ct((parsers.line / f)^1), s)
5608   return util.ropo_to_string(rope)
5609 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.

```
5610 if options.preserveTabs then
5611   self.expandtabs = function(s) return s end
5612 else
5613   self.expandtabs = function(s)
5614     if s:find("\\t") then
5615       return iterlines(s, util.expand_tabs_in_line)
5616     else
5617       return s
5618     end
5619   end
5620 end
```

3.1.5.2 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.

```
5621 self.parser_functions = {}
5622 self.create_parser = function(name, grammar, toplevel)
5623     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.

```
5624     if toplevel and options.stripIndent then
5625         local min_prefix_length, min_prefix = nil, ''
5626         str = iterlines(str, function(line)
5627             if lpeg.match(parsers.nonemptyline, line) == nil then
5628                 return line
5629             end
5630             line = util.expand_tabs_in_line(line)
5631             local prefix = lpeg.match(C(parsers.optionalspace), line)
5632             local prefix_length = #prefix
5633             local is_shorter = min_prefix_length == nil
5634             is_shorter = is_shorter or prefix_length < min_prefix_length
5635             if is_shorter then
5636                 min_prefix_length, min_prefix = prefix_length, prefix
5637             end
5638             return line
5639         end)
5640         str = str:gsub('^' .. min_prefix, '')
5641     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.

```
5642     if toplevel and (options.texComments or options.hybrid) then
5643         str = lpeg.match(Ct(parsers.commented_line^1), str)
5644         str = util.rope_to_string(str)
5645     end
5646     local res = lpeg.match(grammar(), str)
5647     if res == nil then
5648         error(format("%s failed on:\n%s", name, str:sub(1,20)))
5649     else
5650         return res
5651     end
```

```

5652     end
5653 end
5654
5655 self.create_parser("parse_blocks",
5656                   function()
5657                     return parsers.blocks
5658                   end, true)
5659
5660 self.create_parser("parse_blocks_nested",
5661                   function()
5662                     return parsers.blocks_nested
5663                   end, false)
5664
5665 self.create_parser("parse_inlines",
5666                   function()
5667                     return parsers.inlines
5668                   end, false)
5669
5670 self.create_parser("parse_inlines_no_link",
5671                   function()
5672                     return parsers.inlines_no_link
5673                   end, false)
5674
5675 self.create_parser("parse_inlines_no_inline_note",
5676                   function()
5677                     return parsers.inlines_no_inline_note
5678                   end, false)
5679
5680 self.create_parser("parse_inlines_no_html",
5681                   function()
5682                     return parsers.inlines_no_html
5683                   end, false)
5684
5685 self.create_parser("parse_inlines_nbsp",
5686                   function()
5687                     return parsers.inlines_nbsp
5688                   end, false)

```

3.1.5.3 Parsers Used for Markdown Lists (local)

```

5689 if options.hashEnumerators then
5690   parsers.dig = parsers.digit + parsers.hash
5691 else
5692   parsers.dig = parsers.digit
5693 end
5694
5695 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing

```

```

5696         + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5697           * (parsers.tab + parsers.space^1)
5698         + C(parsers.dig * parsers.period) * #parsers.spacing
5699           * (parsers.tab + parsers.space^-2)
5700         + parsers.space * C(parsers.dig^2 * parsers.period)
5701           * #parsers.spacing
5702         + parsers.space * C(parsers.dig * parsers.period)
5703           * #parsers.spacing
5704           * (parsers.tab + parsers.space^-1)
5705         + parsers.space * parsers.space * C(parsers.dig^1
5706           * parsers.period) * #parsers.spacing

```

3.1.5.4 Parsers Used for Blockquotes (local)

```

5707 -- strip off leading > and indents, and run through blocks
5708 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
5709 1)/""
5710
5711           * parsers.linechar^0 * parsers.newline)^1
5712           * (-(parsers.leader * parsers.more
5713             + parsers.blankline) * parsers.linechar^1
5714             * parsers.newline)^0
5715
5716 if not options.breakableBlockquotes then
5717   parsers.blockquote_body = parsers.blockquote_body
5718   * (parsers.blankline^0 / "")
5719 end

```

3.1.5.5 Helpers for Links and References (local)

```

5718 -- List of references defined in the document
5719 local references
5720
5721 -- add a reference to the list
5722 local function register_link(tag,url,title)
5723   references[self.normalize_tag(tag)] = { url = url, title = title }
5724   return ""
5725 end
5726
5727 -- lookup link reference and return either
5728 -- the link or nil and fallback text.
5729 local function lookup_reference(label,sps,tag)
5730   local tagpart
5731   if not tag then
5732     tag = label
5733     tagpart = ""
5734   elseif tag == "" then
5735     tag = label
5736     tagpart = "[]"

```

```

5737     else
5738         tagpart = {"[",
5739                 self.parser_functions.parse_inlines(tag),
5740                 "]"},
5741     end
5742     if sps then
5743         tagpart = {sps, tagpart}
5744     end
5745     local r = references[self.normalize_tag(tag)]
5746     if r then
5747         return r
5748     else
5749         return nil, {"[",
5750                 self.parser_functions.parse_inlines(label),
5751                 "]", tagpart}
5752     end
5753 end
5754
5755 -- lookup link reference and return a link, if the reference is found,
5756 -- or a bracketed label otherwise.
5757 local function indirect_link(label,sps,tag)
5758     return writer.defer_call(function()
5759         local r, fallback = lookup_reference(label,sps,tag)
5760         if r then
5761             return writer.link(
5762                 self.parser_functions.parse_inlines_no_link(label),
5763                 r.url, r.title)
5764         else
5765             return fallback
5766         end
5767     end)
5768 end
5769
5770 -- lookup image reference and return an image, if the reference is found,
5771 -- or a bracketed label otherwise.
5772 local function indirect_image(label,sps,tag)
5773     return writer.defer_call(function()
5774         local r, fallback = lookup_reference(label,sps,tag)
5775         if r then
5776             return writer.image(writer.string(label), r.url, r.title)
5777         else
5778             return {"!", fallback}
5779         end
5780     end)
5781 end

```


3.1.5.6 Inline Elements (local)

```
5782 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5783                  / writer.string
5784
5785 parsers.Symbol    = (V("SpecialChar") - parsers.tightblocksep)
5786                  / writer.string
5787
5788 parsers.Ellipsis  = P("...") / writer.ellipsis
5789
5790 parsers.Smart     = parsers.Ellipsis
5791
5792 parsers.Code      = parsers.inticks / writer.code
5793
5794 if options.blankBeforeBlockquote then
5795   parsers.bqstart = parsers.fail
5796 else
5797   parsers.bqstart = parsers.more
5798 end
5799
5800 if options.blankBeforeHeading then
5801   parsers.headerstart = parsers.fail
5802 else
5803   parsers.headerstart = parsers.hash
5804                       + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5805                          * parsers.optionalspace * parsers.newline)
5806 end
5807
5808 parsers.EndlineExceptions
5809         = parsers.blankline -- paragraph break
5810         + parsers.tightblocksep -- nested list
5811         + parsers.eof        -- end of document
5812         + parsers.bqstart
5813         + parsers.headerstart
5814
5815 parsers.Endline  = parsers.newline
5816                 * -V("EndlineExceptions")
5817                 * parsers.spacechar^0
5818                 / (options.hardLineBreaks and writer.linebreak
5819                    or writer.space)
5820
5821 parsers.OptionalIndent
5822         = parsers.spacechar^1 / writer.space
5823
5824 parsers.Space    = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5825                 + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5826                 + parsers.spacechar^1 * parsers.Endline
5827                 * parsers.optionalspace
```

```

5828                                     / (options.hardLineBreaks
5829                                     and writer.linebreak
5830                                     or writer.space)
5831         + parsers.spacechar1 * parsers.optionalspace
5832                                     / writer.space
5833
5834 parsers.NonbreakingEndline
5835     = parsers.newline
5836     * -V("EndlineExceptions")
5837     * parsers.spacechar0
5838     / (options.hardLineBreaks and writer.linebreak
5839        or writer.nbsp)
5840
5841 parsers.NonbreakingSpace
5842     = parsers.spacechar2 * parsers.Endline / writer.linebreak
5843     + parsers.spacechar1 * parsers.Endline-1 * parsers.eof / ""
5844     + parsers.spacechar1 * parsers.Endline
5845       * parsers.optionalspace
5846       / (options.hardLineBreaks
5847          and writer.linebreak
5848          or writer.nbsp)
5849     + parsers.spacechar1 * parsers.optionalspace
5850       / writer.nbsp
5851
5852 if options.underscores then
5853     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5854                                     parsers.doubleasterisks)
5855                       + parsers.between(parsers.Inline, parsers.doubleunderscores,
5856                                     parsers.doubleunderscores)
5857                       ) / writer.strong
5858
5859     parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5860                                     parsers.asterisk)
5861                       + parsers.between(parsers.Inline, parsers.underscore,
5862                                     parsers.underscore)
5863                       ) / writer.emphasis
5864 else
5865     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5866                                     parsers.doubleasterisks)
5867                       ) / writer.strong
5868
5869     parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5870                                     parsers.asterisk)
5871                       ) / writer.emphasis
5872 end
5873
5874 parsers.AutoLinkUrl = parsers.less

```

```

5875         * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5876         * parsers.more
5877         / function(url)
5878             return writer.link(writer.escape(url), url)
5879         end
5880
5881 parsers.AutoLinkEmail = parsers.less
5882         * C((parsers.alphanumeric + S("-._+"))^1
5883         * P("@") * parsers.urlchar^1)
5884         * parsers.more
5885         / function(email)
5886             return writer.link(writer.escape(email),
5887                               "mailto: "..email)
5888         end
5889
5890 parsers.AutoLinkRelativeReference
5891         = parsers.less
5892         * C(parsers.urlchar^1)
5893         * parsers.more
5894         / function(url)
5895             return writer.link(writer.escape(url), url)
5896         end
5897
5898 parsers.DirectLink = (parsers.tag / self.parser_functions.parse_inlines_no_link)
5899         * parsers.spnl
5900         * parsers.lparent
5901         * (parsers.url + Cc("")) -- link can be empty [foo]()
5902         * parsers.optionaltitle
5903         * parsers.rparent
5904         / writer.link
5905
5906 parsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
5907         / indirect_link
5908
5909 -- parse a link or image (direct or indirect)
5910 parsers.Link = parsers.DirectLink + parsers.IndirectLink
5911
5912 parsers.DirectImage = parsers.exclamation
5913         * (parsers.tag / self.parser_functions.parse_inlines)
5914         * parsers.spnl
5915         * parsers.lparent
5916         * (parsers.url + Cc("")) -- link can be empty [foo]()
5917         * parsers.optionaltitle
5918         * parsers.rparent
5919         / writer.image
5920

```

```

5921 parsers.IndirectImage = parsers.exclamation * parsers.tag
5922                       * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5923
5924 parsers.Image          = parsers.DirectImage + parsers.IndirectImage
5925
5926 -- avoid parsing long strings of * or _ as emph/strong
5927 parsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
5928                       / writer.string
5929
5930 parsers.EscapedChar   = parsers.backslash * C(parsers.escapable) / writer.string
5931
5932 parsers.InlineHtml    = parsers.emptyelt_any / writer.inline_html_tag
5933                       + (parsers.htmlcomment / self.parser_functions.parse_inlines_r
5934                          / writer.inline_html_comment
5935                          + parsers.htmlinstruction
5936                          + parsers.openelt_any / writer.inline_html_tag
5937                          + parsers.closeelt_any / writer.inline_html_tag
5938
5939 parsers.HtmlEntity    = parsers.hexentity / entities.hex_entity / writer.string
5940                       + parsers.decentity / entities.dec_entity / writer.string
5941                       + parsers.tagentity / entities.char_entity / writer.string

```

3.1.5.7 Block Elements (local)

```

5942 parsers.DisplayHtml  = (parsers.htmlcomment / self.parser_functions.parse_blocks_nes
5943                       / writer.block_html_comment
5944                       + parsers.emptyelt_block / writer.block_html_element
5945                       + parsers.openelt_exact("hr") / writer.block_html_element
5946                       + parsers.in_matched_block_tags / writer.block_html_element
5947                       + parsers.htmlinstruction
5948
5949 parsers.Verbatim     = Cs( (parsers.blanklines
5950                           * ((parsers.indentedline - parsers.blankline))^1)^1
5951                           ) / self.expandtabs / writer.verbatim
5952
5953 parsers.Blockquote   = Cs(parsers.blockquote_body^1)
5954                       / self.parser_functions.parse_blocks_nested
5955                       / writer.blockquote
5956
5957 parsers.ThematicBreak = ( parsers.lineof(parsers.asterisk)
5958                           + parsers.lineof(parsers.dash)
5959                           + parsers.lineof(parsers.underscore)
5960                           ) / writer.thematic_break
5961
5962 parsers.Reference    = parsers.define_reference_parser / register_link
5963
5964 parsers.Paragraph    = parsers.nonindentspace * Ct(parsers.Inline^1)

```

```

5965         * ( parsers.newline
5966         * ( parsers.blankline^1
5967         + #parsers.hash
5968         + #(parsers.leader * parsers.more * parsers.space^-
1)
5969         + parsers.eof
5970         )
5971         + parsers.eof )
5972         / writer.paragraph
5973
5974 parsers.Plain      = parsers.nonindentspace * Ct(parsers.Inline^1)
5975                   / writer.plain

```

3.1.5.8 Lists (local)

```

5976 parsers.starter = parsers.bullet + parsers.enumerator
5977
5978 if options.taskLists then
5979     parsers.tickbox = ( parsers.ticked_box
5980                       + parsers.halfticked_box
5981                       + parsers.unticked_box
5982                       ) / writer.tickbox
5983 else
5984     parsers.tickbox = parsers.fail
5985 end
5986
5987 -- we use \001 as a separator between a tight list item and a
5988 -- nested list under it.
5989 parsers.NestedList      = Cs((parsers.optionallyindentedline
5990                             - parsers.starter)^1)
5991                         / function(a) return "\001"..a end
5992
5993 parsers.ListBlockLine   = parsers.optionallyindentedline
5994                         - parsers.blankline - (parsers.indent^-
1
5995                                     * parsers.starter)
5996
5997 parsers.ListBlock       = parsers.line * parsers.ListBlockLine^0
5998
5999 parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6000                             * parsers.ListBlock
6001
6002 parsers.TightListItem = function(starter)
6003     return -parsers.ThematicBreak
6004           * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Nes
1)
6005           / self.parser_functions.parse_blocks_nested)

```

```

6006             * -(parsers.blanklines * parsers.indent)
6007     end
6008
6009     parsers.LooseListItem = function(starter)
6010         return -parsers.ThematicBreak
6011             * Cs( starter / "" * parsers.checkbox^-1 * parsers.ListBlock * Cc("\n")
6012             * (parsers.NestedList + parsers.ListContinuationBlock^0)
6013             * (parsers.blanklines / "\n\n")
6014             ) / self.parser_functions.parse_blocks_nested
6015     end
6016
6017     parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
6018         * parsers.skipblanklines * -parsers.bullet
6019         + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
6020         * parsers.skipblanklines )
6021         / writer.bulletlist
6022
6023     local function ordered_list(items,tight,startnum)
6024         if options.startNumber then
6025             startnum = tonumber(startnum) or 1 -- fallback for '#'
6026             if startnum ~= nil then
6027                 startnum = math.floor(startnum)
6028             end
6029         else
6030             startnum = nil
6031         end
6032         return writer.orderedlist(items,tight,startnum)
6033     end
6034
6035     parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
6036         ( Ct(parsers.TightListItem(Cb("listtype")))
6037             * parsers.TightListItem(parsers.enumerator)^0)
6038         * Cc(true) * parsers.skipblanklines * -parsers.enumerator
6039         + Ct(parsers.LooseListItem(Cb("listtype")))
6040             * parsers.LooseListItem(parsers.enumerator)^0)
6041         * Cc(false) * parsers.skipblanklines
6042         ) * Cb("listtype") / ordered_list

```

3.1.5.9 Blank (local)

```

6043     parsers.Blank           = parsers.blankline / ""
6044                             + parsers.Reference
6045                             + (parsers.tightblocksep / "\n")

```

3.1.5.10 Headings (local)

```

6046     -- parse atx header
6047     parsers.AtxHeading = Cg(parsers.heading_start, "level")

```

```

6048         * parsers.optionalspace
6049         * (C(parsers.line)
6050           / strip_atx_end
6051           / self.parser_functions.parse_inlines)
6052         * Cb("level")
6053         / writer.heading
6054
6055     parsers.SetextHeading = #(parsers.line * S("=-"))
6056         * Ct(parsers.linechar^1
6057           / self.parser_functions.parse_inlines)
6058         * parsers.newline
6059         * parsers.heading_level
6060         * parsers.optionalspace
6061         * parsers.newline
6062         / writer.heading
6063
6064     parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

3.1.5.11 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.

```

6065     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.

```

6066     local walkable_syntax = (function(global_walkable_syntax)
6067       local local_walkable_syntax = {}
6068       for lhs, rule in pairs(global_walkable_syntax) do
6069         local_walkable_syntax[lhs] = util.table_copy(rule)
6070       end
6071       return local_walkable_syntax
6072     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.

```

6073     local current_extension_name = nil
6074     self.insert_pattern = function(selector, pattern, pattern_name)
6075       assert(pattern_name == nil or type(pattern_name) == "string")
6076       local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a)%s+(%a+)$")
6077       assert(lhs ~= nil,
6078         [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]

```

```

6079     .. selector .. [["]])
6080     assert(walkable_syntax[lhs] ~= nil,
6081           [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
6082     assert(pos == "before" or pos == "after" or pos == "instead of",
6083           [[Expected positional specifier "before", "after", or "instead of", not "]]
6084           .. pos .. [["]])
6085     local rule = walkable_syntax[lhs]
6086     local index = nil
6087     for current_index, current_rhs in ipairs(rule) do
6088       if type(current_rhs) == "string" and current_rhs == rhs then
6089         index = current_index
6090         if pos == "after" then
6091           index = index + 1
6092         end
6093         break
6094       end
6095     end
6096     assert(index ~= nil,
6097           [[Rule ]] .. lhs .. [[ -> ]] .. rhs
6098           .. [[ does not exist in markdown grammar]])
6099     local accountable_pattern
6100     if current_extension_name then
6101       accountable_pattern = { pattern, current_extension_name, pattern_name }
6102     else
6103       assert(type(pattern) == "string",
6104             [[reader->insert_pattern() was called outside an extension with ]]
6105             .. [[a PEG pattern instead of a rule name]])
6106       accountable_pattern = pattern
6107     end
6108     if pos == "instead of" then
6109       rule[index] = accountable_pattern
6110     else
6111       table.insert(rule, index, accountable_pattern)
6112     end
6113   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.

```

6114   local syntax =
6115     { "Blocks",
6116
6117       Blocks           = V("InitializeState")
6118                       * ( V("ExpectedJekyllData")
6119                           * (V("Blank")^0 / writer.interblocksep))^-
6120                       * V("Blank")^0
6121                       * V("Block")^-1

```



```

6122             * ( V("Blank")^0 / writer.interblocksep
6123             * V("Block"))^0
6124             * V("Blank")^0 * parsers.eof,
6125
6126     ExpectedJekyllData = parsers.fail,
6127
6128     Blank = parsers.Blank,
6129
6130     Blockquote = parsers.Blockquote,
6131     Verbatim = parsers.Verbatim,
6132     ThematicBreak = parsers.ThematicBreak,
6133     BulletList = parsers.BulletList,
6134     OrderedList = parsers.OrderedList,
6135     Heading = parsers.Heading,
6136     DisplayHtml = parsers.DisplayHtml,
6137     Paragraph = parsers.Paragraph,
6138     Plain = parsers.Plain,
6139     EndlineExceptions = parsers.EndlineExceptions,
6140
6141     Str = parsers.Str,
6142     Space = parsers.Space,
6143     OptionalIndent = parsers.OptionalIndent,
6144     Endline = parsers.Endline,
6145     U1OrStarLine = parsers.U1OrStarLine,
6146     Strong = parsers.Strong,
6147     Emph = parsers.Emph,
6148     Link = parsers.Link,
6149     Image = parsers.Image,
6150     Code = parsers.Code,
6151     AutoLinkUrl = parsers.AutoLinkUrl,
6152     AutoLinkEmail = parsers.AutoLinkEmail,
6153     AutoLinkRelativeReference
6154     = parsers.AutoLinkRelativeReference,
6155     InlineHtml = parsers.InlineHtml,
6156     HtmlEntity = parsers.HtmlEntity,
6157     EscapedChar = parsers.EscapedChar,
6158     Smart = parsers.Smart,
6159     Symbol = parsers.Symbol,
6160     SpecialChar = parsers.fail,
6161     InitializeState = parsers.succeed,
6162 }

```

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]`.

```

6163 self.update_rule = function(rule_name, get_pattern)
6164   assert(current_extension_name ~= nil)
6165   assert(syntax[rule_name] ~= nil,
6166     [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
6167   local previous_pattern
6168   local extension_name
6169   if walkable_syntax[rule_name] then
6170     local previous_accountable_pattern = walkable_syntax[rule_name][1]
6171     previous_pattern = previous_accountable_pattern[1]
6172     extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_name
6173   else
6174     previous_pattern = nil
6175     extension_name = current_extension_name
6176   end
6177   local pattern = get_pattern(previous_pattern)
6178   local accountable_pattern = { pattern, extension_name, rule_name }
6179   walkable_syntax[rule_name] = { accountable_pattern }
6180 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.

```

6181 local special_characters = {}
6182 self.add_special_character = function(c)
6183   table.insert(special_characters, c)
6184   syntax.SpecialChar = S(table.concat(special_characters, ""))
6185 end
6186
6187 self.add_special_character("*")
6188 self.add_special_character("`")
6189 self.add_special_character("[")
6190 self.add_special_character("]")
6191 self.add_special_character("<")
6192 self.add_special_character("!")
6193 self.add_special_character("\\")

```

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

```

6194 self.initialize_named_group = function(name, value)
6195   syntax.InitializeState = syntax.InitializeState
6196     * Cg(Ct("") / value, name)
6197 end

```

Apply syntax extensions.

```

6198 for _, extension in ipairs(extensions) do
6199   current_extension_name = extension.name
6200   extension.extend_writer(writer)
6201   extension.extend_reader(self)

```

```

6202     end
6203     current_extension_name = nil

```

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

```

6204     if options.debugExtensions then
6205         local sorted_lhs = {}
6206         for lhs, _ in pairs(walkable_syntax) do
6207             table.insert(sorted_lhs, lhs)
6208         end
6209         table.sort(sorted_lhs)
6210
6211         local output_lines = {"{"}
6212         for lhs_index, lhs in ipairs(sorted_lhs) do
6213             local encoded_lhs = util.encode_json_string(lhs)
6214             table.insert(output_lines, [{" "] .. encoded_lhs .. [{": "}]})
6215             local rule = walkable_syntax[lhs]
6216             for rhs_index, rhs in ipairs(rule) do
6217                 local human_readable_rhs
6218                 if type(rhs) == "string" then
6219                     human_readable_rhs = rhs
6220                 else
6221                     local pattern_name
6222                     if rhs[3] then
6223                         pattern_name = rhs[3]
6224                     else
6225                         pattern_name = "Anonymous Pattern"
6226                     end
6227                     local extension_name = rhs[2]
6228                     human_readable_rhs = pattern_name .. [{" ("} .. extension_name .. [{")"}]
6229                 end
6230                 local encoded_rhs = util.encode_json_string(human_readable_rhs)
6231                 local output_line = [{" "] .. encoded_rhs
6232                 if rhs_index < #rule then
6233                     output_line = output_line .. [{","}
6234                 end
6235                 table.insert(output_lines, output_line)
6236             end
6237             local output_line = [{" "]
6238             if lhs_index < #sorted_lhs then
6239                 output_line = output_line .. [{","}
6240             end
6241             table.insert(output_lines, output_line)
6242         end
6243         table.insert(output_lines, [{"}"}])
6244
6245         local output = table.concat(output_lines, "\n")

```

```

6246     local output_filename = options.debugExtensionsFileName
6247     local output_file = assert(io.open(output_filename, "w"),
6248         [[Could not open file ]] .. output_filename .. [[ for writing]])
6249     assert(output_file:write(output))
6250     assert(output_file:close())
6251     end

```

Duplicate the `Inline` rule as `IndentedInline` with the right-hand-side terminal symbol `Space` replaced with `OptionalIndent`.

```

6252     walkable_syntax["IndentedInline"] = util.table_copy(
6253         walkable_syntax["Inline"])
6254     self.insert_pattern(
6255         "IndentedInline instead of Space",
6256         "OptionalIndent")

```

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`.

```

6257     for lhs, rule in pairs(walkable_syntax) do
6258         syntax[lhs] = parsers.fail
6259         for _, rhs in ipairs(rule) do
6260             local pattern

```

Although the interface of the `reader->insert_pattern` method does 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.

```

6261             if type(rhs) == "string" then
6262                 pattern = V(rhs)
6263             else
6264                 pattern = rhs[1]
6265                 if type(pattern) == "string" then
6266                     pattern = V(pattern)
6267                 end
6268             end
6269             syntax[lhs] = syntax[lhs] + pattern
6270         end
6271     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.

```

6272     if options.underscores then
6273         self.add_special_character("_")
6274     end
6275
6276     if not options.codeSpans then
6277         syntax.Code = parsers.fail

```

```

6278     end
6279
6280     if not options.html then
6281         syntax.DisplayHtml = parsers.fail
6282         syntax.InlineHtml = parsers.fail
6283         syntax.HtmlEntity = parsers.fail
6284     else
6285         self.add_special_character("&")
6286     end
6287
6288     if options.preserveTabs then
6289         options.stripIndent = false
6290     end
6291
6292     if not options.smartEllipses then
6293         syntax.Smart = parsers.fail
6294     else
6295         self.add_special_character(".")
6296     end
6297
6298     if not options.relativeReferences then
6299         syntax.AutoLinkRelativeReference = parsers.fail
6300     end
6301
6302     local blocks_nested_t = util.table_copy(syntax)
6303     blocks_nested_t.ExpectedJekyllData = parsers.fail
6304     parsers.blocks_nested = Ct(blocks_nested_t)
6305
6306     parsers.blocks = Ct(syntax)
6307
6308     local inlines_t = util.table_copy(syntax)
6309     inlines_t[1] = "Inlines"
6310     inlines_t.Inlines = V("InitializeState")
6311         * parsers.Inline~0
6312         * ( parsers.spacing~0
6313         * parsers.eof / "" )
6314     parsers.inlines = Ct(inlines_t)
6315
6316     local inlines_no_link_t = util.table_copy(inlines_t)
6317     inlines_no_link_t.Link = parsers.fail
6318     parsers.inlines_no_link = Ct(inlines_no_link_t)
6319
6320     local inlines_no_inline_note_t = util.table_copy(inlines_t)
6321     inlines_no_inline_note_t.InlineNote = parsers.fail
6322     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6323
6324     local inlines_no_html_t = util.table_copy(inlines_t)

```

```

6325     inlines_no_html_t.DisplayHtml = parsers.fail
6326     inlines_no_html_t.InlineHtml = parsers.fail
6327     inlines_no_html_t.HtmlEntity = parsers.fail
6328     parsers.inlines_no_html = Ct(inlines_no_html_t)
6329
6330     local inlines_nbsp_t = util.table_copy(inlines_t)
6331     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6332     inlines_nbsp_t.Space = parsers.NonbreakingSpace
6333     parsers.inlines_nbsp = Ct(inlines_nbsp_t)

```

Return a function that converts markdown string `input` into a plain T_EX output and returns it. Note that the converter assumes that the input has UNIX line endings.

```

6334     return function(input)
6335         references = {}

```

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.

```

6336     local opt_string = {}
6337     for k, _ in pairs(defaultOptions) do
6338         local v = options[k]
6339         if type(v) == "table" then
6340             for _, i in ipairs(v) do
6341                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)
6342             end
6343         elseif k ~= "cacheDir" then
6344             opt_string[#opt_string+1] = k .. "=" .. tostring(v)
6345         end
6346     end
6347     table.sort(opt_string)
6348     local salt = table.concat(opt_string, ",") .. "," .. metadata.version
6349     local output

```

If we cache markdown documents, produce the cache file and transform its filename to plain T_EX output via the `writer->pack` method.

```

6350     local function convert(input)
6351         local document = self.parser_functions.parse_blocks(input)
6352         return util.ropetostring(writer.document(document))
6353     end
6354     if options.eagerCache or options.finalizeCache then
6355         local name = util.cache(options.cacheDir, input, salt, convert,
6356                               ".md" .. writer.suffix)
6357         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

6358     else
6359         output = convert(input)
6360     end

```

If the `finalizeCache` option is enabled, populate the frozen cache in the file `frozenCacheFileName` with an entry for markdown document number `frozenCacheCounter`.

```
6361     if options.finalizeCache then
6362         local file, mode
6363         if options.frozenCacheCounter > 0 then
6364             mode = "a"
6365         else
6366             mode = "w"
6367         end
6368         file = assert(io.open(options.frozenCacheFileName, mode),
6369             [[Could not open file "]] .. options.frozenCacheFileName
6370             .. [[ for writing]])
6371         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
6372             .. [[markdownFrozenCache]] .. options.frozenCacheCounter
6373             .. [[\\endcsname{}}] .. output .. [[]] .. "\\n"))
6374         assert(file:close())
6375     end
6376     return output
6377 end
6378 end
6379 return self
6380 end
```

3.1.6 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.

```
6381 M.extensions = {}
```

3.1.6.1 Bracketed Spans The `extensions.bracketed_spans` function implements the Pandoc bracketed spans syntax extension.

```
6382 M.extensions.bracketed_spans = function()
6383     return {
6384         name = "built-in bracketed_spans syntax extension",
6385         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.

```
6386         function self.span(s, attr)
6387             return {"\\markdownRendererBracketedSpanAttributeContextBegin",
6388                 self.attributes(attr),
```

```

6389             s,
6390             "\\markdownRendererBracketedSpanAttributeContextEnd{}}"
6391         end
6392     end, extend_reader = function(self)
6393         local parsers = self.parsers
6394         local writer = self.writer
6395
6396         local Span = parsers.between(parsers.Inline,
6397                                     parsers.lbracket,
6398                                     parsers.rbracket)
6399             * Ct(parsers.attributes)
6400             / writer.span
6401
6402         self.insert_pattern("Inline after Emph",
6403                             Span, "Span")
6404     end
6405 }
6406 end

```

3.1.6.2 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.

```

6407 M.extensions.citations = function(citation_nbsps)

```

Define table `escaped_citation_chars` containing the characters to escape in citations.

```

6408     local escaped_citation_chars = {
6409         [{""] = "\\markdownRendererLeftBrace{]",
6410         ["}"] = "\\markdownRendererRightBrace{]",
6411         [%"] = "\\markdownRendererPercentSign{]",
6412         [\\"] = "\\markdownRendererBackslash{]",
6413         [#"] = "\\markdownRendererHash{]",
6414     }
6415     return {
6416         name = "built-in citations syntax extension",
6417         extend_writer = function(self)
6418             local options = self.options
6419

```

Use the `escaped_citation_chars` to create the `escape_citation` escaper functions.

```

6420         local escape_citation = util.escaper(
6421             escaped_citation_chars,
6422             self.escaped_minimal_strings)

```


Define `writer->citation` as a function that will transform an input citation name `c` to the output format. If option `hybrid` is enabled, use the `writer->escape_minimal` function. Otherwise, use the `escape_citation` function.

```
6423     if options.hybrid then
6424         self.citation = self.escape_minimal
6425     else
6426         self.citation = escape_citation
6427     end
```

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.

```
6428     function self.citations(text_cites, cites)
6429         local buffer = {"\markdownRenderer", text_cites and "TextCite" or "Cite",
6430             "{", #cites, "}"}
6431         for _,cite in ipairs(cites) do
6432             buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6433                 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
6434         end
6435         return buffer
6436     end
6437 end, extend_reader = function(self)
6438     local parsers = self.parsers
6439     local writer = self.writer
6440
6441     local citation_chars
6442         = parsers.alphanumeric
6443         + S("#$%&-+<>~/_")
6444
6445     local citation_name
6446         = Cs(parsers.dash^-1) * parsers.at
6447         * Cs(citation_chars
6448             * (((citation_chars + parsers.internal_punctuation
6449                 - parsers.comma - parsers.semicolon)
```

```

6450         * -#((parsers.internal_punctuation - parsers.comma
6451           - parsers.semicolon)^0
6452         * -(citation_chars + parsers.internal_punctuation
6453           - parsers.comma - parsers.semicolon))^0
6454         * citation_chars)^-1)
6455
6456     local citation_body_prenote
6457         = Cs((parsers.alphanumeric^1
6458           + parsers.bracketed
6459           + parsers.inticks
6460           + (parsers.anyescaped
6461             - (parsers.rbracket + parsers.blankline^2))
6462           - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6463
6464     local citation_body_postnote
6465         = Cs((parsers.alphanumeric^1
6466           + parsers.bracketed
6467           + parsers.inticks
6468           + (parsers.anyescaped
6469             - (parsers.rbracket + parsers.semicolon
6470               + parsers.blankline^2))
6471           - (parsers.spnl * parsers.rbracket))^0)
6472
6473     local citation_body_chunk
6474         = citation_body_prenote
6475         * parsers.spnl * citation_name
6476         * (parsers.internal_punctuation - parsers.semicolon)^-
1
6477         * parsers.spnl * citation_body_postnote
6478
6479     local citation_body
6480         = citation_body_chunk
6481         * (parsers.semicolon * parsers.spnl
6482         * citation_body_chunk)^0
6483
6484     local citation_headless_body_postnote
6485         = Cs((parsers.alphanumeric^1
6486           + parsers.bracketed
6487           + parsers.inticks
6488           + (parsers.anyescaped
6489             - (parsers.rbracket + parsers.at
6490               + parsers.semicolon + parsers.blankline^2))
6491           - (parsers.spnl * parsers.rbracket))^0)
6492
6493     local citation_headless_body
6494         = citation_headless_body_postnote
6495         * (parsers.sp * parsers.semicolon * parsers.spnl

```

```

6496             * citation_body_chunk)^0
6497
6498     local citations
6499         = function(text_cites, raw_cites)
6500     local function normalize(str)
6501         if str == "" then
6502             str = nil
6503         else
6504             str = (citation_nbsps and
6505                 self.parser_functions.parse_inlines_nbsp or
6506                 self.parser_functions.parse_inlines)(str)
6507         end
6508         return str
6509     end
6510
6511     local cites = {}
6512     for i = 1,#raw_cites,4 do
6513         cites[#cites+1] = {
6514             prenote = normalize(raw_cites[i]),
6515             suppress_author = raw_cites[i+1] == "-",
6516             name = writer.citation(raw_cites[i+2]),
6517             postnote = normalize(raw_cites[i+3]),
6518         }
6519     end
6520     return writer.citations(text_cites, cites)
6521 end
6522
6523 local TextCitations
6524     = Ct((parsers.spnl
6525         * Cc("")
6526         * citation_name
6527         * ((parsers.spnl
6528             * parsers.lbracket
6529             * citation_headless_body
6530             * parsers.rbracket) + Cc("")))^1)
6531 / function(raw_cites)
6532     return citations(true, raw_cites)
6533 end
6534
6535 local ParenthesizedCitations
6536     = Ct((parsers.spnl
6537         * parsers.lbracket
6538         * citation_body
6539         * parsers.rbracket)^1)
6540 / function(raw_cites)
6541     return citations(false, raw_cites)
6542 end

```

```

6543
6544     local Citations = TextCitations + ParenthesizedCitations
6545
6546     self.insert_pattern("Inline after Emph",
6547                         Citations, "Citations")
6548
6549     self.add_special_character("@")
6550     self.add_special_character("-")
6551 end
6552 }
6553 end

```

3.1.6.3 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.

```

6554 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.

```

6555     local languages_json = (function()
6556         local base, prev, curr
6557         for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
6558             local file = io.open(pathname, "r")
6559             if not file then goto continue end
6560             local input = assert(file:read("*a"))
6561             assert(file:close())
6562             local json = input:gsub('"[\n]-'):','[%1]=')
6563             curr = load("_ENV = {}; return "..json)()
6564             if type(curr) == "table" then
6565                 if base == nil then
6566                     base = curr
6567                 else
6568                     setmetatable(prev, { __index = curr })
6569                 end
6570                 prev = curr
6571             end
6572             ::continue::
6573         end
6574         return base or {}
6575     end)()
6576
6577     return {
6578         name = "built-in content_blocks syntax extension",

```

```
6579     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.

```
6580     function self.contentblock(src,suf,type,tit)
6581         if not self.is_writing then return "" end
6582         src = src..".."..suf
6583         suf = suf:lower()
6584         if type == "onlineimage" then
6585             return {"\\markdownRendererContentBlockOnlineImage{" ,suf,"} ",
6586                   "{" ,self.string(src),"} ",
6587                   "{" ,self.uri(src),"} ",
6588                   "{" ,self.string(tit or ""),"} "}
6589         elseif languages_json[suf] then
6590             return {"\\markdownRendererContentBlockCode{" ,suf,"} ",
6591                   "{" ,self.string(languages_json[suf]),"} ",
6592                   "{" ,self.string(src),"} ",
6593                   "{" ,self.uri(src),"} ",
6594                   "{" ,self.string(tit or ""),"} "}
6595         else
6596             return {"\\markdownRendererContentBlock{" ,suf,"} ",
6597                   "{" ,self.string(src),"} ",
6598                   "{" ,self.uri(src),"} ",
6599                   "{" ,self.string(tit or ""),"} "}
6600         end
6601     end
6602 end, extend_reader = function(self)
6603     local parsers = self.parsers
6604     local writer = self.writer
6605
6606     local contentblock_tail
6607         = parsers.optionaltitle
6608         * (parsers.newline + parsers.eof)
6609
6610     -- case insensitive online image suffix:
6611     local onlineimagesuffix
6612         = (function(...)
6613             local parser = nil
6614             for _, suffix in ipairs({...}) do
6615                 local pattern=nil
6616                 for i=1,#suffix do
6617                     local char=suffix:sub(i,i)
6618                     char = S(char:lower()..char:upper())
6619                     if pattern == nil then
6620                         pattern = char
```

```

6621         else
6622             pattern = pattern * char
6623         end
6624     end
6625     if parser == nil then
6626         parser = pattern
6627     else
6628         parser = parser + pattern
6629     end
6630 end
6631 return parser
6632 end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6633
6634 -- online image url for iA Writer content blocks with mandatory suffix,
6635 -- allowing nested brackets:
6636 local onlineimageurl
6637     = (parsers.less
6638         * Cs((parsers.anyescaped
6639             - parsers.more
6640             - #(parsers.period
6641                 * onlineimagesuffix
6642                 * parsers.more
6643                 * contentblock_tail)))^0)
6644         * parsers.period
6645         * Cs(onlineimagesuffix)
6646         * parsers.more
6647         + (Cs((parsers.inparens
6648             + (parsers.anyescaped
6649                 - parsers.spacing
6650                 - parsers.rparent
6651                 - #(parsers.period
6652                     * onlineimagesuffix
6653                     * contentblock_tail)))^0)
6654             * parsers.period
6655             * Cs(onlineimagesuffix))
6656         ) * Cc("onlineimage")
6657
6658 -- filename for iA Writer content blocks with mandatory suffix:
6659 local localfilepath
6660     = parsers.slash
6661     * Cs((parsers.anyescaped
6662         - parsers.tab
6663         - parsers.newline
6664         - #(parsers.period
6665             * parsers.alphanumeric^1
6666             * contentblock_tail))^1)
6667     * parsers.period

```

```

6668             * Cs(parsers.alphanumeric~1)
6669             * Cc("localfile")
6670
6671     local ContentBlock
6672         = parsers.leader
6673         * (localfilepath + onlineimageurl)
6674         * contentblock_tail
6675         / writer.contentblock
6676
6677     self.insert_pattern("Block before Blockquote",
6678                       ContentBlock, "ContentBlock")
6679 end
6680 }
6681 end

```

3.1.6.4 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.

```

6682 M.extensions.definition_lists = function(tight_lists)
6683   return {
6684     name = "built-in definition_lists syntax extension",
6685     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.

```

6686     local function dlitem(term, defs)
6687       local retVal = {"\\markdownRendererDlItem{" ,term,"}"}
6688       for _, def in ipairs(defs) do
6689         retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin " ,def,
6690                             "\\markdownRendererDlDefinitionEnd "}
6691       end
6692       retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
6693       return retVal
6694     end
6695
6696     function self.definitionlist(items,tight)
6697       if not self.is_writing then return "" end
6698       local buffer = {}
6699       for _,item in ipairs(items) do
6700         buffer[#buffer + 1] = dlitem(item.term, item.definitions)
6701       end
6702       if tight and tight_lists then
6703         return {"\\markdownRendererDlBeginTight\\n", buffer,
6704               "\\n\\markdownRendererDlEndTight"}

```

```

6705     else
6706         return {"\\markdownRendererDlBegin\n", buffer,
6707             "\\n\\markdownRendererDlEnd"}
6708     end
6709 end
6710 end, extend_reader = function(self)
6711     local parsers = self.parsers
6712     local writer = self.writer
6713
6714     local defstartchar = S("~:")
6715
6716     local defstart = ( defstartchar * #parsers.spacing
6717         * (parsers.tab + parsers.space^-
3)
6718         + parsers.space * defstartchar * #parsers.spacing
6719         * (parsers.tab + parsers.space^-
2)
6720         + parsers.space * parsers.space * defstartchar
6721         * #parsers.spacing
6722         * (parsers.tab + parsers.space^-
1)
6723         + parsers.space * parsers.space * parsers.space
6724         * defstartchar * #parsers.spacing
6725     )
6726
6727     local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
6728
6729     local function definition_list_item(term, defs, _)
6730         return { term = self.parser_functions.parse_inlines(term),
6731             definitions = defs }
6732     end
6733
6734     local DefinitionListItemLoose
6735         = C(parsers.line) * parsers.skipblanklines
6736         * Ct((defstart
6737             * parsers.indented_blocks(dlchunk)
6738             / self.parser_functions.parse_blocks_nested^1)
6739         * Cc(false) / definition_list_item
6740
6741     local DefinitionListItemTight
6742         = C(parsers.line)
6743         * Ct((defstart * dlchunk
6744             / self.parser_functions.parse_blocks_nested^1)
6745         * Cc(true) / definition_list_item
6746
6747     local DefinitionList
6748         = ( Ct(DefinitionListItemLoose^1) * Cc(false)

```



```

6749         + Ct(DefinitionListItemTight^1)
6750         * (parsers.skipblanklines
6751           * -DefinitionListItemLoose * Cc(true))
6752         ) / writer.definitionlist
6753
6754     self.insert_pattern("Block after Heading",
6755                       DefinitionList, "DefinitionList")
6756 end
6757 }
6758 end

```

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

```

6759 M.extensions.fancy_lists = function()
6760   return {
6761     name = "built-in fancy_lists syntax extension",
6762     extend_writer = function(self)
6763       local options = self.options
6764

```

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.

```

6765     function self.fancylist(items,tight,startnum,numstyle,numdelim)
6766       if not self.is_writing then return "" end
6767       local buffer = {}
6768       local num = startnum
6769       for _,item in ipairs(items) do
6770         buffer[#buffer + 1] = self.fancyitem(item,num)
6771         if num ~= nil then
6772           num = num + 1
6773         end

```

```

6774     end
6775     local contents = util.intersperse(buffer, "\n")
6776     if tight and options.tightLists then
6777         return {"\\markdownRendererFancyOlBeginTight{" ,
6778             numstyle, "}{" , numdelim, "}" , contents,
6779             "\n\\markdownRendererFancyOlEndTight "}
6780     else
6781         return {"\\markdownRendererFancyOlBegin{" ,
6782             numstyle, "}{" , numdelim, "}" , contents,
6783             "\n\\markdownRendererFancyOlEnd "}
6784     end
6785     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.

```

6786     function self.fancyitem(s,num)
6787         if num ~= nil then
6788             return {"\\markdownRendererFancyOlItemWithNumber{" , num, "}" , s,
6789                 "\\markdownRendererFancyOlItemEnd "}
6790         else
6791             return {"\\markdownRendererFancyOlItem " , s, "\\markdownRendererFancyOlItemEnd "
6792         end
6793     end
6794 end, extend_reader = function(self)
6795     local parsers = self.parsers
6796     local options = self.options
6797     local writer = self.writer
6798
6799     local label = parsers.dig + parsers.letter
6800     local numdelim = parsers.period + parsers.rparent
6801     local enumerator = C(label^3 * numdelim) * #parsers.spacing
6802         + C(label^2 * numdelim) * #parsers.spacing
6803         * (parsers.tab + parsers.space^1)
6804         + C(label * numdelim) * #parsers.spacing
6805         * (parsers.tab + parsers.space^-
2)
6806         + parsers.space * C(label^2 * numdelim)
6807         * #parsers.spacing
6808         + parsers.space * C(label * numdelim)
6809         * #parsers.spacing
6810         * (parsers.tab + parsers.space^-
1)
6811         + parsers.space * parsers.space * C(label^1
6812         * numdelim) * #parsers.spacing
6813     local starter = parsers.bullet + enumerator
6814

```

```

6815     local NestedList = Cs((parsers.optionallyindentedline
6816         - starter)^1)
6817         / function(a) return "\001"..a end
6818
6819     local ListBlockLine = parsers.optionallyindentedline
6820         - parsers.blankline - (parsers.indent^-1
6821         * starter)
6822
6823     local ListBlock = parsers.line * ListBlockLine^0
6824
6825     local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6826         * ListBlock
6827
6828     local TightListItem = function(starter)
6829         return -parsers.ThematicBreak
6830             * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
6831             / self.parser_functions.parse_blocks_nested)
6832             * -(parsers.blanklines * parsers.indent)
6833     end
6834
6835     local LooseListItem = function(starter)
6836         return -parsers.ThematicBreak
6837             * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
6838             * (NestedList + ListContinuationBlock^0)
6839             * (parsers.blanklines / "\n\n")
6840             ) / self.parser_functions.parse_blocks_nested
6841     end
6842
6843     local function roman2number(roman)
6844         local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
6845         local numeral = 0
6846
6847         local i = 1
6848         local len = string.len(roman)
6849         while i < len do
6850             local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
6851             if z1 < z2 then
6852                 numeral = numeral + (z2 - z1)
6853                 i = i + 2
6854             else
6855                 numeral = numeral + z1
6856                 i = i + 1
6857             end
6858         end
6859         if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
6860         return numeral

```

```

6861     end
6862
6863     local function sniffstyle(itemprefix)
6864         local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])*")
6865         local numdelim
6866         if delimend == ")" then
6867             numdelim = "OneParen"
6868         elseif delimend == "." then
6869             numdelim = "Period"
6870         else
6871             numdelim = "Default"
6872         end
6873         numstr = numstr or itemprefix
6874
6875         local num
6876         num = numstr:match("^([IVXL]+)")
6877         if num then
6878             return roman2number(num), "UpperRoman", numdelim
6879         end
6880         num = numstr:match("^([ivxl]+)")
6881         if num then
6882             return roman2number(string.upper(num)), "LowerRoman", numdelim
6883         end
6884         num = numstr:match("^([A-Z])")
6885         if num then
6886             return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
6887         end
6888         num = numstr:match("^([a-z])")
6889         if num then
6890             return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
6891         end
6892         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
6893     end
6894
6895     local function fancylist(items,tight,start)
6896         local startnum, numstyle, numdelim = sniffstyle(start)
6897         return writer.fancylist(items,tight,
6898             options.startNumber and startnum,
6899             numstyle or "Decimal",
6900             numdelim or "Default")
6901     end
6902
6903     local FancyList = Cg(enumerator, "listtype") *
6904         ( Ct(TightListItem(Cb("listtype")))
6905           * TightListItem(enumerator)^0)
6906         * Cc(true) * parsers.skipblanklines * -enumerator
6907         + Ct(LooseListItem(Cb("listtype")))

```

```

6908             * LooseListItem(enumerator)^0)
6909             * Cc(false) * parsers.skipblanklines
6910             ) * Cb("listtype") / fancylist
6911
6912     self.update_rule("OrderedList", function() return FancyList end)
6913 end
6914 }
6915 end

```

3.1.6.6 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.

```

6916 M.extensions.fenced_code = function(blank_before_code_fence)
6917   return {
6918     name = "built-in fenced_code syntax extension",
6919     extend_writer = function(self)
6920       local options = self.options
6921

```

Define `writer->codeFence` as a function that will transform an input fenced code block `s` with the infostring `i` to the output format.

```

6922     function self.fencedCode(s, i)
6923       if not self.is_writing then return "" end
6924       local name = util.cache_verbatim(options.cacheDir, s)
6925       return {"\\markdownRendererInputFencedCode{" ,
6926             name,"}{" ,self.string(i),"}"}
6927     end
6928   end, extend_reader = function(self)
6929     local parsers = self.parsers
6930     local writer = self.writer
6931
6932     local FencedCode = (parsers.TildeFencedCode
6933                       + parsers.BacktickFencedCode)
6934                       / function(infostring, code)
6935                           local expanded_code = self.expandtabs(code)
6936                           return writer.fencedCode(expanded_code,
6937                                                     infostring)
6938                       end
6939
6940   self.insert_pattern("Block after Verbatim",
6941                     FencedCode, "FencedCode")
6942
6943   local fencestart
6944   if blank_before_code_fence then
6945     fencestart = parsers.fail

```

```

6946     else
6947         fencestart = parsers.fencehead(parsers.backtick,
6948                                         parsers.backtick_infostring)
6949         + parsers.fencehead(parsers.tilde,
6950                             parsers.tilde_infostring)
6951     end
6952
6953     self.update_rule("EndlineExceptions", function(previous_pattern)
6954         if previous_pattern == nil then
6955             previous_pattern = parsers.EndlineExceptions
6956         end
6957         return previous_pattern + fencestart
6958     end)
6959
6960     self.add_special_character("~")
6961 end
6962 }
6963 end

```

3.1.6.7 Fenced Divs The `extensions.fenced_divs` function implements the Pandoc fenced divs 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.

```

6964 M.extensions.fenced_divs = function(blank_before_div_fence)
6965     return {
6966         name = "built-in fenced_divs syntax extension",
6967         extend_writer = function(self)

```

Define `writer->div` as a function that will transform an input fenced div with content `c` and with attributes `attr` to the output format.

```

6968         function self.div(c, attr)
6969             return {"\markdownRendererFencedDivAttributeContextBegin",
6970                     self.attributes(attr),
6971                     c,
6972                     "\markdownRendererFencedDivAttributeContextEnd"}
6973         end
6974     end, extend_reader = function(self)
6975         local parsers = self.parsers
6976         local writer = self.writer

```

Define basic patterns for matching the opening and the closing tag of a div.

```

6977         local fenced_div_infostring
6978             = C((parsers.linechar
6979                 - ( parsers.spacechar^1
6980                   * parsers.colon^1))^1)
6981
6982         local fenced_div_begin = parsers.nonindentospace

```

```

6983         * parsers.colon^3
6984         * parsers.optionalspace
6985         * fenced_div_infostring
6986         * ( parsers.spacechar^1
6987           * parsers.colon^1)^0
6988         * parsers.optionalspace
6989         * (parsers.newline + parsers.eof)
6990
6991     local fenced_div_end = parsers.nonindentspace
6992         * parsers.colon^3
6993         * parsers.optionalspace
6994         * (parsers.newline + parsers.eof)

```

Initialize a named group named `div_level` for tracking how deep we are nested in divs.

```

6995     self.initialize_named_group("div_level", "0")
6996
6997     local function increment_div_level(increment)
6998         local function update_div_level(s, i, current_level) -- luacheck: ignore s i
6999             current_level = tonumber(current_level)
7000             local next_level = tostring(current_level + increment)
7001             return true, next_level
7002         end
7003
7004         return Cg( Cmt(Cb("div_level"), update_div_level)
7005                 , "div_level")
7006     end
7007
7008     local FencedDiv = fenced_div_begin * increment_div_level(1)
7009         * parsers.skipblanklines
7010         * Ct( (V("Block") - fenced_div_end)^-1
7011             * ( parsers.blanklines
7012               / function()
7013                   return writer.interblocksep
7014               end
7015             * (V("Block") - fenced_div_end))^0)
7016         * parsers.skipblanklines
7017         * fenced_div_end * increment_div_level(-1)
7018     / function (infostring, div)
7019         local attr = lpeg.match(Ct(parsers.attributes), infostring)
7020         if attr == nil then
7021             attr = {".." .. infostring}
7022         end
7023         return div, attr
7024     end
7025     / writer.div
7026

```

```

7027     self.insert_pattern("Block after Verbatim",
7028                         FencedDiv, "FencedDiv")
7029
7030     self.add_special_character(":")

```

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.

```

7031     if not blank_before_div_fence then
7032         local function check_div_level(s, i, current_level) -- luacheck: ignore s i
7033             current_level = tonumber(current_level)
7034             return current_level > 0
7035         end
7036
7037         local is_inside_div = Cmt(Cb("div_level"), check_div_level)
7038         local fencestart = is_inside_div * fenced_div_end
7039         self.update_rule("EndlineExceptions", function(previous_pattern)
7040             if previous_pattern == nil then
7041                 previous_pattern = parsers.EndlineExceptions
7042             end
7043             return previous_pattern + fencestart
7044         end)
7045     end
7046 end
7047 }
7048 end

```

3.1.6.8 Header Attributes The `extensions.header_attributes` function implements the Pandoc header attributes syntax extension.

```

7049 M.extensions.header_attributes = function()
7050     return {
7051         name = "built-in header_attributes syntax extension",
7052         extend_writer = function()
7053             end, extend_reader = function(self)
7054                 local parsers = self.parsers
7055                 local writer = self.writer
7056
7057                 local AtxHeading = Cg(parsers.heading_start, "level")
7058                     * parsers.optionalspace
7059                     * (C(((parsers.linechar
7060                         - ((parsers.hash^1
7061                             * parsers.optionalspace
7062                             * parsers.attributes^-1
7063                             + parsers.attributes)
7064                             * parsers.optionalspace
7065                             * parsers.newline))
7066                         * (parsers.linechar

```



```

7067         - parsers.hash
7068         - parsers.lbrace)^0)^1)
7069     / self.parser_functions.parse_inlines)
7070 * Cg(Ct(parsers.newline
7071     + (parsers.hash^1
7072     * parsers.optionalspace
7073     * parsers.attributes^-1
7074     + parsers.attributes)
7075     * parsers.optionalspace
7076     * parsers.newline), "attributes")
7077 * Cb("level")
7078 * Cb("attributes")
7079 / writer.heading
7080
7081 local SettextHeading = #(parsers.line * S("=-"))
7082 * (C(((parsers.linechar
7083     - (parsers.attributes
7084     * parsers.optionalspace
7085     * parsers.newline))
7086     * (parsers.linechar
7087     - parsers.lbrace)^0)^1)
7088     / self.parser_functions.parse_inlines)
7089 * Cg(Ct(parsers.newline
7090     + (parsers.attributes
7091     * parsers.optionalspace
7092     * parsers.newline)), "attributes")
7093 * parsers.heading_level
7094 * Cb("attributes")
7095 * parsers.optionalspace
7096 * parsers.newline
7097 / writer.heading
7098
7099 local Heading = AtxHeading + SettextHeading
7100 self.update_rule("Heading", function() return Heading end)
7101 end
7102 }
7103 end

```

3.1.6.9 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.

```

7104 M.extensions.notes = function(notes, inline_notes)
7105   assert(notes or inline_notes)
7106   return {
7107     name = "built-in notes syntax extension",

```

```

7108     extend_writer = function(self)
    Define writer->note as a function that will transform an input note s to the
    output format.
7109         function self.note(s)
7110             return {"\\markdownRendererNote{",s,""}
7111         end
7112     end, extend_reader = function(self)
7113         local parsers = self.parsers
7114         local writer = self.writer
7115
7116         if inline_notes then
7117             local InlineNote
7118                 = parsers.circumflex
7119                 * (parsers.tag / self.parser_functions.parse_inlines_no_inline_no
7120                 / writer.note
7121
7122             self.insert_pattern("Inline after Emph",
7123                               InlineNote, "InlineNote")
7124         end
7125         if notes then
7126             local function strip_first_char(s)
7127                 return s:sub(2)
7128             end
7129
7130             local RawNoteRef
7131                 = #(parsers.lbracket * parsers.circumflex)
7132                 * parsers.tag / strip_first_char
7133
7134             local rawnotes = {}
7135
7136             -- like indirect_link
7137             local function lookup_note(ref)
7138                 return writer.defer_call(function()
7139                     local found = rawnotes[self.normalize_tag(ref)]
7140                     if found then
7141                         return writer.note(
7142                             self.parser_functions.parse_blocks_nested(found))
7143                     else
7144                         return {"[",
7145                             self.parser_functions.parse_inlines("^" .. ref), "]" }
7146                     end
7147                 end)
7148             end
7149
7150             local function register_note(ref,rawnote)
7151                 rawnotes[self.normalize_tag(ref)] = rawnote

```

```

7152         return ""
7153     end
7154
7155     local NoteRef = RawNoteRef / lookup_note
7156
7157     local NoteBlock
7158         = parsers.leader * RawNoteRef * parsers.colon
7159         * parsers.spnl * parsers.indented_blocks(parsers.chunk)
7160         / register_note
7161
7162     local Blank = NoteBlock + parsers.Blank
7163     self.update_rule("Blank", function() return Blank end)
7164
7165     self.insert_pattern("Inline after Emph",
7166         NoteRef, "NoteRef")
7167 end
7168
7169 self.add_special_character("^")
7170 end
7171 }
7172 end

```

3.1.6.10 Pipe Tables The `extensions.pipe_table` function implements the PHP Markdown table syntax extension (also known as pipe tables in Pandoc). When the `table_captions` parameter is `true`, the function also implements the Pandoc `table_captions` syntax extension for table captions.

```

7173 M.extensions.pipe_tables = function(table_captions)
7174
7175     local function make_pipe_table_rectangular(rows)
7176         local num_columns = #rows[2]
7177         local rectangular_rows = {}
7178         for i = 1, #rows do
7179             local row = rows[i]
7180             local rectangular_row = {}
7181             for j = 1, num_columns do
7182                 rectangular_row[j] = row[j] or ""
7183             end
7184             table.insert(rectangular_rows, rectangular_row)
7185         end
7186         return rectangular_rows
7187     end
7188
7189     local function pipe_table_row(allow_empty_first_column
7190         , nonempty_column
7191         , column_separator
7192         , column)

```

```

7193     local row_beginning
7194     if allow_empty_first_column then
7195         row_beginning = -- empty first column
7196                       #(parsers.spacechar^4
7197                        * column_separator)
7198                       * parsers.optionalspace
7199                       * column
7200                       * parsers.optionalspace
7201         -- non-empty first column
7202         + parsers.nonindentspace
7203         * nonempty_column^-1
7204         * parsers.optionalspace
7205     else
7206         row_beginning = parsers.nonindentspace
7207                       * nonempty_column^-1
7208                       * parsers.optionalspace
7209     end
7210
7211     return Ct(row_beginning
7212              * (-- single column with no leading pipes
7213                #(column_separator
7214                 * parsers.optionalspace
7215                 * parsers.newline)
7216                * column_separator
7217                * parsers.optionalspace
7218                -- single column with leading pipes or
7219                -- more than a single column
7220                + (column_separator
7221                  * parsers.optionalspace
7222                  * column
7223                  * parsers.optionalspace)^1
7224                  * (column_separator
7225                    * parsers.optionalspace)^-1))
7226     end
7227
7228     return {
7229         name = "built-in pipe_tables syntax extension",
7230         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.

```

7231     function self.table(rows, caption)
7232         if not self.is_writing then return "" end
7233         local buffer = {"\\markdownRendererTable{" ,
7234                       caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}"}
7235         local temp = rows[2] -- put alignments on the first row

```

```

7236     rows[2] = rows[1]
7237     rows[1] = temp
7238     for i, row in ipairs(rows) do
7239         table.insert(buffer, "{")
7240         for _, column in ipairs(row) do
7241             if i > 1 then -- do not use braces for alignments
7242                 table.insert(buffer, "{")
7243             end
7244             table.insert(buffer, column)
7245             if i > 1 then
7246                 table.insert(buffer, "}")
7247             end
7248         end
7249         table.insert(buffer, "}")
7250     end
7251     return buffer
7252 end
7253 end, extend_reader = function(self)
7254     local parsers = self.parsers
7255     local writer = self.writer
7256
7257     local table_hline_separator = parsers.pipe + parsers.plus
7258
7259     local table_hline_column = (parsers.dash
7260         - #(parsers.dash
7261             * (parsers.spacechar
7262                 + table_hline_separator
7263                 + parsers.newline)))^1
7264     * (parsers.colon * Cc("r")
7265         + parsers.dash * Cc("d"))
7266     + parsers.colon
7267     * (parsers.dash
7268         - #(parsers.dash
7269             * (parsers.spacechar
7270                 + table_hline_separator
7271                 + parsers.newline)))^1
7272     * (parsers.colon * Cc("c")
7273         + parsers.dash * Cc("l"))
7274
7275     local table_hline = pipe_table_row(false
7276         , table_hline_column
7277         , table_hline_separator
7278         , table_hline_column)
7279
7280     local table_caption_beginning = parsers.skipblanklines
7281         * parsers.nonindentSPACE
7282         * (P("Table")^-1 * parsers.colon)

```

```

7283             * parsers.optionalspace
7284
7285     local table_row = pipe_table_row(true
7286                                     , (C((parsers.linechar - parsers.pipe)^1)
7287                                       / self.parser_functions.parse_inlines)
7288                                     , parsers.pipe
7289                                     , (C((parsers.linechar - parsers.pipe)^0)
7290                                       / self.parser_functions.parse_inlines))
7291
7292     local table_caption
7293     if table_captions then
7294         table_caption = #table_caption_beginning
7295                       * table_caption_beginning
7296                       * Ct(parsers.IndentedInline^1)
7297                       * parsers.newline
7298     else
7299         table_caption = parsers.fail
7300     end
7301
7302     local PipeTable = Ct(table_row * parsers.newline
7303                       * table_hline
7304                       * (parsers.newline * table_row)^0)
7305                       / make_pipe_table_rectangular
7306                       * table_caption^-1
7307                       / writer.table
7308
7309     self.insert_pattern("Block after Blockquote",
7310                       PipeTable, "PipeTable")
7311 end
7312 }
7313 end

```

3.1.6.11 Raw Attributes The `extensions.raw_attribute` function implements the Pandoc raw attribute syntax extension.

```

7314 M.extensions.raw_attribute = function()
7315     return {
7316         name = "built-in raw_attribute syntax extension",
7317         extend_writer = function(self)
7318             local options = self.options
7319

```

Define `writer->rawInline` as a function that will transform an input inline raw span `s` with the raw attribute `attr` to the output format.

```

7320     function self.rawInline(s, attr)
7321         if not self.is_writing then return "" end
7322         local name = util.cache_verbatim(options.cacheDir, s)
7323         return {"\\markdownRendererInputRawInline{" ,

```

```

7324             name,"}{" , self.string(attr),"}"}
7325         end
7326
7327         if options.fencedCode then
            Define writer->rawBlock as a function that will transform an input raw block s
            with the raw attribute attr to the output format.
7328             function self.rawBlock(s, attr)
7329                 if not self.is_writing then return "" end
7330                 local name = util.cache_verbatim(options.cacheDir, s)
7331                 return {"\\markdownRendererInputRawBlock{" ,
7332                     name,"}{" , self.string(attr),"}"}
7333             end
7334         end
7335     end, extend_reader = function(self)
7336         local options = self.options
7337         local writer = self.writer
7338
7339         local raw_attribute = parsers.lbrace
7340             * parsers.optionalspace
7341             * parsers.equal
7342             * C(parsers.attribute_key)
7343             * parsers.optionalspace
7344             * parsers.rbrace
7345
7346         local RawInline = parsers.inticks
7347             * raw_attribute
7348             / writer.rawInline
7349
7350         self.insert_pattern("Inline before Code",
7351             RawInline, "RawInline")
7352
7353         if options.fencedCode then
7354             local RawBlock = (parsers.TildeFencedCode
7355                 + parsers.BacktickFencedCode)
7356                 / function(infostring, code)
7357                     local expanded_code = self.expandtabs(code)
7358                     local attr = lpeg.match(raw_attribute, infostring)
7359                     if attr then
7360                         return writer.rawBlock(expanded_code, attr)
7361                     else
7362                         return writer.fencedCode(expanded_code,
7363                             infostring)
7364                     end
7365                 end
7366
7367         self.insert_pattern("Block after Verbatim",

```

```

7368             RawBlock, "RawBlock")
7369         end
7370     end
7371 }
7372 end

```

3.1.6.12 Strike-Through The `extensions.strike_through` function implements the Pandoc strike-through syntax extension.

```

7373 M.extensions.strike_through = function()
7374     return {
7375         name = "built-in strike_through syntax extension",
7376         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.

```

7377             function self.strike_through(s)
7378                 return {"\\markdownRendererStrikeThrough{" ,s,"}"}
7379             end
7380         end, extend_reader = function(self)
7381             local parsers = self.parsers
7382             local writer = self.writer
7383
7384             local StrikeThrough = (
7385                 parsers.between(parsers.Inline, parsers.doubletildes,
7386                               parsers.doubletildes)
7387             ) / writer.strike_through
7388
7389             self.insert_pattern("Inline after Emph",
7390                               StrikeThrough, "StrikeThrough")
7391
7392             self.add_special_character("~")
7393         end
7394     }
7395 end

```

3.1.6.13 Subscripts The `extensions.subscripts` function implements the Pandoc subscript syntax extension.

```

7396 M.extensions.subscripts = function()
7397     return {
7398         name = "built-in subscripts syntax extension",
7399         extend_writer = function(self)

```

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

```

7400             function self.subscript(s)
7401                 return {"\\markdownRendererSubscript{" ,s,"}"}

```



```

7402     end
7403 end, extend_reader = function(self)
7404     local parsers = self.parsers
7405     local writer = self.writer
7406
7407     local Subscript = (
7408         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
7409     ) / writer.subscript
7410
7411     self.insert_pattern("Inline after Emph",
7412                         Subscript, "Subscript")
7413
7414     self.add_special_character("~")
7415 end
7416 }
7417 end

```

3.1.6.14 Superscripts The `extensions.superscripts` function implements the Pandoc superscript syntax extension.

```

7418 M.extensions.superscripts = function()
7419     return {
7420         name = "built-in superscripts syntax extension",
7421         extend_writer = function(self)

```

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

```

7422         function self.superscript(s)
7423             return {"\\markdownRendererSuperscript{" ,s,""}"}
7424         end
7425     end, extend_reader = function(self)
7426         local parsers = self.parsers
7427         local writer = self.writer
7428
7429         local Superscript = (
7430             parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
7431         ) / writer.superscript
7432
7433         self.insert_pattern("Inline after Emph",
7434                             Superscript, "Superscript")
7435
7436         self.add_special_character("^")
7437     end
7438 }
7439 end

```

3.1.6.15 YAML Metadata The `extensions.jekyll_data` function implements the Pandoc `yml_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.

```
7440 M.extensions.jekyll_data = function(expect_jekyll_data)
7441   return {
7442     name = "built-in jekyll_data syntax extension",
7443     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`.

```
7444     function self.jekyllData(d, t, p)
7445       if not self.is_writing then return "" end
7446
7447       local buf = {}
7448
7449       local keys = {}
7450       for k, _ in pairs(d) do
7451         table.insert(keys, k)
7452       end
7453       table.sort(keys)
7454
7455       if not p then
7456         table.insert(buf, "\\markdownRendererJekyllDataBegin")
7457       end
7458
7459       if #d > 0 then
7460         table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
7461         table.insert(buf, self.uri(p or "null"))
7462         table.insert(buf, "{")
7463         table.insert(buf, #keys)
7464         table.insert(buf, ")")
7465       else
7466         table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
7467         table.insert(buf, self.uri(p or "null"))
7468         table.insert(buf, "{")
7469         table.insert(buf, #keys)
7470         table.insert(buf, ")")
7471       end
7472
7473       for _, k in ipairs(keys) do
7474         local v = d[k]
7475         local typ = type(v)
```

```

7476         k = tostring(k or "null")
7477     if typ == "table" and next(v) ~= nil then
7478         table.insert(
7479             buf,
7480             self.jekyllData(v, t, k)
7481         )
7482     else
7483         k = self.uri(k)
7484         v = tostring(v)
7485         if typ == "boolean" then
7486             table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
7487             table.insert(buf, k)
7488             table.insert(buf, "}{")
7489             table.insert(buf, v)
7490             table.insert(buf, "}")
7491         elseif typ == "number" then
7492             table.insert(buf, "\\markdownRendererJekyllDataNumber{")
7493             table.insert(buf, k)
7494             table.insert(buf, "}{")
7495             table.insert(buf, v)
7496             table.insert(buf, "}")
7497         elseif typ == "string" then
7498             table.insert(buf, "\\markdownRendererJekyllDataString{")
7499             table.insert(buf, k)
7500             table.insert(buf, "}{")
7501             table.insert(buf, t(v))
7502             table.insert(buf, "}")
7503         elseif typ == "table" then
7504             table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
7505             table.insert(buf, k)
7506             table.insert(buf, "}")
7507         else
7508             error(format("Unexpected type %s for value of " ..
7509                 "YAML key %s", typ, k))
7510         end
7511     end
7512 end
7513
7514 if #d > 0 then
7515     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
7516 else
7517     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
7518 end
7519
7520 if not p then
7521     table.insert(buf, "\\markdownRendererJekyllDataEnd")
7522 end

```

```

7523
7524     return buf
7525 end
7526 end, extend_reader = function(self)
7527     local parsers = self.parsers
7528     local writer = self.writer
7529
7530     local JekyllData
7531         = Cmt( C((parsers.line - P("---") - P("..."))^0)
7532             , function(s, i, text) -- luacheck: ignore s i
7533                 local data
7534                 local ran_ok, _ = pcall(function()
7535                     local tinyyaml = require("markdown-tinyyaml")
7536                     data = tinyyaml.parse(text, {timestamps=false})
7537                 end)
7538                 if ran_ok and data ~= nil then
7539                     return true, writer.jekyllData(data, function(s)
7540                         return self.parser_functions.parse_blocks_nested(s)
7541                     end, nil)
7542                 else
7543                     return false
7544                 end
7545             end
7546         )
7547
7548     local UnexpectedJekyllData
7549         = P("---")
7550         * parsers.blankline / 0
7551         * #(-parsers.blankline) -- if followed by blank, it's thematic br
7552         * JekyllData
7553         * (P("---") + P("..."))
7554
7555     local ExpectedJekyllData
7556         = ( P("---")
7557             * parsers.blankline / 0
7558             * #(-parsers.blankline) -- if followed by blank, it's thematic
7559             )^-1
7560         * JekyllData
7561         * (P("---") + P("..."))^-1
7562
7563     self.insert_pattern("Block before Blockquote",
7564                         UnexpectedJekyllData, "UnexpectedJekyllData")
7565     if expect_jekyll_data then
7566         self.update_rule("ExpectedJekyllData", function() return ExpectedJekyllData end)
7567     end
7568 end
7569 }

```

7570 end

3.1.7 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.

7571 function M.new(options)

 Make the `options` table inherit from the `defaultOptions` table.

```
7572 options = options or {}
7573 setmetatable(options, { __index = function (_, key)
7574     return defaultOptions[key] end })
```

 Apply built-in syntax extensions based on `options`.

```
7575 local extensions = {}
7576
7577 if options.bracketedSpans then
7578     local bracketed_spans_extension = M.extensions.bracketed_spans()
7579     table.insert(extensions, bracketed_spans_extension)
7580 end
7581
7582 if options.contentBlocks then
7583     local content_blocks_extension = M.extensions.content_blocks(
7584         options.contentBlocksLanguageMap)
7585     table.insert(extensions, content_blocks_extension)
7586 end
7587
7588 if options.definitionLists then
7589     local definition_lists_extension = M.extensions.definition_lists(
7590         options.tightLists)
7591     table.insert(extensions, definition_lists_extension)
7592 end
7593
7594 if options.fencedCode then
7595     local fenced_code_extension = M.extensions.fenced_code(
7596         options.blankBeforeCodeFence)
7597     table.insert(extensions, fenced_code_extension)
7598 end
7599
7600 if options.fencedDivs then
7601     local fenced_div_extension = M.extensions.fenced_divs(
7602         options.blankBeforeDivFence)
7603     table.insert(extensions, fenced_div_extension)
7604 end
7605
7606 if options.headerAttributes then
7607     local header_attributes_extension = M.extensions.header_attributes()
```

```

7608     table.insert(extensions, header_attributes_extension)
7609 end
7610
7611 if options.jekyllData then
7612     local jekyll_data_extension = M.extensions.jekyll_data(
7613         options.expectJekyllData)
7614     table.insert(extensions, jekyll_data_extension)
7615 end
7616
7617 if options.pipeTables then
7618     local pipe_tables_extension = M.extensions.pipe_tables(
7619         options.tableCaptions)
7620     table.insert(extensions, pipe_tables_extension)
7621 end
7622
7623 if options.rawAttribute then
7624     local raw_attribute_extension = M.extensions.raw_attribute()
7625     table.insert(extensions, raw_attribute_extension)
7626 end
7627
7628 if options.strikeThrough then
7629     local strike_through_extension = M.extensions.strike_through()
7630     table.insert(extensions, strike_through_extension)
7631 end
7632
7633 if options.subscripts then
7634     local subscript_extension = M.extensions.subscripts()
7635     table.insert(extensions, subscript_extension)
7636 end
7637
7638 if options.superscripts then
7639     local superscript_extension = M.extensions.superscripts()
7640     table.insert(extensions, superscript_extension)
7641 end
7642

```

The footnotes and inlineFootnotes option has been deprecated and will be removed in Markdown 3.0.0.

```

7643 if options.footnotes or options.inlineFootnotes or
7644     options.notes or options.inlineNotes then
7645     local notes_extension = M.extensions.notes(
7646         options.footnotes or options.notes,
7647         options.inlineFootnotes or options.inlineNotes)
7648     table.insert(extensions, notes_extension)
7649 end
7650
7651 if options.citations then

```

```

7652     local citations_extension = M.extensions.citations(options.citationNbsps)
7653     table.insert(extensions, citations_extension)
7654 end
7655
7656 if options.fancyLists then
7657     local fancy_lists_extension = M.extensions.fancy_lists()
7658     table.insert(extensions, fancy_lists_extension)
7659 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

7660 for _, user_extension_filename in ipairs(options.extensions) do
7661     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

7662         local pathname = util.lookup_files(filename)
7663         local input_file = assert(io.open(pathname, "r"),
7664             [[Could not open user-defined syntax extension "]]
7665             .. pathname .. [[ for reading]])
7666         local input = assert(input_file:read("*a"))
7667         assert(input_file:close())
7668         local user_extension, err = load([[
7669             local sandbox = {}
7670             setmetatable(sandbox, {__index = _G})
7671             _ENV = sandbox
7672         ]] .. input)()
7673         assert(user_extension,
7674             [[Failed to compile user-defined syntax extension "]]
7675             .. pathname .. [[: ]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

7676         assert(user_extension.api_version ~= nil,
7677             [[User-defined syntax extension "]] .. pathname
7678             .. [[ does not specify mandatory field "api_version"]])
7679         assert(type(user_extension.api_version) == "number",
7680             [[User-defined syntax extension "]] .. pathname
7681             .. [[ specifies field "api_version" of type "]]
7682             .. type(user_extension.api_version)
7683             .. [[ but "number" was expected]])
7684         assert(user_extension.api_version > 0
7685             and user_extension.api_version <= metadata.user_extension_api_version,
7686             [[User-defined syntax extension "]] .. pathname
7687             .. [[ uses syntax extension API version "]]
7688             .. user_extension.api_version .. [[ but markdown.lua ]]
7689             .. metadata.version .. [[ uses API version ]]
7690             .. metadata.user_extension_api_version
7691             .. [[, which is incompatible]])
7692
7693         assert(user_extension.grammar_version ~= nil,

```

```

7694     [[User-defined syntax extension "]] .. pathname
7695     .. [[ " does not specify mandatory field "grammar_version" ]])
7696     assert(type(user_extension.grammar_version) == "number",
7697     [[User-defined syntax extension "]] .. pathname
7698     .. [[ " specifies field "grammar_version" of type "]]
7699     .. type(user_extension.grammar_version)
7700     .. [[ " but "number" was expected ]])
7701     assert(user_extension.grammar_version == metadata.grammar_version,
7702     [[User-defined syntax extension "]] .. pathname
7703     .. [[ " uses grammar version "]] .. user_extension.grammar_version
7704     .. [[ but markdown.lua ]] .. metadata.version
7705     .. [[ uses grammar version ]] .. metadata.grammar_version
7706     .. [[ , which is incompatible ]])
7707
7708     assert(user_extension.finalize_grammar ~= nil,
7709     [[User-defined syntax extension "]] .. pathname
7710     .. [[ " does not specify mandatory "finalize_grammar" field ]])
7711     assert(type(user_extension.finalize_grammar) == "function",
7712     [[User-defined syntax extension "]] .. pathname
7713     .. [[ " specifies field "finalize_grammar" of type "]]
7714     .. type(user_extension.finalize_grammar)
7715     .. [[ " 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 3.1.6.)

```

7716     local extension = {
7717         name = [[user-defined "]] .. pathname .. [[ " syntax extension]],
7718         extend_reader = user_extension.finalize_grammar,
7719         extend_writer = function() end,
7720     }
7721     return extension
7722     end)(user_extension_filename)
7723     table.insert(extensions, user_extension)
7724 end

```

Produce and return a conversion function from markdown to plain \TeX .

```

7725     local writer = M.writer.new(options)
7726     local reader = M.reader.new(writer, options)
7727     local convert = reader.finalize_grammar(extensions)
7728
7729     return convert
7730 end
7731
7732 return M

```


3.1.8 Command-Line Implementation

The command-line implementation provides the actual conversion routine for the command-line interface described in Section 2.1.6.

```
7733
7734 local input
7735 if input_filename then
7736   local input_file = assert(io.open(input_filename, "r"),
7737     [[Could not open file ]] .. input_filename .. [[ for reading]])
7738   input = assert(input_file:read("*a"))
7739   assert(input_file:close())
7740 else
7741   input = assert(io.read("*a"))
7742 end
7743
```

First, ensure that the `options.cacheDir` directory exists.

```
7744 local lfs = require("lfs")
7745 if options.cacheDir and not lfs.isdir(options.cacheDir) then
7746   assert(lfs.mkdir(options["cacheDir"]))
7747 end
7748
7749 local ran_ok, kpse = pcall(require, "kpse")
7750 if ran_ok then kpse.set_program_name("luatex") end
7751 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.

```
7752 if metadata.version ~= md.metadata.version then
7753   warn("markdown-cli.lua " .. metadata.version .. " used with " ..
7754     "markdown.lua " .. md.metadata.version .. ".")
7755 end
7756 local convert = md.new(options)
```

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.

```
7757 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
7758
7759 if output_filename then
7760   local output_file = assert(io.open(output_filename, "w"),
7761     [[Could not open file ]] .. output_filename .. [[ for writing]])
7762   assert(output_file:write(output))
7763   assert(output_file:close())
7764 else
7765   assert(io.write(output))
7766 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

```
7767 \ifx\markdownInfo\undefined
7768   \def\markdownInfo#1{%
7769     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
7770 \fi
7771 \ifx\markdownWarning\undefined
7772   \def\markdownWarning#1{%
7773     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
7774 \fi
7775 \ifx\markdownError\undefined
7776   \def\markdownError#1#2{%
7777     \errhelp{#2.}%
7778     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
7779 \fi
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
7780 \def\markdownRendererInterblockSeparatorPrototype{\par}%
7781 \def\markdownRendererLineBreakPrototype{\hfil\break}%
7782 \let\markdownRendererEllipsisPrototype\dots
7783 \def\markdownRendererNbspPrototype{~}%
7784 \def\markdownRendererLeftBracePrototype{\char`{}%
7785 \def\markdownRendererRightBracePrototype{\char`}%
7786 \def\markdownRendererDollarSignPrototype{\char`$}%
7787 \def\markdownRendererPercentSignPrototype{\char`}%
7788 \def\markdownRendererAmpersandPrototype{&}%
7789 \def\markdownRendererUnderscorePrototype{\char`_%
7790 \def\markdownRendererHashPrototype{\char`#}%
7791 \def\markdownRendererCircumflexPrototype{\char`^}%
7792 \def\markdownRendererBackslashPrototype{\char`\}%
7793 \def\markdownRendererTildePrototype{\char`~}%
7794 \def\markdownRendererPipePrototype{|}%
7795 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
7796 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
7797 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7798   \markdownInput{#3}}%
7799 \def\markdownRendererContentBlockOnlineImagePrototype{%
7800   \markdownRendererImage}%
```

```

7801 \def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
7802   \markdownRendererInputFencedCode{#3}{#2}}%
7803 \def\markdownRendererImagePrototype#1#2#3#4{#2}%
7804 \def\markdownRendererUlBeginPrototype{}%
7805 \def\markdownRendererUlBeginTightPrototype{}%
7806 \def\markdownRendererUlItemPrototype{}%
7807 \def\markdownRendererUlItemEndPrototype{}%
7808 \def\markdownRendererUlEndPrototype{}%
7809 \def\markdownRendererUlEndTightPrototype{}%
7810 \def\markdownRendererOlBeginPrototype{}%
7811 \def\markdownRendererOlBeginTightPrototype{}%
7812 \def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
7813 \def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
7814 \def\markdownRendererOlItemPrototype{}%
7815 \def\markdownRendererOlItemWithNumberPrototype#1{}%
7816 \def\markdownRendererOlItemEndPrototype{}%
7817 \def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
7818 \def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}%
7819 \def\markdownRendererFancyOlItemEndPrototype{}%
7820 \def\markdownRendererOlEndPrototype{}%
7821 \def\markdownRendererOlEndTightPrototype{}%
7822 \def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
7823 \def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
7824 \def\markdownRendererDlBeginPrototype{}%
7825 \def\markdownRendererDlBeginTightPrototype{}%
7826 \def\markdownRendererDlItemPrototype#1{#1}%
7827 \def\markdownRendererDlItemEndPrototype{}%
7828 \def\markdownRendererDlDefinitionBeginPrototype{}%
7829 \def\markdownRendererDlDefinitionEndPrototype{\par}%
7830 \def\markdownRendererDlEndPrototype{}%
7831 \def\markdownRendererDlEndTightPrototype{}%
7832 \def\markdownRendererEmphasisPrototype#1{\it#1}%
7833 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
7834 \def\markdownRendererBlockQuoteBeginPrototype{\par\begingroup\it}%
7835 \def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
7836 \def\markdownRendererInputVerbatimPrototype#1{%
7837   \par{\tt\input#1\relax{}}\par}%
7838 \def\markdownRendererInputFencedCodePrototype#1#2{%
7839   \markdownRendererInputVerbatimPrototype{#1}}%
7840 \def\markdownRendererHeadingOnePrototype#1{#1}%
7841 \def\markdownRendererHeadingTwoPrototype#1{#1}%
7842 \def\markdownRendererHeadingThreePrototype#1{#1}%
7843 \def\markdownRendererHeadingFourPrototype#1{#1}%
7844 \def\markdownRendererHeadingFivePrototype#1{#1}%
7845 \def\markdownRendererHeadingSixPrototype#1{#1}%
7846 \def\markdownRendererThematicBreakPrototype{}%
7847 \def\markdownRendererNotePrototype#1{#1}%

```

```

7848 \def\markdownRendererCitePrototype#1{}%
7849 \def\markdownRendererTextCitePrototype#1{}%
7850 \def\markdownRendererTickedBoxPrototype{[X]}%
7851 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
7852 \def\markdownRendererUntickedBoxPrototype{[ ]}%
7853 \def\markdownRendererStrikeThroughPrototype#1{#1}%
7854 \def\markdownRendererSuperscriptPrototype#1{#1}%
7855 \def\markdownRendererSubscriptPrototype#1{#1}%

```

3.2.2.1 Raw Attribute Renderer Prototypes 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.

```

7856 \ExplSyntaxOn
7857 \cs_gset:Npn
7858   \markdownRendererInputRawInlinePrototype#1#2
7859   {
7860     \str_case:nn
7861       { #2 }
7862       {
7863         { tex } { \markdownEscape{#1} }
7864         { md } { \markdownInput{#1} }
7865       }
7866   }
7867 \cs_gset_eq:NN
7868   \markdownRendererInputRawBlockPrototype
7869   \markdownRendererInputRawInlinePrototype
7870 \ExplSyntaxOff

```

3.2.2.2 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_t1` The currently traversed branch of the YAML document contains a mapping at depth p .

`\c_@@_jekyll_data_scalar_t1` The currently traversed branch of the YAML document contains a scalar value at depth p .

```

7871 \ExplSyntaxOn
7872 \seq_new:N \g_@@_jekyll_data_datatypes_seq

```

```

7873 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
7874 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
7875 \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.

```

7876 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
7877 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
7878 {
7879   \seq_if_empty:NF
7880     \g_@@_jekyll_data_datatypes_seq
7881     {
7882       \seq_get_right:NN
7883       \g_@@_jekyll_data_datatypes_seq
7884       \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.

```

7885   \str_if_eq:NNTF
7886     \l_tmpa_tl
7887     \c_@@_jekyll_data_sequence_tl
7888     {
7889       \seq_put_right:Nn
7890         \g_@@_jekyll_data_wildcard_absolute_address_seq
7891         { * }
7892     }
7893     {
7894       \seq_put_right:Nn
7895         \g_@@_jekyll_data_wildcard_absolute_address_seq
7896         { #1 }
7897     }
7898   }
7899 }

```

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.

```
7900 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
7901 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
7902 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
7903   {
7904     \seq_pop_left:NN #1 \l_tmpa_tl
7905     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
7906     \seq_put_left:NV #1 \l_tmpa_tl
7907   }
7908 \cs_new:Nn \markdown_jekyll_data_update_address_tls:
7909   {
7910     \markdown_jekyll_data_concatenate_address:NN
7911     \g_@@_jekyll_data_wildcard_absolute_address_seq
7912     \g_@@_jekyll_data_wildcard_absolute_address_tl
7913     \seq_get_right:NN
7914     \g_@@_jekyll_data_wildcard_absolute_address_seq
7915     \g_@@_jekyll_data_wildcard_relative_address_tl
7916   }
```

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.

```
7917 \cs_new:Nn \markdown_jekyll_data_push:nN
7918   {
7919     \markdown_jekyll_data_push_address_segment:n
7920     { #1 }
7921     \seq_put_right:NV
7922     \g_@@_jekyll_data_datatypes_seq
7923     #2
7924     \markdown_jekyll_data_update_address_tls:
7925   }
7926 \cs_new:Nn \markdown_jekyll_data_pop:
```

```

7927 {
7928   \seq_pop_right:NN
7929   \g_@@_jekyll_data_wildcard_absolute_address_seq
7930   \l_tmpa_tl
7931   \seq_pop_right:NN
7932   \g_@@_jekyll_data_datatypes_seq
7933   \l_tmpa_tl
7934   \markdown_jekyll_data_update_address_tls:
7935 }

```

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.

```

7936 \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
7937 {
7938   \keys_set_known:nn
7939   { markdown/jekyllData }
7940   { { #1 } = { #2 } }
7941 }
7942 \cs_generate_variant:Nn
7943   \markdown_jekyll_data_set_keyval:nn
7944   { Vn }
7945 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
7946 {
7947   \markdown_jekyll_data_push:nN
7948   { #1 }
7949   \c_@@_jekyll_data_scalar_tl
7950   \markdown_jekyll_data_set_keyval:Vn
7951   \g_@@_jekyll_data_wildcard_absolute_address_tl
7952   { #2 }
7953   \markdown_jekyll_data_set_keyval:Vn
7954   \g_@@_jekyll_data_wildcard_relative_address_tl
7955   { #2 }
7956   \markdown_jekyll_data_pop:
7957 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

7958 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
7959   \markdown_jekyll_data_push:nN
7960   { #1 }
7961   \c_@@_jekyll_data_sequence_tl
7962 }
7963 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
7964   \markdown_jekyll_data_push:nN
7965   { #1 }
7966   \c_@@_jekyll_data_mapping_tl
7967 }

```

```

7968 \def\markdownRendererJekyllDataSequenceEndPrototype{
7969   \markdown_jekyll_data_pop:
7970 }
7971 \def\markdownRendererJekyllDataMappingEndPrototype{
7972   \markdown_jekyll_data_pop:
7973 }
7974 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
7975   \markdown_jekyll_data_set_keyvals:nn
7976   { #1 }
7977   { #2 }
7978 }
7979 \def\markdownRendererJekyllDataEmptyPrototype#1{}
7980 \def\markdownRendererJekyllDataNumberPrototype#1#2{
7981   \markdown_jekyll_data_set_keyvals:nn
7982   { #1 }
7983   { #2 }
7984 }
7985 \def\markdownRendererJekyllDataStringPrototype#1#2{
7986   \markdown_jekyll_data_set_keyvals:nn
7987   { #1 }
7988   { #2 }
7989 }
7990 \ExplSyntaxOff

```

3.2.3 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).

```

7991 \ExplSyntaxOn
7992 \tl_new:N \g_@@_formatted_lua_options_tl
7993 \cs_new:Nn \@@_format_lua_options:
7994   {
7995     \tl_gclear:N
7996     \g_@@_formatted_lua_options_tl
7997     \seq_map_function:NN
7998     \g_@@_lua_options_seq
7999     \@@_format_lua_option:n
8000   }
8001 \cs_new:Nn \@@_format_lua_option:n
8002   {
8003     \@@_typecheck_option:n
8004     { #1 }
8005     \@@_get_option_type:nN
8006     { #1 }
8007     \l_tmpa_tl

```



```

8008 \bool_case_true:nF
8009 {
8010 {
8011   \str_if_eq_p:VV
8012   \l_tmpa_tl
8013   \c_@@_option_type_boolean_tl ||
8014   \str_if_eq_p:VV
8015   \l_tmpa_tl
8016   \c_@@_option_type_number_tl ||
8017   \str_if_eq_p:VV
8018   \l_tmpa_tl
8019   \c_@@_option_type_counter_tl
8020 }
8021 {
8022   \@@_get_option_value:nN
8023   { #1 }
8024   \l_tmpa_tl
8025   \tl_gput_right:Nx
8026   \g_@@_formatted_lua_options_tl
8027   { #1~::~ \l_tmpa_tl ,~ }
8028 }
8029 {
8030   \str_if_eq_p:VV
8031   \l_tmpa_tl
8032   \c_@@_option_type_clist_tl
8033 }
8034 {
8035   \@@_get_option_value:nN
8036   { #1 }
8037   \l_tmpa_tl
8038   \tl_gput_right:Nx
8039   \g_@@_formatted_lua_options_tl
8040   { #1~::~\c_left_brace_str }
8041   \clist_map_inline:Vn
8042   \l_tmpa_tl
8043   {
8044     \tl_gput_right:Nx
8045     \g_@@_formatted_lua_options_tl
8046     { "##1" ,~ }
8047   }
8048   \tl_gput_right:Nx
8049   \g_@@_formatted_lua_options_tl
8050   { \c_right_brace_str ,~ }
8051 }
8052 }
8053 {
8054   \@@_get_option_value:nN

```

```

8055         { #1 }
8056         \l_tmpa_tl
8057     \tl_gput_right:Nx
8058         \g_@@_formatted_lua_options_tl
8059     { #1~::~ " \l_tmpa_tl " ,~ }
8060     }
8061 }
8062 \cs_generate_variant:Nn
8063     \clist_map_inline:nn
8064     { Vn }
8065 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
8066 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
8067 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain T_EX. It exposes the `convert` function for the use by any further Lua code.

```

8068 \def\markdownPrepare{%
First, ensure that the cacheDir directory exists.
8069     local lfs = require("lfs")
8070     local cacheDir = "\markdownOptionCacheDir"
8071     if not lfs.isdir(cacheDir) then
8072         assert(lfs.mkdir(cacheDir))
8073     end

```

Next, load the `markdown` module and create a converter function using the plain T_EX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```

8074     local md = require("markdown")
8075     local convert = md.new(\markdownLuaOptions)
8076 }%

```

3.2.4 Buffering Markdown Input

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.

```

8077 \ExplSyntaxOn
8078 \cs_new:Nn
8079     \@@_if_option:nTF
8080     {
8081         \@@_get_option_type:nN
8082         { #1 }
8083         \l_tmpa_tl
8084     \str_if_eq:NNF
8085         \l_tmpa_tl
8086         \c_@@_option_type_boolean_tl

```

```

8087     {
8088         \msg_error:nnxx
8089         { @@ }
8090         { expected-boolean-option }
8091         { #1 }
8092         { \l_tmpa_tl }
8093     }
8094     \@@_get_option_value:nN
8095     { #1 }
8096     \l_tmpa_tl
8097     \str_if_eq:NNTF
8098     \l_tmpa_tl
8099     \c_@@_option_value_true_tl
8100     { #2 }
8101     { #3 }
8102 }
8103 \msg_new:nnn
8104 { @@ }
8105 { expected-boolean-option }
8106 {
8107     Option~#1~has~type~#2,~
8108     but~a~boolean~was~expected.
8109 }
8110 \let\markdownIfOption=\@@_if_option:nTF
8111 \ExplSyntaxOff

```

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.

```

8112 \csname newread\endcsname\markdownInputFileStream
8113 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

8114 \begingroup
8115   \catcode\^^I=12%
8116   \gdef\markdownReadAndConvertTab{^^I}%
8117 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the $\text{\LaTeX} 2_{\epsilon}$ `\filecontents` macro to plain \TeX .

```

8118 \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.

```

8119   \catcode\^^M=13%

```

```

8120 \catcode\^^I=13%
8121 \catcode|=0%
8122 \catcode\|=12%
8123 |catcode@=14%
8124 |catcode|=12@
8125 |gdef|markdownReadAndConvert#1#2{@
8126   |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `inputTempFileName` file for writing.

```

8127   |markdownIfOption{frozenCache}{}@
8128     |immediate|openout|markdownOutputFileStream@
8129     |markdownOptionInputTempFileName|relax@
8130     |markdownInfo{Buffering markdown input into the temporary @
8131       input file "|markdownOptionInputTempFileName" and scanning @
8132       for the closing token sequence "#1"}@
8133   }@

```

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.

```

8134   |def|do##1{|catcode`##1=12}|dospecials@
8135   |catcode`|=12@
8136   |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `stripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```

8137   |def|markdownReadAndConvertStripPercentSign##1{@
8138     |markdownIfOption{stripPercentSigns}{@
8139       |if##1%@
8140         |expandafter|expandafter|expandafter@
8141         |markdownReadAndConvertProcessLine@
8142       |else@
8143         |expandafter|expandafter|expandafter@
8144         |markdownReadAndConvertProcessLine@
8145         |expandafter|expandafter|expandafter##1@
8146       |fi@
8147     }{@
8148     |expandafter@
8149     |markdownReadAndConvertProcessLine@
8150     |expandafter##1@
8151   }@
8152 }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (`@`) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (`^^M`) are produced.

```
8153      |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@
```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `inputTempFileName` file. If we are reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, gobble the line.

```
8154      |ifx|relax##3|relax@
8155          |markdownIfOption{frozenCache}{-}{@
8156              |immediate|write|markdownOutputFileStream{##1}@
8157          }@
8158      |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.

```
8159      |def^^M{@
8160          |markdownInfo{The ending token sequence was found}@
8161          |markdownIfOption{frozenCache}{-}{@
8162              |immediate|closeout|markdownOutputFileStream@
8163          }@
8164          |endgroup@
8165          |markdownInput{@
8166              |markdownOptionOutputDir@
8167              /|markdownOptionInputTempFileName@
8168          }@
8169          #2}@
8170      |fi@
```

Repeat with the next line.

```
8171      ^^M}@
```

Make the tab character active at expansion time and make it expand to a literal tab character.

```
8172      |catcode`|^~I=13@
8173      |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.

```
8174      |catcode`|^~M=13@
8175      |def^^M##1^^M{@
8176          |def^^M####1^^M{@
8177              |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
```

```
8178     ^^M}@
8179     ^^M}@
```

Reset the character categories back to the former state.

```
8180 |endgroup
```

The following two sections of the implementation have been deprecated and will be removed in Markdown 3.0.0. The code that corresponds to `\markdownMode` value of `3` will be the only implementation.

```
8181 \ExplSyntaxOn
8182 \int_compare:nT
8183   { \markdownMode = 3 }
8184   {
8185     \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
8186     \file_input:n { lt3luabridge.tex }
8187     \cs_new:Npn
8188       \markdownLuaExecute
8189       { \luabridgeExecute }
8190   }
8191 \ExplSyntaxOff
```

3.2.5 Lua Shell Escape Bridge

The following \TeX code is intended for \TeX engines that do not provide direct access to Lua, but expose the shell of the operating system. This corresponds to the `\markdownMode` values of `0` and `1`.

The `\markdownLuaExecute` macro defined here and in Section 3.2.6 are meant to be indistinguishable to the remaining code.

The package assumes that although the user is not using the Lua \TeX engine, their \TeX distribution contains it, and uses shell access to produce and execute Lua scripts using the \TeX Lua interpreter [1, Section 4.1.1].

```
8192 \ifnum\markdownMode<2\relax
8193 \ifnum\markdownMode=0\relax
8194   \markdownWarning{Using mode 0: Shell escape via write18
8195                   (deprecated, to be removed in Markdown 3.0.0)}%
8196 \else
8197   \markdownWarning{Using mode 1: Shell escape via os.execute
8198                   (deprecated, to be removed in Markdown 3.0.0)}%
8199 \fi
```

The `\markdownExecuteShellEscape` macro contains the numeric value indicating whether the shell access is enabled (`1`), disabled (`0`), or restricted (`2`).

Inherit the value of the the `\pdfshellescape` (Lua \TeX , Pdf \TeX) or the `\shellescape` (X \TeX) commands. If neither of these commands is defined and Lua is available, attempt to access the `status.shell_escape` configuration item.

If you cannot detect, whether the shell access is enabled, act as if it were.

```

8200 \ifx\pdfshellescape\undefined
8201   \ifx\shellescape\undefined
8202     \ifnum\markdownMode=0\relax
8203       \def\markdownExecuteShellEscape{1}%
8204     \else
8205       \def\markdownExecuteShellEscape{%
8206         \directlua{tex.sprint(status.shell_escape or "1")}}%
8207     \fi
8208   \else
8209     \let\markdownExecuteShellEscape\shellescape
8210   \fi
8211 \else
8212   \let\markdownExecuteShellEscape\pdfshellescape
8213 \fi

```

The `\markdownExecuteDirect` macro executes the code it has received as its first argument by writing it to the output file stream 18, if Lua is unavailable, or by using the Lua `os.execute` method otherwise.

```

8214 \ifnum\markdownMode=0\relax
8215   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
8216 \else
8217   \def\markdownExecuteDirect#1{%
8218     \directlua{os.execute("\luaescapestring{#1}")}}%
8219 \fi

```

The `\markdownExecute` macro is a wrapper on top of `\markdownExecuteDirect` that checks the value of `\markdownExecuteShellEscape` and prints an error message if the shell is inaccessible.

```

8220 \def\markdownExecute#1{%
8221   \ifnum\markdownExecuteShellEscape=1\relax
8222     \markdownExecuteDirect{#1}%
8223   \else
8224     \markdownError{I can not access the shell}{Either run the TeX
8225       compiler with the --shell-escape or the --enable-write18 flag,
8226       or set shell_escape=t in the texmf.cnf file}%
8227   \fi}%

```

The `\markdownLuaExecute` macro executes the Lua code it has received as its first argument. The Lua code may not directly interact with the TeX engine, but it can use the `print` function in the same manner it would use the `tex.print` method.

```

8228 \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.

```

8229   \catcode`\|=0%
8230   \catcode`\|=12%
8231   |gdef|markdownLuaExecute#1{%

```

Create the file `helperScriptFileName` and fill it with the input Lua code prepended with `kpathsea` initialization, so that Lua modules from the `TEX` distribution are available.

```

8232 |immediate|openout|markdownOutputFileStream=%
8233 |markdownOptionHelperScriptFileName
8234 |markdownInfo{Writing a helper Lua script to the file
8235 "|markdownOptionHelperScriptFileName"}%
8236 |immediate|write|markdownOutputFileStream{%
8237 local ran_ok, error = pcall(function()
8238 local ran_ok, kpse = pcall(require, "kpse")
8239 if ran_ok then kpse.set_program_name("luatex") end
8240 #1
8241 end)

```

If there was an error, use the file `errorTempFileName` to store the error message.

```

8242 if not ran_ok then
8243 local file = io.open("%
8244 |markdownOptionOutputDir
8245 /|markdownOptionErrorTempFileName", "w")
8246 if file then
8247 file:write(error .. "\n")
8248 file:close()
8249 end
8250 print('\|markdownError{An error was encountered while executing
8251 Lua code}{For further clues, examine the file
8252 "|markdownOptionOutputDir
8253 /|markdownOptionErrorTempFileName}')
8254 end}%
8255 |immediate|closeout|markdownOutputFileStream

```

Execute the generated `helperScriptFileName` Lua script using the `TEX Lua` binary and store the output in the `outputTempFileName` file.

```

8256 |markdownInfo{Executing a helper Lua script from the file
8257 "|markdownOptionHelperScriptFileName" and storing the result in the
8258 file "|markdownOptionOutputTempFileName"}%
8259 |markdownExecute{texlua "|markdownOptionOutputDir
8260 /|markdownOptionHelperScriptFileName" > %
8261 "|markdownOptionOutputDir
8262 /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `outputTempFileName` file.

```

8263 |input|markdownOptionOutputTempFileName|relax}%
8264 |endgroup

```

3.2.6 Direct Lua Access

The following `TEX` code is intended for `TEX` engines that provide direct access to Lua (Lua`TEX`). The macro `\markdownLuaExecute` defined here and in Section 3.2.5

are meant to be indistinguishable to the remaining code. This corresponds to the `\markdownMode` value of 2.

```
8265 \fi
8266 \ifnum\markdownMode=2\relax
8267   \markdownWarning{Using mode 2: Direct Lua access
8268     (deprecated, to be removed in Markdown 3.0.0)}%
```

The direct Lua access version of the `\markdownLuaExecute` macro is defined in terms of the `\directlua` primitive. The `print` function is set as an alias to the `tex.print` method in order to mimic the behaviour of the `\markdownLuaExecute` definition from Section 3.2.5,

```
8269 \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.

```
8270   \catcode`\|=0%
8271   \catcode`\|=12%
8272   |gdef|markdownLuaExecute#1{%
8273     |directlua{%
8274       local function print(input)
8275         local output = {}
8276         for line in input:gmatch("[^\r\n]+") do
8277           table.insert(output, line)
8278         end
8279         tex.print(output)
8280       end
8281       #1
8282     }%
8283   }%
8284 |endgroup
8285 \fi
```

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.

```
8286 \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.

```
8287   \catcode`\|=0%
8288   \catcode`\|=12%
8289   \catcode`\&=6%
8290   |gdef|markdownInput#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.

```
8291     |begingroup
8292     |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.

```
8293     |catcode`|#=12
```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache<number>` macro, and increment `frozenCacheCounter`.

```
8294     |markdownIfOption{frozenCache}{%
8295         |ifnum|markdownOptionFrozenCacheCounter=0|relax
8296         |markdownInfo{Reading frozen cache from
8297             "|markdownOptionFrozenCacheFileName"}%
8298         |input|markdownOptionFrozenCacheFileName|relax
8299         |fi
8300         |markdownInfo{Including markdown document number
8301             "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
8302         |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
8303         |global|advance|markdownOptionFrozenCacheCounter by 1|relax
8304     }{%
8305         |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.

```
8306     |openin|markdownInputFileStream&1
8307     |closein|markdownInputFileStream
8308     |markdownPrepareLuaOptions
8309     |markdownLuaExecute{%
8310         |markdownPrepare
8311         local file = assert(io.open("&1", "r"),
8312             [[Could not open file "&1" for reading]])
8313         local input = assert(file:read("*a"))
8314         assert(file:close())
```

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.

```
8315         print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%
```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```
8316     |global|advance|markdownOptionFrozenCacheCounter by 1|relax
8317     }%
8318     |endgroup
8319 }
```

```
8320 |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.

```
8321 \gdef\markdownEscape#1{%
8322   \catcode`\%=14\relax
8323   \catcode`\#=6\relax
8324   \input #1\relax
8325   \catcode`\%=12\relax
8326   \catcode`\#=12\relax
8327 }%
```

3.3 L^AT_EX Implementation

The L^AT_EX implemenation makes use of the fact that, apart from some subtle differences, L^AT_EX implements the majority of the plain `TEX` format [12, Section 9]. As a consequence, we can directly reuse the existing plain `TEX` implementation.

```
8328 \def\markdownVersionSpace{ }%
8329 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
8330   \markdownVersion\markdownVersionSpace markdown renderer]%
```

Use reflection to define the `renderers` and `rendererPrototypes` keys of `\markdownSetup` as well as the keys that correspond to Lua options.

```
8331 \ExplSyntaxOn
8332 \@@_latex_define_renderers:
8333 \@@_latex_define_renderer_prototypes:
8334 \ExplSyntaxOff
```

3.3.1 Logging Facilities

The L^AT_EX implementation redefines the plain `TEX` logging macros (see Section 3.2.1) to use the L^AT_EX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

3.3.2 Typesetting Markdown

The `\markdownInputPlainTeX` macro is used to store the original plain `TEX` implementation of the `\markdownInput` macro. The `\markdownInput` is then redefined to accept an optional argument with options recognized by the L^AT_EX interface (see Section 2.3.2).

```
8335 \let\markdownInputPlainTeX\markdownInput
8336 \renewcommand\markdownInput[2] []{%
8337   \begingroup
8338     \markdownSetup{#1}%
```

```

8339     \markdownInputPlainTeX{#2}%
8340 \endgroup}%

```

The `markdown`, and `markdown*` L^AT_EX environments are implemented using the `\markdownReadAndConvert` macro.

```

8341 \renewenvironment{markdown}{%
8342   \markdownReadAndConvert@markdown{}}{%
8343   \markdownEnd}%
8344 \renewenvironment{markdown*}[1]{%
8345   \markdownSetup{#1}%
8346   \markdownReadAndConvert@markdown*}%
8347   \markdownEnd}%
8348 \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*.

```

8349   \catcode`\|=0\catcode`\<=1\catcode`\>=2%
8350   \catcode`\|=12\catcode`\{=12\catcode`\}=12%
8351   |gdef|markdownReadAndConvert@markdown#1<%
8352     |markdownReadAndConvert<\end{markdown#1}>%
8353     <|end<markdown#1>>%
8354 |endgroup

```

3.3.2.1 L^AT_EX Themes This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```

8355 \ExplSyntaxOn

```

To keep track of our current place when packages themes have been nested, we will maintain the `\g_@@_latex_themes_seq` stack of theme names.

```

8356 \newcommand\markdownLaTeXThemeName{}
8357 \seq_new:N \g_@@_latex_themes_seq
8358 \seq_gput_right:NV
8359   \g_@@_latex_themes_seq
8360   \markdownLaTeXThemeName
8361 \newcommand\markdownLaTeXThemeLoad[2]{
8362   \def\@tempa{%
8363     \def\markdownLaTeXThemeName{#2}
8364     \seq_gput_right:NV
8365       \g_@@_latex_themes_seq
8366       \markdownLaTeXThemeName
8367     \RequirePackage{#1}
8368     \seq_pop_right:NN
8369       \g_@@_latex_themes_seq
8370     \l_tmpa_tl
8371     \seq_get_right:NN

```

```

8372     \g_@@_latex_themes_seq
8373     \l_tmpa_tl
8374     \exp_args:NNV
8375     \def
8376     \markdownLaTeXThemeName
8377     \l_tmpa_tl}
8378 \ifmarkdownLaTeXLoaded
8379   \@tempa
8380 \else
8381   \exp_args:No
8382   \AtEndOfPackage
8383   { \@tempa }
8384 \fi}
8385 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```
8386 \markdownSetup{fencedCode}%
```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```
8387 \RequirePackage{ifthen,grffile}
```

We store the previous definition of the fenced code token renderer prototype:

```
8388 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
8389 \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:

```

8390 \renewcommand\markdownRendererInputFencedCode[2]{%
8391   \def\next##1 ##2\relax{%
8392     \ifthenelse{\equal{##1}{dot}}{%
8393       \markdownIfOption{frozenCache}{-}{%
8394         \immediate\write18{%
8395           if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
8396           then
8397             dot -Tpdf -o #1.pdf #1;
8398             cp #1 #1.pdf.source;
8399             fi}}%

```

We include the typeset image using the image token renderer:

```
8400   \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:

```

8401   }{%
8402     \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
8403   }%
8404 }%

```

```
8405 \next#2 \relax}%
```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```
8406 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
8407 \markdownRendererImagePrototype
```

We load the catchfile and grffile packages, see also Section 1.1.3:

```
8408 \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.

```
8409 \newcount\markdown@witiko@graphicx@http@counter
8410 \markdown@witiko@graphicx@http@counter=0
8411 \newcommand\markdown@witiko@graphicx@http@filename{%
8412   \markdownOptionCacheDir/witiko_graphicx_http%
8413   .\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.

```
8414 \newcommand\markdown@witiko@graphicx@http@download[2]{%
8415   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:

```
8416 \begingroup
8417 \catcode`\%=12
8418 \catcode`\^^A=14
```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
8419 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
8420   \begingroup
8421     \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:

```
8422   \markdownIfOption{frozenCache}{}{^^A
8423     \immediate\write18{^^A
8424       mkdir -p "\markdownOptionCacheDir";
8425       if printf '%s' "#3" | grep -q -E '^https?:';
8426       then
```

The image will be downloaded to the pathname `cacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```

8427     OUTPUT_PREFIX="\markdownOptionCacheDir";
8428     OUTPUT_BODY="\$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
8429     OUTPUT_SUFFIX="\$(printf '%s' '#3' | sed 's/.*[.]//')";
8430     OUTPUT="\$OUTPUT_PREFIX/\$OUTPUT_BODY.\$OUTPUT_SUFFIX";

```

The image will be downloaded only if it has not already been downloaded:

```

8431     if ! [ -e "$OUTPUT" ];
8432     then
8433         \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
8434         printf '%s' "$OUTPUT" > "\filename";
8435     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:

```

8436     else
8437         printf '%s' '#3' > "\filename";
8438     fi}}^^A

```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```

8439     \CatchFileDef{\filename}{\filename}{\newlinechar=-1}^^A
8440     \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
8441     {#1}{#2}{\filename}{#4}^^A
8442 \endgroup
8443 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
8444 \endgroup

```

The [witiko/tilde](#) theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```

8445 \renewcommand\markdownRendererTildePrototype{~}%

```

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```

8446 \DeclareOption*{%
8447     \expandafter\markdownSetup\expandafter{\CurrentOption}}%
8448 \ProcessOptions\relax

```

After processing the options, activate the `jeekyllDataRenderes`, `renderers`, `rendererPrototypes`, and `code` keys.

```

8449 \ExplSyntaxOn
8450 \keys_define:nn
8451 { markdown/latex-options }
8452 {
8453     renderers .code:n = {
8454         \keys_set:nn
8455         { markdown/latex-options/renderers }
8456         { #1 }

```

```

8457     },
8458   }
8459 \@@_with_various_cases:nn
8460 { rendererPrototypes }
8461 {
8462   \keys_define:nn
8463     { markdown/latex-options }
8464     {
8465       #1 .code:n = {
8466         \keys_set:nn
8467           { markdown/latex-options/renderer-prototypes }
8468           { ##1 }
8469       },
8470     }
8471 }

```

The `code` key is used to immediately expand and execute code, which can be especially useful in L^AT_EX setup snippets.

```

8472 \keys_define:nn
8473 { markdown/latex-options }
8474 {
8475   code .code:n = { #1 },
8476 }

```

The `jekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```

8477 \@@_with_various_cases:nn
8478 { jekyllDataRenderers }
8479 {
8480   \keys_define:nn
8481     { markdown/latex-options }
8482     {
8483       #1 .code:n = {
8484         \tl_set:Nn
8485           \l_tmpa_tl
8486           { ##1 }

```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the `nput` 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.

```

8487         \tl_replace_all:NnV
8488           \l_tmpa_tl
8489           { / }
8490         \c_backslash_str

```



```

8491         \keys_set:nV
8492         { markdown/latex-options/jekyll-data-renderers }
8493         \l_tmpa_tl
8494     },
8495 }
8496 }
8497 \keys_define:nn
8498 { markdown/latex-options/jekyll-data-renderers }
8499 {
8500     unknown .code:n = {
8501         \tl_set_eq:NN
8502         \l_tmpa_tl
8503         \l_keys_key_str
8504         \tl_replace_all:NVn
8505         \l_tmpa_tl
8506         \c_backslash_str
8507         { / }
8508         \tl_put_right:Nn
8509         \l_tmpa_tl
8510         {
8511             .code:n = { #1 }
8512         }
8513         \keys_define:nV
8514         { markdown/jekyllData }
8515         \l_tmpa_tl
8516     }
8517 }
8518 \cs_generate_variant:Nn
8519 \keys_define:nn
8520 { nV }
8521 \cs_generate_variant:Nn
8522 \tl_replace_all:Nnn
8523 { NVn }
8524 \cs_generate_variant:Nn
8525 \tl_replace_all:Nnn
8526 { NnV }
8527 \ExplSyntaxOff

```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.1), none of it will take effect.

```
8528 \markdownIfOption{plain}{\iffalse}{\iftrue}
```

If the `tightLists` Lua option is disabled or the current document class is `beamer`, do not load the `paralist` package.

```
8529 \markdownIfOption{tightLists}{
```

```

8530 \ifclassloaded{beamer}{\RequirePackage{paralist}}%
8531 }-{}

```

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.

```

8532 \ExplSyntaxOn
8533 \ifpackageloaded{paralist}{
8534   \tl_new:N
8535     \l_@@_latex_fancy_list_item_label_number_style_tl
8536   \tl_new:N
8537     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8538   \cs_new:Nn
8539     \@@_latex_fancy_list_item_label_number:nn
8540     {
8541       \str_case:nn
8542         { #1 }
8543         {
8544           { Decimal } { #2 }
8545           { LowerRoman } { \int_to_roman:n { #2 } }
8546           { UpperRoman } { \int_to_Roman:n { #2 } }
8547           { LowerAlpha } { \int_to_alph:n { #2 } }
8548           { UpperAlpha } { \int_to_alph:n { #2 } }
8549         }
8550     }
8551   \cs_new:Nn
8552     \@@_latex_fancy_list_item_label_delimiter:n
8553     {
8554       \str_case:nn
8555         { #1 }
8556         {
8557           { Default } { . }
8558           { OneParen } { ) }
8559           { Period } { . }
8560         }
8561     }
8562   \cs_new:Nn
8563     \@@_latex_fancy_list_item_label:nnn
8564     {
8565       \@@_latex_fancy_list_item_label_number:nn
8566         { #1 }
8567         { #3 }
8568       \@@_latex_fancy_list_item_label_delimiter:n
8569         { #2 }
8570     }
8571   \cs_new:Nn
8572     \@@_latex_paralist_style:nn

```

```

8573 {
8574   \str_case:nn
8575     { #1 }
8576     {
8577       { Decimal } { 1 }
8578       { LowerRoman } { i }
8579       { UpperRoman } { I }
8580       { LowerAlpha } { a }
8581       { UpperAlpha } { A }
8582     }
8583   \@@_latex_fancy_list_item_label_delimiter:n
8584     { #2 }
8585 }
8586 \markdownSetup{rendererPrototypes={
8587   ulBeginTight = {\begin{compactitem}},
8588   ulEndTight = {\end{compactitem}},
8589   fancyOlBegin = {
8590     \group_begin:
8591     \tl_set:Nn
8592       \l_@@_latex_fancy_list_item_label_number_style_tl
8593       { #1 }
8594     \tl_set:Nn
8595       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8596       { #2 }
8597     \tl_set:Nn
8598       \l_tmpa_tl
8599       { \begin{enumerate}[ ]
8600     \tl_put_right:Nx
8601       \l_tmpa_tl
8602       { \@@_latex_paralist_style:nn { #1 } { #2 } }
8603     \tl_put_right:Nn
8604       \l_tmpa_tl
8605       { ] }
8606     \l_tmpa_tl
8607   },
8608   fancyOlEnd = {
8609     \end{enumerate}
8610     \group_end:
8611   },
8612   olBeginTight = {\begin{compactenum}},
8613   olEndTight = {\end{compactenum}},
8614   fancyOlBeginTight = {
8615     \group_begin:
8616     \tl_set:Nn
8617       \l_@@_latex_fancy_list_item_label_number_style_tl
8618       { #1 }
8619     \tl_set:Nn

```

```

8620     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8621     { #2 }
8622     \tl_set:Nn
8623     \l_tmpa_tl
8624     { \begin{compactenum}[ ]
8625     \tl_put_right:Nx
8626     \l_tmpa_tl
8627     { \@@_latex_paralist_style:nn { #1 } { #2 } }
8628     \tl_put_right:Nn
8629     \l_tmpa_tl
8630     { ] }
8631     \l_tmpa_tl
8632   },
8633   fancyO1EndTight = {
8634     \end{compactenum}
8635     \group_end:
8636   },
8637   fancyO1ItemWithNumber = {
8638     \item
8639     [
8640       \@@_latex_fancy_list_item_label:VVn
8641       \l_@@_latex_fancy_list_item_label_number_style_tl
8642       \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8643       { #1 }
8644     ]
8645   },
8646   dlBeginTight = {\begin{compactdesc}},
8647   dlEndTight = {\end{compactdesc}}}}
8648 \cs_generate_variant:Nn
8649 \@@_latex_fancy_list_item_label:nnn
8650 { VVn }
8651 }{
8652 \markdownSetup{rendererPrototypes={
8653   ulBeginTight = {\markdownRendererUlBegin},
8654   ulEndTight = {\markdownRendererUlEnd},
8655   fancyO1Begin = {\markdownRendererO1Begin},
8656   fancyO1End = {\markdownRendererO1End},
8657   olBeginTight = {\markdownRendererO1Begin},
8658   olEndTight = {\markdownRendererO1End},
8659   fancyO1BeginTight = {\markdownRendererO1Begin},
8660   fancyO1EndTight = {\markdownRendererO1End},
8661   dlBeginTight = {\markdownRendererDlBegin},
8662   dlEndTight = {\markdownRendererDlEnd}}}
8663 }
8664 \ExplSyntaxOff
8665 \RequirePackage{amsmath}

```

Unless the `unicode-math` package has been loaded, load the `amssymb` package with symbols to be used for tickboxes.

```

8666 \@ifpackageloaded{unicode-math}{
8667   \markdownSetup{rendererPrototypes={
8668     untickedBox = {\mdlgwhtsquare},
8669   }}
8670 }{
8671   \RequirePackage{amssymb}
8672   \markdownSetup{rendererPrototypes={
8673     untickedBox = {\square},
8674   }}
8675 }
8676 \RequirePackage{cvsimple}
8677 \RequirePackage{fancyvrb}
8678 \RequirePackage{graphicx}
8679 \markdownSetup{rendererPrototypes={
8680   lineBreak = {\},
8681   leftBrace = {\textbraceleft},
8682   rightBrace = {\textbraceright},
8683   dollarSign = {\textdollar},
8684   underscore = {\textunderscore},
8685   circumflex = {\textasciicircum},
8686   backslash = {\textbackslash},
8687   tilde = {\textasciitilde},
8688   pipe = {\textbar},

```

We can capitalize on the fact that the expansion of renderers is performed by \TeX during the typesetting. Therefore, even if we don't know whether a span of text is part of math formula or not when we are parsing markdown,⁸ 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.

```

8689   codeSpan = {%
8690     \ifmode
8691       \text{#1}%
8692     \else
8693       \texttt{#1}%
8694     \fi
8695   }}
8696 \ExplSyntaxOn
8697 \markdownSetup{
8698   rendererPrototypes = {
8699     contentBlock = {

```

⁸This 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.

```

8700   \str_case:nnF
8701     { #1 }
8702     {
8703       { csv }
8704       {
8705         \begin{table}
8706           \begin{center}
8707             \csvautotabular{#3}
8708           \end{center}
8709           \tl_if_empty:nF
8710             { #4 }
8711             { \caption{#4} }
8712         \end{table}
8713       }
8714       { tex } { \markdownEscape{#3} }
8715     }
8716     { \markdownInput{#3} }
8717   },
8718 },
8719 }
8720 \ExplSyntaxOff
8721 \markdownSetup{rendererPrototypes={
8722   image = {%
8723     \begin{figure}%
8724     \begin{center}%
8725       \includegraphics{#3}%
8726     \end{center}%
8727     \ifx\empty#4\empty\else
8728       \caption{#4}%
8729     \fi
8730   \end{figure}},
8731   ulBegin = {\begin{itemize}},
8732   ulEnd = {\end{itemize}},
8733   olBegin = {\begin{enumerate}},
8734   olItem = {\item{}},
8735   olItemWithNumber = {\item[#1.]},
8736   olEnd = {\end{enumerate}},
8737   dlBegin = {\begin{description}},
8738   dlItem = {\item[#1]},
8739   dlEnd = {\end{description}},
8740   emphasis = {\emph{#1}},
8741   tickedBox = {\$\boxtimes$},
8742   halfTickedBox = {\$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

8743   headerAttributeContextBegin = {

```

```

8744 \markdownSetup{
8745   rendererPrototypes = {
8746     attributeIdentifier = {%
8747       \begingroup
8748       \def\next####1{%
8749         \def####1#####1{%
8750           \endgroup
8751           ####1{#####1}%
8752           \label{##1}%
8753         }%
8754       }%
8755       \next\markdownRendererHeadingOne
8756       \next\markdownRendererHeadingTwo
8757       \next\markdownRendererHeadingThree
8758       \next\markdownRendererHeadingFour
8759       \next\markdownRendererHeadingFive
8760       \next\markdownRendererHeadingSix
8761     },
8762   },
8763 }%
8764 },
8765 superscript = {\textsuperscript{#1}},
8766 subscript = {\textsubscript{#1}},
8767 blockQuoteBegin = {\begin{quotation}},
8768 blockQuoteEnd = {\end{quotation}},
8769 inputVerbatim = {\VerbatimInput{#1}},
8770 inputFencedCode = {%
8771   \ifx\relax#2\relax
8772     \VerbatimInput{#1}%
8773   \else
8774     \@ifundefined{minted@code}{%
8775       \@ifundefined{lst@version}{%
8776         \markdownRendererInputFencedCode{#1}{}%

```

When the listings package is loaded, use it for syntax highlighting.

```

8777   }{%
8778     \lstinputlisting[language=#2]{#1}%
8779   }%

```

When the minted package is loaded, use it for syntax highlighting. The minted package is preferred over listings.

```

8780   }{%
8781     \catcode`\#=6\relax
8782     \inputminted{#2}{#1}%
8783     \catcode`\#=12\relax
8784   }%
8785 \fi},
8786 thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},

```

```
8787 note = {\footnote{#1}}}
```

Support the nesting of strong emphasis.

```
8788 \ExplSyntaxOn
8789 \def\markdownLATEXStrongEmphasis#1{%
8790   \str_if_in:NnTF
8791     \f@series
8792     { b }
8793     { \textnormal{#1} }
8794     { \textbf{#1} }
8795 }
8796 \ExplSyntaxOff
8797 \markdownSetup{rendererPrototypes={strongEmphasis={%
8798   \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support L^AT_EX document classes that do not provide chapters.

```
8799 \@ifundefined{chapter}{%
8800   \markdownSetup{rendererPrototypes = {
8801     headingOne = {\section{#1}},
8802     headingTwo = {\subsection{#1}},
8803     headingThree = {\subsubsection{#1}},
8804     headingFour = {\paragraph{#1}\leavevmode},
8805     headingFive = {\subparagraph{#1}\leavevmode}}}
8806 }{%
8807   \markdownSetup{rendererPrototypes = {
8808     headingOne = {\chapter{#1}},
8809     headingTwo = {\section{#1}},
8810     headingThree = {\subsection{#1}},
8811     headingFour = {\subsubsection{#1}},
8812     headingFive = {\paragraph{#1}\leavevmode},
8813     headingSix = {\subparagraph{#1}\leavevmode}}}
8814 }%
```

3.3.4.1 Tickboxes If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```
8815 \markdownSetup{
8816   rendererPrototypes = {
8817     ulItem = {%
8818       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
8819     },
8820   },
8821 }
8822 \def\markdownLaTeXUListItem{%
8823   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
8824     \item[\markdownLaTeXCheckbox]%
8825     \expandafter\@gobble
8826   \else
```



```

8827 \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
8828 \item[\markdownLaTeXCheckbox]%
8829 \expandafter\expandafter\expandafter\@gobble
8830 \else
8831 \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
8832 \item[\markdownLaTeXCheckbox]%
8833 \expandafter\expandafter\expandafter\expandafter
8834 \expandafter\expandafter\expandafter\@gobble
8835 \else
8836 \item{}%
8837 \fi
8838 \fi
8839 \fi
8840 }

```

3.3.4.2 HTML elements If the `html` option is enabled and we are using $\text{T}_{\text{E}}\text{X}_{4\text{ht}}$ ⁹, we will pass HTML elements to the output HTML document unchanged.

```

8841 \@ifundefined{HCode}{}{
8842 \markdownSetup{
8843   rendererPrototypes = {
8844     inlineHtmlTag = {%
8845       \ifvmode
8846         \IgnorePar
8847         \EndP
8848       \fi
8849       \HCode{#1}%
8850     },
8851     inputBlockHtmlElement = {%
8852       \ifvmode
8853         \IgnorePar
8854       \fi
8855       \EndP
8856       \special{t4ht* <#1}%
8857       \par
8858       \ShowPar
8859     },
8860   },
8861 }
8862 }

```

3.3.4.3 Citations Here is a basic implementation for citations that uses the $\text{L}_{\text{A}}\text{T}_{\text{E}}\text{X}$ `\cite` macro. There are also implementations that use the `natbib` `\citep`, and `\citet` macros, and the `BibL A \text{T}_{\text{E}}\text{X} \autocites and \textcites macros. These implementations will be used, when the respective packages are loaded.`

⁹See <https://tug.org/tex4ht/>.

```

8863 \newcount\markdownLaTeXCitationsCounter
8864
8865 % Basic implementation
8866 \RequirePackage{gobble}
8867 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
8868   \advance\markdownLaTeXCitationsCounter by 1\relax
8869   \ifx\relax#4\relax
8870     \ifx\relax#5\relax
8871       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8872         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
8873         \expandafter\expandafter\expandafter
8874         \expandafter\expandafter\expandafter\expandafter
8875         \@gobblethree
8876       \fi
8877     \else% Before a postnote (#5), dump the accumulator
8878       \ifx\relax#1\relax\else
8879         \cite{#1}%
8880       \fi
8881     \cite[#5]{#6}%
8882     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8883     \else
8884       \expandafter\expandafter\expandafter
8885       \expandafter\expandafter\expandafter\expandafter
8886       \expandafter\expandafter\expandafter
8887       \expandafter\expandafter\expandafter\expandafter
8888       \markdownLaTeXBasicCitations
8889     \fi
8890     \expandafter\expandafter\expandafter
8891     \expandafter\expandafter\expandafter\expandafter{%
8892     \expandafter\expandafter\expandafter
8893     \expandafter\expandafter\expandafter\expandafter}%
8894     \expandafter\expandafter\expandafter
8895     \expandafter\expandafter\expandafter\expandafter{%
8896     \expandafter\expandafter\expandafter
8897     \expandafter\expandafter\expandafter\expandafter}%
8898     \expandafter\expandafter\expandafter
8899     \@gobblethree
8900   \fi
8901   \else% Before a prenote (#4), dump the accumulator
8902     \ifx\relax#1\relax\else
8903       \cite{#1}%
8904     \fi
8905     \ifnum\markdownLaTeXCitationsCounter>1\relax
8906       \space % Insert a space before the prenote in later citations
8907     \fi
8908     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
8909     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax

```

```

8910     \else
8911         \expandafter\expandafter\expandafter
8912         \expandafter\expandafter\expandafter\expandafter
8913         \markdownLaTeXBasicCitations
8914     \fi
8915     \expandafter\expandafter\expandafter{%
8916     \expandafter\expandafter\expandafter}%
8917     \expandafter\expandafter\expandafter{%
8918     \expandafter\expandafter\expandafter}%
8919     \expandafter
8920     \@gobblethree
8921     \fi\markdownLaTeXBasicCitations{#1#2#6},}
8922 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
8923
8924 % Natbib implementation
8925 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
8926     \advance\markdownLaTeXCitationsCounter by 1\relax
8927     \ifx\relax#3\relax
8928         \ifx\relax#4\relax
8929             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8930                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
8931                 \expandafter\expandafter\expandafter
8932                 \expandafter\expandafter\expandafter\expandafter
8933                 \@gobbletwo
8934             \fi
8935         \else% Before a postnote (#4), dump the accumulator
8936             \ifx\relax#1\relax\else
8937                 \citep{#1}%
8938             \fi
8939             \citep[] [#4]{#5}%
8940             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8941                 \else
8942                     \expandafter\expandafter\expandafter
8943                     \expandafter\expandafter\expandafter\expandafter
8944                     \expandafter\expandafter\expandafter
8945                     \expandafter\expandafter\expandafter\expandafter
8946                     \markdownLaTeXNatbibCitations
8947                 \fi
8948                 \expandafter\expandafter\expandafter
8949                 \expandafter\expandafter\expandafter\expandafter{%
8950                 \expandafter\expandafter\expandafter
8951                 \expandafter\expandafter\expandafter\expandafter}%
8952                 \expandafter\expandafter\expandafter
8953                 \@gobbletwo
8954             \fi
8955         \else% Before a prenote (#3), dump the accumulator
8956             \ifx\relax#1\relax\relax\else

```

```

8957     \citep{#1}%
8958     \fi
8959     \citep[#3][#4]{#5}%
8960     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8961     \else
8962         \expandafter\expandafter\expandafter
8963         \expandafter\expandafter\expandafter\expandafter
8964         \markdownLaTeXNatbibCitations
8965     \fi
8966     \expandafter\expandafter\expandafter{%
8967     \expandafter\expandafter\expandafter}%
8968     \expandafter
8969     \@gobbletwo
8970 \fi\markdownLaTeXNatbibCitations{#1,#5}}
8971 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
8972     \advance\markdownLaTeXCitationsCounter by 1\relax
8973     \ifx\relax#3\relax
8974         \ifx\relax#4\relax
8975             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8976                 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
8977                 \expandafter\expandafter\expandafter
8978                 \expandafter\expandafter\expandafter\expandafter
8979                 \@gobbletwo
8980             \fi
8981         \else% After a prenote or a postnote, dump the accumulator
8982             \ifx\relax#1\relax\else
8983                 \citet{#1}%
8984             \fi
8985             , \citet[#3][#4]{#5}%
8986             \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8987                 ,
8988             \else
8989                 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8990                     ,
8991                 \fi
8992             \fi
8993             \expandafter\expandafter\expandafter
8994             \expandafter\expandafter\expandafter\expandafter
8995             \markdownLaTeXNatbibTextCitations
8996             \expandafter\expandafter\expandafter
8997             \expandafter\expandafter\expandafter\expandafter{%
8998             \expandafter\expandafter\expandafter
8999             \expandafter\expandafter\expandafter\expandafter}%
9000             \expandafter\expandafter\expandafter
9001             \@gobbletwo
9002         \fi
9003     \else% After a prenote or a postnote, dump the accumulator

```

```

9004 \ifx\relax#1\relax\relax\else
9005 \citet{#1}%
9006 \fi
9007 , \citet[#3][#4]{#5}%
9008 \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
9009 ,
9010 \else
9011 \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
9012 ,
9013 \fi
9014 \fi
9015 \expandafter\expandafter\expandafter
9016 \markdownLaTeXNatbibTextCitations
9017 \expandafter\expandafter\expandafter{%
9018 \expandafter\expandafter\expandafter}%
9019 \expandafter
9020 \@gobbletwo
9021 \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
9022
9023 % BibLaTeX implementation
9024 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
9025 \advance\markdownLaTeXCitationsCounter by 1\relax
9026 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9027 \autocites#1[#3][#4]{#5}%
9028 \expandafter\@gobbletwo
9029 \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
9030 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
9031 \advance\markdownLaTeXCitationsCounter by 1\relax
9032 \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
9033 \textcites#1[#3][#4]{#5}%
9034 \expandafter\@gobbletwo
9035 \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}
9036
9037 \markdownSetup{rendererPrototypes = {
9038 cite = {%
9039 \markdownLaTeXCitationsCounter=1%
9040 \def\markdownLaTeXCitationsTotal{#1}%
9041 \@ifundefined{autocites}{%
9042 \@ifundefined{citep}{%
9043 \expandafter\expandafter\expandafter
9044 \markdownLaTeXBasicCitations
9045 \expandafter\expandafter\expandafter{%
9046 \expandafter\expandafter\expandafter}%
9047 \expandafter\expandafter\expandafter{%
9048 \expandafter\expandafter\expandafter}%
9049 }{%
9050 \expandafter\expandafter\expandafter

```

```

9051         \markdownLaTeXNatbibCitations
9052         \expandafter\expandafter\expandafter{%
9053         \expandafter\expandafter\expandafter}%
9054     }%
9055 }{%
9056     \expandafter\expandafter\expandafter
9057     \markdownLaTeXBibLaTeXCitations
9058     \expandafter{\expandafter}%
9059 }},
9060 textCite = {%
9061     \markdownLaTeXCitationsCounter=1%
9062     \def\markdownLaTeXCitationsTotal{#1}%
9063     \@ifundefined{autocites}{%
9064         \@ifundefined{citep}{%
9065             \expandafter\expandafter\expandafter
9066             \markdownLaTeXBasicTextCitations
9067             \expandafter\expandafter\expandafter{%
9068             \expandafter\expandafter\expandafter}%
9069             \expandafter\expandafter\expandafter{%
9070             \expandafter\expandafter\expandafter}%
9071         }{%
9072             \expandafter\expandafter\expandafter
9073             \markdownLaTeXNatbibTextCitations
9074             \expandafter\expandafter\expandafter{%
9075             \expandafter\expandafter\expandafter}%
9076         }%
9077     }{%
9078         \expandafter\expandafter\expandafter
9079         \markdownLaTeXBibLaTeXTextCitations
9080         \expandafter{\expandafter}%
9081     }}}}

```

3.3.4.4 Links Before consuming the parameters for the hyperlink renderer, we change the category code of the hash sign (#) to other, so that it cannot be mistaken for a parameter character.

```

9082 \RequirePackage{url}
9083 \RequirePackage{expl3}
9084 \ExplSyntaxOn
9085 \def\markdownRendererLinkPrototype#1#2#3#4{
9086     \tl_set:Nn \l_tmpa_tl { #1 }
9087     \tl_set:Nn \l_tmpb_tl { #2 }
9088     \bool_set:Nn
9089     \l_tmpa_bool
9090     {
9091         \tl_if_eq_p:NN
9092         \l_tmpa_tl

```

```

9093     \l_tmpb_tl
9094   }
9095   \tl_set:Nn \l_tmpa_tl { #4 }
9096   \bool_set:Nn
9097     \l_tmpb_bool
9098     {
9099       \tl_if_empty_p:N
9100         \l_tmpa_tl
9101     }

```

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.

```

9102   \bool_if:nTF
9103     {
9104       \l_tmpa_bool && \l_tmpb_bool
9105     }
9106     {
9107       \markdownLaTeXRendererAutolink { #2 } { #3 }
9108     }{
9109       \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
9110     }
9111 }
9112 \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.

```

9113   \tl_set:Nn
9114     \l_tmpa_tl
9115     { #2 }
9116   \tl_trim_spaces:N
9117     \l_tmpa_tl
9118   \tl_set:Nx
9119     \l_tmpb_tl
9120     {
9121       \tl_range:Nnn
9122         \l_tmpa_tl
9123         { 1 }
9124         { 1 }
9125     }
9126   \str_if_eq:NNTF
9127     \l_tmpb_tl
9128     \c_hash_str
9129     {
9130       \tl_set:Nx
9131         \l_tmpb_tl
9132         {
9133           \tl_range:Nnn

```

```

9134         \l_tmpa_tl
9135         { 2 }
9136         { -1 }
9137     }
9138     \exp_args:NV
9139     \ref
9140     \l_tmpb_tl
9141 }{
9142     \url { #2 }
9143 }
9144 }
9145 \ExplSyntaxOff
9146 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
9147     #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}

```

3.3.4.5 Tables Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```

9148 \newcount\markdownLaTeXRowCount
9149 \newcount\markdownLaTeXRowTotal
9150 \newcount\markdownLaTeXColumnCounter
9151 \newcount\markdownLaTeXColumnTotal
9152 \newtoks\markdownLaTeXTable
9153 \newtoks\markdownLaTeXTableAlignment
9154 \newtoks\markdownLaTeXTableEnd
9155 \AtBeginDocument{%
9156     \@ifpackageloaded{booktabs}{%
9157         \def\markdownLaTeXTopRule{\toprule}%
9158         \def\markdownLaTeXMidRule{\midrule}%
9159         \def\markdownLaTeXBottomRule{\bottomrule}%
9160     }{%
9161         \def\markdownLaTeXTopRule{\hline}%
9162         \def\markdownLaTeXMidRule{\hline}%
9163         \def\markdownLaTeXBottomRule{\hline}%
9164     }%
9165 }
9166 \markdownSetup{rendererPrototypes={
9167     table = {%
9168         \markdownLaTeXTable={}%
9169         \markdownLaTeXTableAlignment={}%
9170         \markdownLaTeXTableEnd={%
9171             \markdownLaTeXBottomRule
9172             \end{tabular}}%
9173         \ifx\empty#1\empty\else
9174             \addto@hook\markdownLaTeXTable{%
9175                 \begin{table}
9176                 \centering}%

```



```

9177     \addto@hook\markdownLaTeXTableEnd{%
9178         \caption{#1}
9179         \end{table}}%
9180     \fi
9181     \addto@hook\markdownLaTeXTable{\begin{tabular}}%
9182     \markdownLaTeXRowCounter=0%
9183     \markdownLaTeXRowTotal=#2%
9184     \markdownLaTeXColumnTotal=#3%
9185     \markdownLaTeXRenderTableRow
9186 }
9187 }}
9188 \def\markdownLaTeXRenderTableRow#1{%
9189     \markdownLaTeXColumnCounter=0%
9190     \ifnum\markdownLaTeXRowCounter=0\relax
9191         \markdownLaTeXReadAlignments#1%
9192         \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
9193             \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
9194                 \the\markdownLaTeXTableAlignment}}%
9195         \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
9196     \else
9197         \markdownLaTeXRenderTableCell#1%
9198     \fi
9199     \ifnum\markdownLaTeXRowCounter=1\relax
9200         \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
9201     \fi
9202     \advance\markdownLaTeXRowCounter by 1\relax
9203     \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
9204         \the\markdownLaTeXTable
9205         \the\markdownLaTeXTableEnd
9206         \expandafter\@gobble
9207     \fi\markdownLaTeXRenderTableRow}
9208 \def\markdownLaTeXReadAlignments#1{%
9209     \advance\markdownLaTeXColumnCounter by 1\relax
9210     \if#1d%
9211         \addto@hook\markdownLaTeXTableAlignment{1}%
9212     \else
9213         \addto@hook\markdownLaTeXTableAlignment{#1}%
9214     \fi
9215     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
9216         \expandafter\@gobble
9217     \fi\markdownLaTeXReadAlignments}
9218 \def\markdownLaTeXRenderTableCell#1{%
9219     \advance\markdownLaTeXColumnCounter by 1\relax
9220     \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
9221         \addto@hook\markdownLaTeXTable{#1&}%
9222     \else
9223         \addto@hook\markdownLaTeXTable{#1\\}%

```

```

9224     \expandafter\@gobble
9225     \fi\markdownLaTeXRenderTableCell}

```

3.3.4.6 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.

```

9226 \ExplSyntaxOn
9227 \keys_define:nn
9228   { markdown/jekyllData }
9229   {
9230     author .code:n = { \author{#1} },
9231     date   .code:n = { \date{#1}   },
9232     title  .code:n = { \title{#1}  },
9233   }

```

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.

```

9234 % TODO: Remove the command definition in TeX Live 2021.
9235 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
9236 \markdownSetup{
9237   rendererPrototypes = {
9238     jekyllDataEnd = {
9239 %     TODO: Remove the else branch in TeX Live 2021.
9240       \IfFormatAtLeastTF
9241         { 2020-10-01 }
9242         { \AddToHook{begindocument/end}{\maketitle} }
9243         {
9244           \ifx\@onlypreamble\@notprerr
9245             % We are in the document
9246             \maketitle
9247           \else
9248             % We are in the preamble
9249             \RequirePackage{etoolbox}
9250             \AfterEndPreamble{\maketitle}
9251           \fi
9252         }
9253     },
9254   },
9255 }
9256 \ExplSyntaxOff

```

3.3.4.7 Strike-Through If the `strikeThrough` option is enabled, we will load the `soulutf8` package and use it to implement strike-throughs.

```

9257 \markdownIfOption{strikeThrough}{%
9258   \RequirePackage{soulutf8}%
9259   \markdownSetup{
9260     rendererPrototypes = {
9261       strikeThrough = {%
9262         \st{#1}%
9263       },
9264     }
9265   }
9266 }-{}

```

3.3.4.8 Raw Attribute Renderer Prototypes In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex` or `latex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

9267 \ExplSyntaxOn
9268 \cs_gset:Npn
9269   \markdownRendererInputRawInlinePrototype#1#2
9270   {
9271     \str_case:nn
9272       { #2 }
9273       {
9274         { tex   } { \markdownEscape{#1} }
9275         { latex } { \markdownEscape{#1} }
9276         { md    } { \markdownInput{#1}  }
9277       }
9278   }
9279 \cs_gset_eq:NN
9280   \markdownRendererInputRawBlockPrototype
9281   \markdownRendererInputRawInlinePrototype
9282 \ExplSyntaxOff
9283 \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.

```

9284 \newcommand\markdownMakeOther{%
9285   \count0=128\relax
9286   \loop
9287     \catcode\count0=11\relax
9288     \advance\count0 by 1\relax
9289   \ifnum\count0<256\repeat}%

```

3.4 ConTeXt Implementation

The ConTeXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTeXt formats *seem* to implement (the documentation is scarce) the majority of the plain TeX format required by the plain TeX implementation. As a consequence, we can directly reuse the existing plain TeX implementation after supplying the missing plain TeX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents L^AT_EX package.

```
9290 \def\markdownMakeOther{%
9291   \count0=128\relax
9292   \loop
9293     \catcode\count0=11\relax
9294     \advance\count0 by 1\relax
9295   \ifnum\count0<256\repeat
```

On top of that, make the pipe character (`|`) inactive during the scanning. This is necessary, since the character is active in ConTeXt.

```
9296   \catcode`|=12}%
```

3.4.1 Typesetting Markdown

The `\inputmarkdown` is defined to accept an optional argument with options recognized by the ConTeXt interface (see Section 2.4.2).

```
9297 \long\def\inputmarkdown{%
9298   \dosingleempty
9299   \doinputmarkdown}%
9300 \long\def\doinputmarkdown[#1]#2{%
9301   \begingroup
9302     \iffirstargument
9303       \setupmarkdown{#1}%
9304     \fi
9305     \markdownInput{#2}%
9306   \endgroup}%
```

The `\startmarkdown` and `\stopmarkdown` macros are implemented using the `\markdownReadAndConvert` macro.

In Knuth's TeX, trailing spaces are removed very early on when a line is being put to the input buffer. [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.

```

9307 \ifx\startluacode\undefined % MkII
9308   \begingroup
9309     \catcode`\|=0%
9310     \catcode`\|=12%
9311     |gdef|startmarkdown{%
9312       |markdownReadAndConvert{\stopmarkdown}%
9313                                   {|\stopmarkdown}}%
9314     |gdef|stopmarkdown{%
9315       |markdownEnd}%
9316   |endgroup
9317 \else % MkIV
9318   \startluacode
9319     document.markdown_buffering = false
9320     local function preserve_trailing_spaces(line)
9321       if document.markdown_buffering then
9322         line = line:gsub("[ \t][ \t]$", "\t\t")
9323       end
9324       return line
9325     end
9326     resolvers.installinputlinehandler(preserve_trailing_spaces)
9327   \stoptluacode
9328   \begingroup
9329     \catcode`\|=0%
9330     \catcode`\|=12%
9331     |gdef|startmarkdown{%
9332       |ctxlua{document.markdown_buffering = true}%
9333       |markdownReadAndConvert{\stopmarkdown}%
9334                                   {|\stopmarkdown}}%
9335     |gdef|stopmarkdown{%
9336       |ctxlua{document.markdown_buffering = false}%
9337       |markdownEnd}%
9338   |endgroup
9339 \fi

```

3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

9340 \def\markdownRendererLineBreakPrototype{\blank}%
9341 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
9342 \def\markdownRendererRightBracePrototype{\textbraceright}%
9343 \def\markdownRendererDollarSignPrototype{\textdollar}%
9344 \def\markdownRendererPercentSignPrototype{\percent}%
9345 \def\markdownRendererUnderscorePrototype{\textunderscore}%
9346 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
9347 \def\markdownRendererBackslashPrototype{\textbackslash}%
9348 \def\markdownRendererTildePrototype{\textasciitilde}%
9349 \def\markdownRendererPipePrototype{\char`|}%

```

```

9350 \def\markdownRendererLinkPrototype#1#2#3#4{%
9351   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
9352   \fi\texttt<\hyphenatedurl{#3}>}}%
9353 \usemodule[database]
9354 \defineseparatedlist
9355   [MarkdownConTeXtCSV]
9356   [separator={,},
9357   before=\bTABLE,after=\eTABLE,
9358   first=\bTR,last=\eTR,
9359   left=\bTD,right=\eTD]
9360 \def\markdownConTeXtCSV{csv}
9361 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
9362   \def\markdownConTeXtCSV@arg{#1}%
9363   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
9364     \placetable[] [tab:#1]{#4}{%
9365       \processeparatedfile[MarkdownConTeXtCSV][#3]}%
9366   \else
9367     \markdownInput{#3}%
9368   \fi}%
9369 \def\markdownRendererImagePrototype#1#2#3#4{%
9370   \placefigure[] []{#4}{\externalfigure[#3]}%
9371 \def\markdownRendererUlBeginPrototype{\startitemize}%
9372 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
9373 \def\markdownRendererUlItemPrototype{\item}%
9374 \def\markdownRendererUlEndPrototype{\stopitemize}%
9375 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
9376 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
9377 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
9378 \def\markdownRendererOlItemPrototype{\item}%
9379 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
9380 \def\markdownRendererOlEndPrototype{\stopitemize}%
9381 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
9382 \definedescription
9383   [MarkdownConTeXtDlItemPrototype]
9384   [location=hanging,
9385   margin=standard,
9386   headstyle=bold]%
9387 \definestartstop
9388   [MarkdownConTeXtDlPrototype]
9389   [before=\blank,
9390   after=\blank]%
9391 \definestartstop
9392   [MarkdownConTeXtDlTightPrototype]
9393   [before=\blank\startpacked,
9394   after=\stoppacked\blank]%
9395 \def\markdownRendererDlBeginPrototype{%
9396   \startMarkdownConTeXtDlPrototype}%

```

```

9397 \def\markdownRendererDlBeginTightPrototype{%
9398   \startMarkdownConTeXtDlTightPrototype}%
9399 \def\markdownRendererDlItemPrototype#1{%
9400   \startMarkdownConTeXtDlItemPrototype{#1}}%
9401 \def\markdownRendererDlItemEndPrototype{%
9402   \stopMarkdownConTeXtDlItemPrototype}%
9403 \def\markdownRendererDlEndPrototype{%
9404   \stopMarkdownConTeXtDlPrototype}%
9405 \def\markdownRendererDlEndTightPrototype{%
9406   \stopMarkdownConTeXtDlTightPrototype}%
9407 \def\markdownRendererEmphasisPrototype#1{\em#1}%
9408 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
9409 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
9410 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
9411 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
9412 \def\markdownRendererInputFencedCodePrototype#1#2{%
9413   \ifx\relax#2\relax
9414     \typefile{#1}%
9415   \else

```

The code fence infostring is used as a name from the ConT_EXt `\definetyping` macro. This 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

```

```

9416   \typefile[#2] []{#1}%
9417   \fi}%
9418 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
9419 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
9420 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
9421 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
9422 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
9423 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
9424 \def\markdownRendererThematicBreakPrototype{%

```

```

9425 \blackrule[height=1pt, width=\hsize]}%
9426 \def\markdownRendererNotePrototype#1{\footnote{#1}}%
9427 \def\markdownRendererTickedBoxPrototype{\$ \boxtimes$}
9428 \def\markdownRendererHalfTickedBoxPrototype{\$ \boxdot$}
9429 \def\markdownRendererUntickedBoxPrototype{\$ \square$}
9430 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
9431 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
9432 \def\markdownRendererSubscriptPrototype#1{\low{#1}}

```

3.4.2.1 Tables

There is a basic implementation of tables.

```

9433 \newcount\markdownConTeXtRowCounter
9434 \newcount\markdownConTeXtRowTotal
9435 \newcount\markdownConTeXtColumnCounter
9436 \newcount\markdownConTeXtColumnTotal
9437 \newtoks\markdownConTeXtTable
9438 \newtoks\markdownConTeXtTableFloat
9439 \def\markdownRendererTablePrototype#1#2#3{%
9440 \markdownConTeXtTable={}%
9441 \ifx\empty#1\empty
9442 \markdownConTeXtTableFloat={%
9443 \the\markdownConTeXtTable}%
9444 \else
9445 \markdownConTeXtTableFloat={%
9446 \placetable{#1}{\the\markdownConTeXtTable}}%
9447 \fi
9448 \begingroup
9449 \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
9450 \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
9451 \setupTABLE[r][1][topframe=on, bottomframe=on]
9452 \setupTABLE[r][#1][bottomframe=on]
9453 \markdownConTeXtRowCounter=0%
9454 \markdownConTeXtRowTotal=#2%
9455 \markdownConTeXtColumnTotal=#3%
9456 \markdownConTeXtRenderTableRow}
9457 \def\markdownConTeXtRenderTableRow#1{%
9458 \markdownConTeXtColumnCounter=0%
9459 \ifnum\markdownConTeXtRowCounter=0\relax
9460 \markdownConTeXtReadAlignments#1%
9461 \markdownConTeXtTable={\bTABLE}%
9462 \else
9463 \markdownConTeXtTable=\expandafter{%
9464 \the\markdownConTeXtTable\bTR}%
9465 \markdownConTeXtRenderTableRowCell#1%
9466 \markdownConTeXtTable=\expandafter{%
9467 \the\markdownConTeXtTable\eTR}%
9468 \fi

```



```

9469 \advance\markdownConTeXtRowCounter by 1\relax
9470 \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
9471   \markdownConTeXtTable=\expandafter{%
9472     \the\markdownConTeXtTable\eTABLE}%
9473   \the\markdownConTeXtTableFloat
9474   \endgroup
9475   \expandafter\gobbleoneargument
9476 \fi\markdownConTeXtRenderTableRow}
9477 \def\markdownConTeXtReadAlignments#1{%
9478   \advance\markdownConTeXtColumnCounter by 1\relax
9479   \if#1d%
9480     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
9481   \fi\if#1l%
9482     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
9483   \fi\if#1c%
9484     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
9485   \fi\if#1r%
9486     \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
9487   \fi
9488   \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
9489     \expandafter\gobbleoneargument
9490   \fi\markdownConTeXtReadAlignments}
9491 \def\markdownConTeXtRenderTableCell#1{%
9492   \advance\markdownConTeXtColumnCounter by 1\relax
9493   \markdownConTeXtTable=\expandafter{%
9494     \the\markdownConTeXtTable\bTD#1\eTD}%
9495   \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
9496     \expandafter\gobbleoneargument
9497   \fi\markdownConTeXtRenderTableCell}

```

3.4.2.2 Raw Attribute Renderer Prototypes In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex` or `context`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```

9498 \ExplSyntaxOn
9499 \cs_gset:Npn
9500   \markdownRendererInputRawInlinePrototype#1#2
9501   {
9502     \str_case:nn
9503       { #2 }
9504       {
9505         { tex      } { \markdownEscape{#1} }
9506         { context } { \markdownEscape{#1} }
9507         { md       } { \markdownInput{#1}  }
9508       }
9509   }

```

```
9510 \cs_gset_eq:NN
9511   \markdownRendererInputRawBlockPrototype
9512   \markdownRendererInputRawInlinePrototype
9513 \ExplSyntaxOff
9514 \stopmodule\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 Miletić. *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 Novotný. *L^AT_EX 2_ε no longer keys packages by pathnames*. Feb. 20, 2021. URL: <https://github.com/latex3/latex2e/issues/510> (visited on 02/21/2021).
- [10] Geoffrey M. Poore. *The minted Package. Highlighted source code in L^AT_EX*. July 19, 2017. URL: <https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf> (visited on 09/01/2020).
- [11] Roberto Ierusalimsky. *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 L^AT_EX 2_ε Sources*. Apr. 15, 2017. URL: <https://mirrors.ctan.org/macros/latex/base/source2e.pdf> (visited on 01/08/2018).

- [13] Donald Ervin Knuth. *TEX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [14] Victor Eijkhout. *TEX by Topic. A TEXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

Index

<code>blankBeforeBlockquote</code>	16
<code>blankBeforeCodeFence</code>	17
<code>blankBeforeDivFence</code>	17
<code>blankBeforeHeading</code>	17
<code>bracketedSpans</code>	18, 51
<code>breakableBlockquotes</code>	18
<code>cacheDir</code>	3, 15, 21, 44, 45, 100, 198, 242, 254
<code>citationNbsps</code>	18
<code>citations</code>	19, 72, 78
<code>codeSpans</code>	19
<code>contentBlocks</code>	15, 20
<code>contentBlocksLanguageMap</code>	15
<code>debugExtensions</code>	8, 15, 20, 195
<code>debugExtensionsFileName</code>	15, 20
<code>defaultOptions</code>	9, 40, 229
<code>definitionLists</code>	20, 56
<code>eagerCache</code>	21, 21
<code>\endmarkdown</code>	88
<code>entities.char_entity</code>	156
<code>entities.dec_entity</code>	156
<code>entities.hex_entity</code>	156
<code>errorTempFileName</code>	46, 248
<code>escape_citation</code>	200, 201
<code>escaped_citation_chars</code>	200, 200
<code>expandtabs</code>	180
<code>expectJekyllData</code>	22
<code>extensions</code>	23, 107, 199
<code>extensions.bracketed_spans</code>	199
<code>extensions.citations</code>	200
<code>extensions.content_blocks</code>	204
<code>extensions.definition_lists</code>	207
<code>extensions.fancy_lists</code>	209

<code>extensions.fenced_code</code>	213
<code>extensions.fenced_divs</code>	214
<code>extensions.header_attributes</code>	216
<code>extensions.jekyll_data</code>	226
<code>extensions.notes</code>	217
<code>extensions.pipe_table</code>	219
<code>extensions.raw_attribute</code>	222
<code>extensions.strike_through</code>	224
<code>extensions.subscripts</code>	224
<code>extensions.superscripts</code>	225
<code>fancyLists</code>	24, 68–72
<code>fencedCode</code>	25, 31, 54, 73, 253
<code>fencedDiv</code>	59
<code>fencedDivs</code>	25
<code>finalizeCache</code>	16, 21, 26, 26, 44, 45, 100, 199
<code>frozenCache</code>	16, 26, 45, 94, 95, 100, 253, 254
<code>frozenCacheCounter</code>	26, 199, 250
<code>frozenCacheFileName</code>	16, 26, 44, 199
<code>hardLineBreaks</code>	27
<code>hashEnumerators</code>	27
<code>headerAttributes</code>	27, 33, 49, 60
<code>helperScriptFileName</code>	45–47, 248
<code>html</code>	28, 62, 63, 265
<code>hybrid</code>	28, 31, 36, 37, 46, 74, 95, 100, 160, 181, 201, 250
<code>inlineNotes</code>	28
<code>\inputmarkdown</code>	104, 104, 105, 276
<code>inputTempFileName</code>	244, 245
<code>iterlines</code>	180
<code>jekyllData</code>	3, 22, 29, 80–83
<code>languages_json</code>	204, 204
<code>\markdown</code>	88
<code>markdown</code>	88, 88, 252
<code>markdown*</code>	88, 88, 89, 252
<code>\markdown_jekyll_data_concatenate_address:NN</code>	238
<code>\markdown_jekyll_data_pop:</code>	238
<code>\markdown_jekyll_data_push:nN</code>	238
<code>\markdown_jekyll_data_push_address_segment:n</code>	237

<code>\markdown_jekyll_data_set_keyval:Nn</code>	239
<code>\markdown_jekyll_data_set_keyvals:nn</code>	239
<code>\markdown_jekyll_data_update_address_tls:</code>	238
<code>\markdownBegin</code>	42, 42, 43, 86, 88, 104
<code>\markdownEnd</code>	42, 42, 43, 86, 88, 104
<code>\markdownError</code>	85, 85
<code>\markdownEscape</code>	42, 43, 44, 251
<code>\markdownExecute</code>	247
<code>\markdownExecuteDirect</code>	247, 247
<code>\markdownExecuteShellEscape</code>	246, 247
<code>\markdownIfOption</code>	242
<code>\markdownIfSnippetExists</code>	90
<code>\markdownInfo</code>	85
<code>\markdownInput</code>	42, 43, 88, 89, 104, 105, 249, 251
<code>\markdownInputFileStream</code>	243
<code>\markdownInputPlainTeX</code>	251
<code>\markdownLuaExecute</code>	246, 247, 248, 249
<code>\markdownLuaOptions</code>	240, 242
<code>\markdownLuaRegisterIBCallback</code>	87
<code>\markdownLuaUnregisterIBCallback</code>	87
<code>\markdownMakeOther</code>	86, 275, 276
<code>\markdownMode</code>	4, 45, 46, 86, 86, 87, 246, 249
<code>\markdownOptionErrorTempFileName</code>	46
<code>\markdownOptionFinalizeCache</code>	44
<code>\markdownOptionFrozenCache</code>	44
<code>\markdownOptionHelperScriptFileName</code>	45
<code>\markdownOptionHybrid</code>	46
<code>\markdownOptionInputTempFileName</code>	45
<code>\markdownOptionOutputDir</code>	46
<code>\markdownOptionOutputTempFileName</code>	45
<code>\markdownOptionStripPercentSigns</code>	48
<code>\markdownOutputFileStream</code>	243
<code>\markdownPrepare</code>	242
<code>\markdownPrepareLuaOptions</code>	240
<code>\markdownReadAndConvert</code>	86, 243, 252, 276
<code>\markdownReadAndConvertProcessLine</code>	245, 245
<code>\markdownReadAndConvertStripPercentSigns</code>	244
<code>\markdownReadAndConvertTab</code>	243
<code>\markdownRendererAttributeClassName</code>	49
<code>\markdownRendererAttributeIdentifier</code>	49
<code>\markdownRendererAttributeKeyValue</code>	49
<code>\markdownRendererBlockHtmlCommentBegin</code>	63

<code>\markdownRendererBlockHtmlCommentEnd</code>	63
<code>\markdownRendererBlockQuoteBegin</code>	50
<code>\markdownRendererBlockQuoteEnd</code>	50
<code>\markdownRendererBracketedSpanAttributeContextBegin</code>	51
<code>\markdownRendererBracketedSpanAttributeContextEnd</code>	51
<code>\markdownRendererCite</code>	72, 78
<code>\markdownRendererCodeSpan</code>	54
<code>\markdownRendererContentBlock</code>	54, 55
<code>\markdownRendererContentBlockCode</code>	55
<code>\markdownRendererContentBlockOnlineImage</code>	55
<code>\markdownRendererDlBegin</code>	56
<code>\markdownRendererDlBeginTight</code>	56
<code>\markdownRendererDlDefinitionBegin</code>	57
<code>\markdownRendererDlDefinitionEnd</code>	57
<code>\markdownRendererDlEnd</code>	58
<code>\markdownRendererDlEndTight</code>	58
<code>\markdownRendererDlItem</code>	56
<code>\markdownRendererDlItemEnd</code>	57
<code>\markdownRendererDocumentBegin</code>	65
<code>\markdownRendererDocumentEnd</code>	65
<code>\markdownRendererEllipsis</code>	33, 58
<code>\markdownRendererEmphasis</code>	59, 102
<code>\markdownRendererFancyOlBegin</code>	68, 69
<code>\markdownRendererFancyOlBeginTight</code>	69
<code>\markdownRendererFancyOlEnd</code>	72
<code>\markdownRendererFancyOlEndTight</code>	72
<code>\markdownRendererFancyOlItem</code>	70
<code>\markdownRendererFancyOlItemEnd</code>	70
<code>\markdownRendererFancyOlItemWithNumber</code>	71
<code>\markdownRendererFencedDivAttributeContextBegin</code>	59
<code>\markdownRendererFencedDivAttributeContextEnd</code>	59
<code>\markdownRendererFootnote</code>	66, 84
<code>\markdownRendererFootnotePrototype</code>	66, 84
<code>\markdownRendererHalfTickedBox</code>	79
<code>\markdownRendererHeaderAttributeContextBegin</code>	60
<code>\markdownRendererHeaderAttributeContextEnd</code>	60
<code>\markdownRendererHeadingFive</code>	62
<code>\markdownRendererHeadingFour</code>	62
<code>\markdownRendererHeadingOne</code>	61
<code>\markdownRendererHeadingSix</code>	62
<code>\markdownRendererHeadingThree</code>	61
<code>\markdownRendererHeadingTwo</code>	61

<code>\markdownRendererHorizontalRule</code>	78, 84
<code>\markdownRendererHorizontalRulePrototype</code>	78, 84
<code>\markdownRendererImage</code>	64
<code>\markdownRendererInlineHtmlComment</code>	62
<code>\markdownRendererInlineHtmlTag</code>	63
<code>\markdownRendererInputBlockHtmlElement</code>	63
<code>\markdownRendererInputFencedCode</code>	54
<code>\markdownRendererInputRawBlock</code>	73
<code>\markdownRendererInputRawInline</code>	73
<code>\markdownRendererInputVerbatim</code>	53
<code>\markdownRendererInterblockSeparator</code>	64
<code>\markdownRendererJekyllDataBegin</code>	80
<code>\markdownRendererJekyllDataBoolean</code>	82
<code>\markdownRendererJekyllDataEmpty</code>	83
<code>\markdownRendererJekyllDataEnd</code>	81
<code>\markdownRendererJekyllDataMappingBegin</code>	81
<code>\markdownRendererJekyllDataMappingEnd</code>	81
<code>\markdownRendererJekyllDataNumber</code>	83
<code>\markdownRendererJekyllDataSequenceBegin</code>	82
<code>\markdownRendererJekyllDataSequenceEnd</code>	82
<code>\markdownRendererJekyllDataString</code>	83
<code>\markdownRendererLineBreak</code>	65
<code>\markdownRendererLink</code>	65, 102
<code>\markdownRendererNbsp</code>	66
<code>\markdownRendererNote</code>	66
<code>\markdownRendererOlBegin</code>	67
<code>\markdownRendererOlBeginTight</code>	68
<code>\markdownRendererOlEnd</code>	71
<code>\markdownRendererOlEndTight</code>	71
<code>\markdownRendererOlItem</code>	33, 69
<code>\markdownRendererOlItemEnd</code>	69
<code>\markdownRendererOlItemWithNumber</code>	33, 70
<code>\markdownRendererStrikeThrough</code>	76
<code>\markdownRendererStrongEmphasis</code>	59
<code>\markdownRendererSubscript</code>	77
<code>\markdownRendererSuperscript</code>	77
<code>\markdownRendererTable</code>	77
<code>\markdownRendererTextCite</code>	78
<code>\markdownRendererThematicBreak</code>	78
<code>\markdownRendererTickedBox</code>	79
<code>\markdownRendererUlBegin</code>	51
<code>\markdownRendererUlBeginTight</code>	52

<code>\markdownRendererUPEnd</code>	53
<code>\markdownRendererUPEndTight</code>	53
<code>\markdownRendererUItem</code>	52
<code>\markdownRendererUItemEnd</code>	52
<code>\markdownRendererUntickedBox</code>	79
<code>\markdownSetup</code>	89, 89, 251, 255
<code>\markdownSetupSnippet</code>	89, 89
<code>\markdownWarning</code>	85
<code>new</code>	6, 229
<code>notes</code>	30, 66
<code>outputTempFileName</code>	46, 248
<code>parsers</code>	167, 180
<code>parsers.commented_line</code>	169
<code>pipeTables</code>	6, 30, 35, 77
<code>preserveTabs</code>	31, 34, 180
<code>rawAttribute</code>	31, 31, 73
<code>reader</code>	7, 107, 167, 179, 199
<code>reader->add_special_character</code>	7, 8, 194
<code>reader->create_parser</code>	181
<code>reader->finalize_grammar</code>	191
<code>reader->initialize_named_group</code>	194
<code>reader->insert_pattern</code>	7, 8, 191, 196
<code>reader->normalize_tag</code>	180
<code>reader->options</code>	179
<code>reader->parser_functions</code>	181
<code>reader->parser_functions.name</code>	181
<code>reader->parsers</code>	180, 180
<code>reader->update_rule</code>	191, 193, 196
<code>reader->writer</code>	179
<code>reader.new</code>	179, 179
<code>relativeReferences</code>	32
<code>\setupmarkdown</code>	105, 105
<code>shiftHeadings</code>	6, 32
<code>slice</code>	6, 33, 157
<code>smartEllipses</code>	33, 58, 100
<code>\startmarkdown</code>	104, 104, 276
<code>startNumber</code>	33, 69–71
<code>\stopmarkdown</code>	104, 104, 276

strikeThrough	34, 76, 274
stripIndent	34, 181
stripPercentSigns	243, 244
subscripts	35, 77
superscripts	35, 77
syntax	192, 196
tableCaptions	6, 35
taskLists	36, 79, 264
texComments	36, 181
tightLists	37, 52, 53, 56, 58, 68, 69, 71, 72, 257
underscores	37
util.cache	108, 109
util.cache_verbatim	109
util.encode_json_string	109
util.err	108
util.escaper	112
util.expand_tabs_in_line	109
util.flatten	110
util.intersperse	111
util.lookup_files	109
util.map	111
util.pathname	113
util.rope_last	111
util.rope_to_string	111
util.table_copy	109
util.walk	110, 111
walkable_syntax	7, 15, 20, 191, 193, 195, 196
writer	107, 107, 157, 199
writer->active_attributes	164
writer->active_headings	164
writer->attributes	164
writer->block_html_comment	162
writer->block_html_element	162
writer->blockquote	163
writer->bulletitem	161
writer->bulletlist	160
writer->citation	201
writer->citations	201
writer->code	160
writer->codeFence	213

writer->contentblock	205
writer->defer_call	167, 167
writer->definitionlist	207
writer->div	214
writer->document	163
writer->ellipsis	159
writer->emphasis	162
writer->escape	159, 160
writer->escape_minimal	159, 160, 201
writer->escape_uri	159, 160
writer->escaped_chars	159, 159
writer->escaped_minimal_strings	159, 159
writer->escaped_uri_chars	159, 159
writer->fancyitem	210
writer->fancylist	209
writer->get_state	166
writer->heading	164
writer->image	160
writer->inline_html_comment	162
writer->inline_html_tag	162
writer->interblocksep	158
writer->is_writing	157, 157
writer->jekyllData	226
writer->linebreak	158
writer->link	160
writer->nbsp	158
writer->note	218
writer->options	157
writer->ordereditem	161
writer->orderedlist	161
writer->pack	158, 198
writer->paragraph	158
writer->plain	158
writer->rawBlock	223
writer->rawInline	222
writer->set_state	167
writer->slice_begin	157
writer->slice_end	157
writer->space	158
writer->span	199
writer->strike_through	224
writer->string	160

<code>writer->strong</code>	<i>163</i>
<code>writer->subscript</code>	<i>224</i>
<code>writer->suffix</code>	<i>158</i>
<code>writer->superscript</code>	<i>225</i>
<code>writer->table</code>	<i>220</i>
<code>writer->thematic_break</code>	<i>159</i>
<code>writer->checkbox</code>	<i>162</i>
<code>writer->uri</code>	<i>160</i>
<code>writer->verbatim</code>	<i>163</i>
<code>writer.new</code>	<i>157, 157</i>