

A Markdown Interpreter for T_EX

Vít Novotný
witiko@mail.muni.cz

Version 2.17.0-0-g6428569
2022-09-30

Contents

1	Introduction	1	3	Implementation	98
1.1	Requirements	2	3.1	Lua Implementation	98
1.2	Feedback	5	3.2	Plain T _E X Implementation	217
1.3	Acknowledgements	5	3.3	L ^A T _E X Implementation . .	233
2	Interfaces	6	3.4	ConT _E Xt Implementation	256
2.1	Lua Interface	6			
2.2	Plain T _E X Interface	37			
2.3	L ^A T _E X Interface	79			
2.4	ConT _E Xt Interface	95			
				References	262
				Index	263

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 . .	34
3	A sequence diagram of typesetting a document using the Lua CLI	35
4	Various formats of mathematical formulae	86
5	The banner of the Markdown package	87
6	A pushdown automaton that recognizes T _E X comments	160

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 converters, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😊

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

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 LuaTeX engine:

LPeg ≥ 0.10 A pattern-matching library for the writing of recursive descent parsers via the Parsing Expression Grammars (PEGs). It is used by the Lunamark library to parse the markdown input. LPeg ≥ 0.10 is included in LuaTeX $\geq 0.72.0$ (TeXLive ≥ 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 footnote tags to the lower case. Selene Unicode is included in all releases of LuaTeX (TeXLive ≥ 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 ≥ 5.3 , we will use the built-in support for Unicode.

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

```

17   if not ran_ok then
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/lua-tinyyaml 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 `\markdownOptionCacheDir` macro 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^ATeX Requirements

The L^ATeX part of the package requires that the L^ATeX 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^ATeX 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^ATeX theme (see Section 2.3.2.2), and to provide default token renderer prototypes.

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 \geq 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 $2006 \leq \text{T_EX Live} \leq 2019$. Since $\text{T_EX Live} \geq 2020$, the functionality of the package has been integrated in the L^AT_EX 2_ε kernel. It is used in the [witiko/dot](#) and [witiko/graphicx/http](#) L^AT_EX themes, see Section 2.3.2.2.

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 \TeX implementation of the package draws inspiration from several sources including the source code of $\text{\LaTeX} 2_{\epsilon}$, the minted package by Geoffrey M. Poore, which likewise tackles the issue of interfacing with an external interpreter from \TeX , 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 \TeX 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 \TeX *token renderers* is exposed by the Lua layer. The plain \TeX layer exposes the conversion capabilities of Lua as \TeX macros. The \LaTeX and $\text{Con}\text{\TeX}$ t layers provide syntactic sugar on top of plain \TeX 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 \TeX . This interface is used by the plain \TeX 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 \TeX

The Lua interface exposes the `new(options)` function. This function returns a conversion function from markdown to plain \TeX 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.

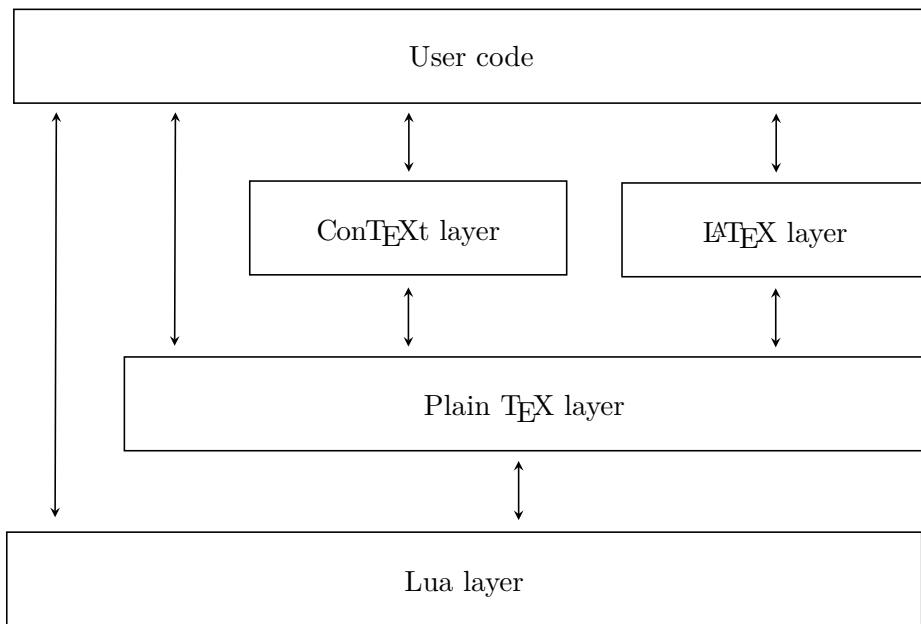


Figure 1: A block diagram of the Markdown package

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     "HorizontalRule",
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 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. 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_put_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_put_right:cn
```

```
82     { g_@@_ #1 _options_seq }
```

```
83     { #2 }
```

```
84     \prop_put:cnn
```

```
85     { g_@@_ #1 _option_types_prop }
```

```
86     { #2 }
```

```
87     { #3 }
```

```
88     \prop_put: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_put_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
148 \tl_const:Nn \c_@@_option_type_counter_tl { counter }
149 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl

```

```

150 \tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
151 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
152 \tl_const:Nn \c_@@_option_type_number_tl { number }
153 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
154 \tl_const:Nn \c_@@_option_type_path_tl { path }
155 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
156 \tl_const:Nn \c_@@_option_type_slice_tl { slice }
157 \seq_put_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
158 \tl_const:Nn \c_@@_option_type_string_tl { string }
159 \seq_put_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 }

```

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.

```
290 \@@_add_lua_option:nnn
291   { cacheDir }
292   { path }
293   { \markdownOptionOutputDir / _markdown_\jobname }
294 defaultOptions.cacheDir = "."
```

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

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

The frozen cache makes it possible to later typeset a plain T_EX document that contains markdown documents without invoking Lua using the `\markdownOptionFrozenCache` 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.

```
295 \@@_add_lua_option:nnn
296   { frozenCacheFileName }
297   { path }
298   { \markdownOptionCacheDir / frozenCache.tex }
299 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.

```
300 \@@_add_lua_option:nnn
301   { blankBeforeBlockquote }
302   { boolean }
303   { false }
304 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.

```
305 \@@_add_lua_option:nnn
306   { blankBeforeCodeFence }
307   { boolean }
308   { false }
309 defaultOptions.blankBeforeCodeFence = 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.

```
310 \@@_add_lua_option:nnn
311   { blankBeforeHeading }
312   { boolean }
313   { false }
314 defaultOptions.blankBeforeHeading = 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.

```
315 \@@_add_lua_option:nnn
316   { breakableBlockquotes }
317   { boolean }
318   { false }
319 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.

```
320 \@@_add_lua_option:nnn
321   { citationNbsps }
322   { boolean }
323   { true }
324 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.

```
325 \@@_add_lua_option:nnn
326   { citations }
327   { boolean }
328   { false }

329 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.``
```

```
330 \@@_add_lua_option:nnn
331   { codeSpans }
332   { boolean }
333   { true }

334 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.

```
335 \@@_add_lua_option:nnn
336   { contentBlocks }
337   { boolean }
338   { false }

339 defaultOptions.contentBlocks = false
```

`contentBlocksLanguageMap=<filename>`

default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA,Writer content blocks. See Section 2.2.3.11 for more information.

```
340 \@@_add_lua_option:nnn
341   { contentBlocksLanguageMap }
342   { path }
343   { markdown-languages.json }
344 defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`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.

```
345 \@@_add_lua_option:nnn
346   { definitionLists }
347   { boolean }
348   { false }
349 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.

```
350 \@@_add_lua_option:nnn
351   { eagerCache }
352   { boolean }
353   { true }
354 defaultOptions.eagerCache = true
```

`extensions`= $\langle filenames \rangle$

The filenames of user-defined syntax extensions that will be applied to the markdown reader. If the kpathsea library is available, files will be searched for not only in the current working directory but also in the T_EX directory structure.

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

```
local strike_through = {
  api_version = 1,
  grammar_version = 1,
  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"), doubletildes, doubletildes
    ) / function(s) return {"\\st{" , s, "}"} end

    reader.insert_pattern("Inline after Emph", read_strike_through)
    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:

```
355 metadata.user_extension_api_version = 1
356 metadata.grammar_version = 1
```

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>`.

```
357 \cs_generate_variant:Nn
358   \@@_add_lua_option:nnn
359   { nnV }
360 \@@_add_lua_option:nnV
361   { extensions }
362   { clist }
363   \c_empty_clist
364 defaultOptions.extensions = {}
```

`expectJekyllData=true, false`

default: false

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

```
\documentclass{article}
\usepackage[jekyllData]{markdown}
\begin{document}
\begin{markdown}
---
- this
- is
- YAML
```

```

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

```

`true`

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

```

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

```

```

365 \@@_add_lua_option:nnn
366   { expectJekyllData }
367   { boolean }
368   { false }
369 defaultOptions.expectJekyllData = false

```

`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.

```
370 \@@_add_lua_option:nnn
371 { fancyLists }
372 { boolean }
373 { false }
374 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
<pre>
 <code>
 // Some comments
 line 1 of code
 line 2 of code
 line 3 of code
 </code>
</pre>
```
```

`false` Disable the commonmark fenced code block extension.

```
375 \@@_add_lua_option:nnn
376 { fencedCode }
377 { boolean }
378 { false }
379 defaultOptions.fencedCode = 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 `\markdownOptionFrozenCache` 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.

```
380 \@@_add_lua_option:nnn
381   { finalizeCache }
382   { boolean }
383   { false }
384 defaultOptions.finalizeCache = false
```

`footnotes=true, false`

default: false

`true` Enable the Pandoc footnote syntax extension:

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

`[^1]`: Here is the footnote.

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

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

`{ some.code }`

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

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

`false` Disable the Pandoc footnote syntax extension.

```
385 \@@_add_lua_option:nnn
386   { footnotes }
```

```

387 { boolean }
388 { false }

389 defaultOptions.footnotes = false

```

frozenCacheCounter=*<number>* default: 0

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

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

```

390 \@@_add_lua_option:nnn
391 { frozenCacheCounter }
392 { counter }
393 { 0 }

394 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.

```

395 \@@_add_lua_option:nnn
396 { hardLineBreaks }
397 { boolean }
398 { false }

399 defaultOptions.hardLineBreaks = false

```

hashEnumerators=true, false default: false

true Enable the use of hash symbols (#) as ordered item list markers:

| |
|--|
| <pre> #. Bird #. McHale #. Parish </pre> |
|--|

false Disable the use of hash symbols (#) as ordered item list markers.


```

400 \@@_add_lua_option:nnn
401   { hashEnumerators }
402   { boolean }
403   { false }
404 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}
=====

```

These HTML attributes have currently no effect other than enabling content slicing, see the `slice` option.

false Disable the assignment of HTML attributes to headings.

```

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

```

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

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

`inlineFootnotes=true, false`

default: false

- true** Enable the Pandoc inline footnote 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 footnote syntax extension.

```
420 \@@_add_lua_option:nnn
421   { inlineFootnotes }
422   { boolean }
423   { false }
424 defaultOptions.inlineFootnotes = 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 `yml_metadata_block` syntax extension for entering metadata in YAML.

```

425 \@@_add_lua_option:nnn
426 { jekyllData }
427 { boolean }
428 { false }
429 defaultOptions.jekyllData = 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.

```

430 \@@_add_lua_option:nnn
431 { pipeTables }
432 { boolean }
433 { false }
434 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.

```

435 \@@_add_lua_option:nnn
436 { preserveTabs }
437 { boolean }
438 { false }
439 defaultOptions.preserveTabs = false
```

`relativeReferences=true, false`

default: false

`true` Enable relative references⁶ 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.

```
440 \@@_add_lua_option:nnn
441 { relativeReferences }
442 { boolean }
443 { false }
444 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.

```
445 \@@_add_lua_option:nnn
446 { shiftHeadings }
447 { number }
448 { 0 }
449 defaultOptions.shiftHeadings = 0
```

`slice=<the beginning and the end of a slice>`

default: ^ \$

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

- The circumflex (^) selects the beginning of a document.
- The dollar sign (\$) selects the end of a document.
- ^<identifier> selects the beginning of a section with the HTML attribute #<identifier> (see the `headerAttributes` option).

⁶See <https://datatracker.ietf.org/doc/html/rfc3986#section-4.2>.

- $\$ \langle identifier \rangle$ selects the end of a section with the HTML attribute $\# \langle identifier \rangle$.
- $\langle identifier \rangle$ corresponds to $\wedge \langle identifier \rangle$ for the first selector and to $\$ \langle identifier \rangle$ for the second selector.

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

```

450 \@@_add_lua_option:nnn
451   { slice }
452   { slice }
453   { ^~$ }
454 defaultOptions.slice = "^ $"

```

`smartEllipses=true, false` default: false

- true** Convert any ellipses in the input to the `\markdownRenderEllipsis` TeX macro.
- false** Preserve all ellipses in the input.

```

455 \@@_add_lua_option:nnn
456   { smartEllipses }
457   { boolean }
458   { false }
459 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 `\markdownRenderOliItemWithNumber` TeX macro.
- false** Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRenderOliItem` TeX macro.

```

460 \@@_add_lua_option:nnn
461   { startNumber }
462   { boolean }
463   { true }
464 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.

```
465 \@@_add_lua_option:nnn
466 { strikeThrough }
467 { boolean }
468 { false }
469 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.

```
470 \@@_add_lua_option:nnn
471 { stripIndent }
472 { boolean }
473 { false }
474 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.

```

475 \@@_add_lua_option:nnn
476   { subscripts }
477   { boolean }
478   { false }
479 defaultOptions.subscripts = false

```

`superscripts=true, false`

default: false

true Enable the Pandoc superscript syntax extension:

| |
|---------------------|
| 2^{10^4} is 1024. |
|---------------------|

false Disable the Pandoc superscript syntax extension.

```

480 \@@_add_lua_option:nnn
481   { superscripts }
482   { boolean }
483   { false }
484 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.

```

485 \@@_add_lua_option:nnn
486   { tableCaptions }
487   { boolean }
488   { false }
489 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.

```
490 \@@_add_lua_option:nnn
491   { taskLists }
492   { boolean }
493   { false }
494 defaultOptions.taskLists = false
```

`texComments=true, false`

default: false

true Strip T_EX-style comments.

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

Always enabled when `hybrid` is enabled.

false Do not strip T_EX-style comments.

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

```
500 \@@_add_lua_option:nnn
501   { tightLists }
502   { boolean }
503   { true }
504 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.

```
505 \@@_add_lua_option:nnn
506   { underscores }
507   { boolean }
508   { true }
509 \ExplSyntaxOff
510 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 \TeX layer hands markdown documents to the Lua layer. Lua converts the documents to \TeX , and hands the converted documents back to plain \TeX layer for typesetting, see Figure 2.

This procedure has the advantage of being fully automated. However, it also has several important disadvantages: The converted \TeX documents are cached on the file system, taking up increasing amount of space. Unless the \TeX 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 \TeX is also provided, see Figure 3.

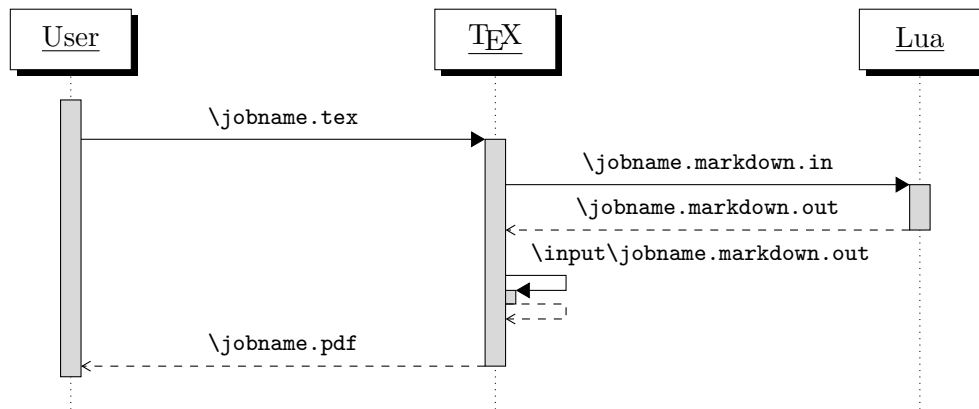


Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the \TeX interface

```
511
512 local HELP_STRING = [[
513 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
514 where OPTIONS are documented in the Lua interface section of the
515 technical Markdown package documentation.
516
517 When OUTPUT_FILE is unspecified, the result of the conversion will be
518 written to the standard output. When INPUT_FILE is also unspecified, the
519 result of the conversion will be read from the standard input.
520
521 Report bugs to: witiko@mail.muni.cz
522 Markdown package home page: <https://github.com/witiko/markdown>]]
523
```

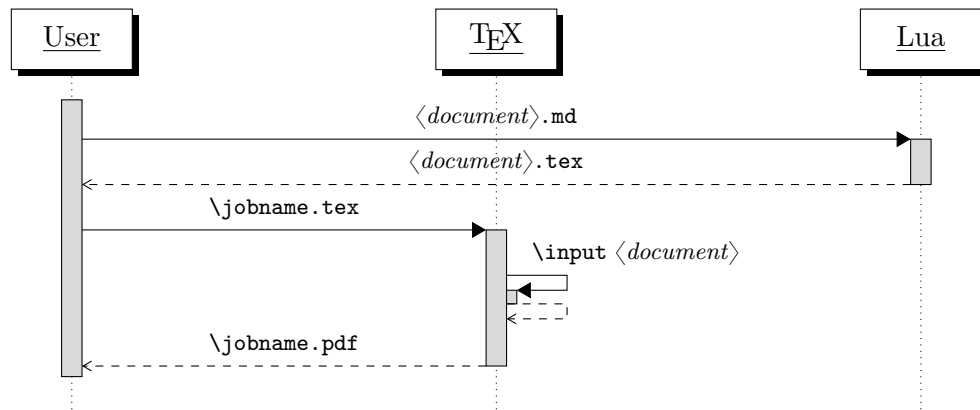


Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```

524 local VERSION_STRING = [[
525 markdown-cli.lua (Markdown) ]] .. metadata.version .. [[
526
527 Copyright (C) ]] .. table.concat(metadata.copyright,
528                                   "\nCopyright (C) ") .. [[
529
530 License: ]] .. metadata.license
531
532 local function warn(s)
533   io.stderr:write("Warning: " .. s .. "\n") end
534
535 local function error(s)
536   io.stderr:write("Error: " .. s .. "\n")
537   os.exit(1) end
538
539 local process_options = true
540 local options = {}
541 local input_filename
542 local output_filename
543 for i = 1, #arg do
544   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.

```

545     if arg[i] == "--" then
546       process_options = false
547       goto continue

```

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

```
548     elseif arg[i]:match("=") then
549         local key, value = arg[i]:match("(.)=(.*)")
```

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

```
550         local default_type = type(defaultOptions[key])
551         if default_type == "boolean" then
552             options[key] = (value == "true")
553         elseif default_type == "number" then
554             options[key] = tonumber(value)
555         elseif default_type == "table" then
556             options[key] = {}
557             for item in value:gmatch("[^,]+") do
558                 table.insert(options[key], item)
559             end
560         else
561             if default_type ~= "string" then
562                 if default_type == "nil" then
563                     warn('Option "' .. key .. '" not recognized.')
564                 else
565                     warn('Option "' .. key .. '" type not recognized, please file ' ..
566                         'a report to the package maintainer.')
567                 end
568                 warn('Parsing the ' .. 'value "' .. value .. '" of option "' ..
569                     key .. '" as a string.')
570             end
571             options[key] = value
572         end
573         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.

```
574     elseif arg[i] == "--help" or arg[i] == "-h" then
575         print(HELP_STRING)
576         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.

```
577     elseif arg[i] == "--version" or arg[i] == "-v" then
578         print(VERSION_STRING)
579         os.exit()
580     end
```

```
581 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 T_EX document.

```
582 if input_filename == nil then
```

```
583     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 T_EX document that will result from the conversion.

```
584 elseif output_filename == nil then
```

```
585     output_filename = arg[i]
```

```
586 else
```

```
587     error('Unexpected argument: "' .. arg[i] .. '".')
```

```
588 end
```

```
589 ::continue::
```

```
590 end
```

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

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

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

```
\input hello
```

2.2 Plain T_EX Interface

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

```
591 \def\markdownLastModified{((LASTMODIFIED))}%
```

```
592 \def\markdownVersion{((VERSION))}%
```

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

```
\input markdown
```

It is expected that the special plain T_EX characters have the expected category codes, when `\input`ting the file.

2.2.1 Typesetting Markdown

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

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

```
593 \let\markdownBegin\relax
```

```
594 \let\markdownEnd\relax
```

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

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

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of T_EX [4, p. 46]. As a corollary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```
595 \let\markdownInput\relax
```

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

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

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

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.

```
596 \ExplSyntaxOn
```

```
597 \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.

```
598 \prop_new:N \g_@@_plain_tex_option_types_prop
```

```
599 \prop_new:N \g_@@_default_plain_tex_options_prop
```

```
600 \tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
```

```
601 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
```

```
602 \cs_new:Nn
```

```
603   \@@_add_plain_tex_option:nnn
```

```
604   {
```

```
605     \@@_add_option:Vnnn
```

```
606     \c_@@_option_layer_plain_tex_tl
```

```
607     { #1 }
```

```
608     { #2 }
```

```
609     { #3 }
```

```
610   }
```

2.2.2.1 Finalizing and Freezing the Cache The `\markdownOptionFinalizeCache` option corresponds to the Lua interface `finalizeCache` option, which creates an output file `\markdownOptionFrozenCacheFileName` (frozen cache) that contains a

mapping between an enumeration of the markdown documents in the plain T_EX document and their auxiliary files cached in the `cacheDir` directory.

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

```
611 \@@_add_plain_tex_option:nnn
612   { frozenCache }
613   { boolean }
614   { 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 `\markdownOptionFinalizeCache` option.
4. Typeset the plain T_EX document to populate and finalize the cache.
5. Enable the `\markdownOptionFrozenCache` option.
6. Publish the source code of the plain T_EX document and the `cacheDir` directory.

2.2.2.2 File and Directory Names The `\markdownOptionHelperScriptFileName` macro sets the filename of the helper Lua script file that is created during the conversion from markdown to plain T_EX in T_EX 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_EX engines tend to put quotation marks around `\jobname`, when it contains spaces.

```
615 \@@_add_plain_tex_option:nnn
616   { helperScriptFileName }
617   { path }
618   { \jobname.markdown.lua }
```

The `\markdownOptionHelperScriptFileName` 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.

```
619 \str_new:N
620   \g_luabridge_helper_script_filename_str
621 \tl_gset:Nn
622   \g_luabridge_helper_script_filename_str
623   { \markdownOptionHelperScriptFileName }
```

The `\markdownOptionInputTempFileName` macro sets the filename of the temporary input file that is created during the buffering of markdown text from a T_EX source.

It defaults to `\jobname.markdown.in`. The same limitations as in the case of the `\markdownOptionHelperScriptFileName` macro apply here.

```
624 \@@_add_plain_tex_option:nnn
625   { inputTempFileName }
626   { path }
627   { \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_EX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.out`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
628 \@@_add_plain_tex_option:nnn
629   { outputTempFileName }
630   { path }
631   { \jobname.markdown.out }
```

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

```
632 \str_new:N
633   \g_luabridge_standard_output_filename_str
634 \tl_gset:Nn
635   \g_luabridge_standard_output_filename_str
636   { \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 T_EX in `\markdownMode` other than 2. It defaults to `\jobname.markdown.err`. The same limitations apply here as in the case of the `\markdownOptionHelperScriptFileName` macro.

```
637 \@@_add_plain_tex_option:nnn
638   { errorTempFileName }
639   { path }
640   { \jobname.markdown.err }
```

The `\markdownOptionErrorTempFileName` 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.

```
641 \str_new:N
642   \g_luabridge_error_output_filename_str
643 \tl_gset:Nn
644   \g_luabridge_error_output_filename_str
645   { \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 T_EX 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 `\markdownOptionHelperScriptFileName` macro.

```
646 \@@_add_plain_tex_option:nnn
647 { outputDir }
648 { path }
649 { . }
```

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 `\markdownOptionHelperScriptFileName` macro.

```
650 \cs_new:Nn \@@_plain_tex_define_option_commands:
651 {
652   \seq_map_inline:Nn
653     \g_@@_option_layers_seq
654     {
655       \seq_map_inline:cn
656         { g_@@_ ##1 _options_seq }
657         {
658           \@@_plain_tex_define_option_command:n
659             { ####1 }
660         }
661     }
662 }
663 \cs_new:Nn \@@_plain_tex_define_option_command:n
664 {
665   \@@_get_default_option_value:nN
666     { #1 }
667     \l_tmpa_tl
668   \@@_set_option_value:nV
669     { #1 }
670     \l_tmpa_tl
671 }
672 \cs_new:Nn
673   \@@_set_option_value:nn
674   {
675     \@@_define_option:n
676       { #1 }
677     \@@_get_option_type:nN
678       { #1 }
```

```

679     \l_tmpa_tl
680 \str_if_eq:NNTF
681     \c_@@_option_type_counter_tl
682     \l_tmpa_tl
683     {
684         \@@_option_tl_to_csname:nN
685         { #1 }
686         \l_tmpa_tl
687         \int_gset:cn
688         { \l_tmpa_tl }
689         { #2 }
690     }
691     {
692         \@@_option_tl_to_csname:nN
693         { #1 }
694         \l_tmpa_tl
695         \cs_set:cpn
696         { \l_tmpa_tl }
697         { #2 }
698     }
699 }
700 \cs_generate_variant:Nn
701     \@@_set_option_value:nn
702     { nV }
703 \cs_new:Nn
704     \@@_define_option:n
705     {
706         \@@_option_tl_to_csname:nN
707         { #1 }
708         \l_tmpa_tl
709         \cs_if_free:cT
710         { \l_tmpa_tl }
711         {
712             \@@_get_option_type:nN
713             { #1 }
714             \l_tmpb_tl
715             \str_if_eq:NNT
716             \c_@@_option_type_counter_tl
717             \l_tmpb_tl
718             {
719                 \@@_option_tl_to_csname:nN
720                 { #1 }
721                 \l_tmpa_tl
722                 \int_new:c
723                 { \l_tmpa_tl }
724             }
725         }

```

```

726 }
727 \@@_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_EX package documentation using the Doc L^AT_EX package [5] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```

728 \seq_put_right:Nn
729   \g_@@_plain_tex_options_seq
730   { stripPercentSigns }
731 \prop_put:Nnn
732   \g_@@_plain_tex_option_types_prop
733   { stripPercentSigns }
734   { boolean }
735 \prop_put:Nnx
736   \g_@@_default_plain_tex_options_prop
737   { stripPercentSigns }
738   { false }
739 \ExplSyntaxOff

```

2.2.3 Token Renderers

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

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

```

740 \ExplSyntaxOn
741 \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.

```

742 \prop_new:N \g_@@_renderer_arities_prop
743 \ExplSyntaxOff

```

2.2.3.1 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.

```

744 \def\markdownRendererTickedBox{%
745   \markdownRendererTickedBoxPrototype}%
746 \ExplSyntaxOn
747 \seq_put_right:Nn
748   \g_@@_renderers_seq
749   { tickedBox }
750 \prop_put:Nnn
751   \g_@@_renderer_arities_prop
752   { tickedBox }
753   { 0 }
754 \ExplSyntaxOff
755 \def\markdownRendererHalfTickedBox{%
756   \markdownRendererHalfTickedBoxPrototype}%
757 \ExplSyntaxOn
758 \seq_put_right:Nn
759   \g_@@_renderers_seq
760   { halfTickedBox }
761 \prop_put:Nnn
762   \g_@@_renderer_arities_prop
763   { halfTickedBox }
764   { 0 }
765 \ExplSyntaxOff
766 \def\markdownRendererUntickedBox{%
767   \markdownRendererUntickedBoxPrototype}%
768 \ExplSyntaxOn
769 \seq_put_right:Nn
770   \g_@@_renderers_seq
771   { untickedBox }
772 \prop_put:Nnn
773   \g_@@_renderer_arities_prop
774   { untickedBox }
775   { 0 }
776 \ExplSyntaxOff

```

2.2.3.2 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.

```

777 \def\markdownRendererDocumentBegin{%
778   \markdownRendererDocumentBeginPrototype}%
779 \ExplSyntaxOn
780 \seq_put_right:Nn

```

```

781 \g_@@_renderers_seq
782 { documentBegin }
783 \prop_put:Nnn
784 \g_@@_renderer_arities_prop
785 { documentBegin }
786 { 0 }
787 \ExplSyntaxOff
788 \def\markdownRendererDocumentEnd{%
789 \markdownRendererDocumentEndPrototype}%
790 \ExplSyntaxOn
791 \seq_put_right:Nn
792 \g_@@_renderers_seq
793 { documentEnd }
794 \prop_put:Nnn
795 \g_@@_renderer_arities_prop
796 { documentEnd }
797 { 0 }
798 \ExplSyntaxOff

```

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

```

799 \def\markdownRendererInterblockSeparator{%
800 \markdownRendererInterblockSeparatorPrototype}%
801 \ExplSyntaxOn
802 \seq_put_right:Nn
803 \g_@@_renderers_seq
804 { interblockSeparator }
805 \prop_put:Nnn
806 \g_@@_renderer_arities_prop
807 { interblockSeparator }
808 { 0 }
809 \ExplSyntaxOff

```

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

```

810 \def\markdownRendererLineBreak{%
811 \markdownRendererLineBreakPrototype}%
812 \ExplSyntaxOn
813 \seq_put_right:Nn
814 \g_@@_renderers_seq
815 { lineBreak }
816 \prop_put:Nnn
817 \g_@@_renderer_arities_prop
818 { lineBreak }

```

```

819 { 0 }
820 \ExplSyntaxOff

```

2.2.3.5 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.

```

821 \def\markdownRendererEllipsis{%
822   \markdownRendererEllipsisPrototype}%
823 \ExplSyntaxOn
824 \seq_put_right:Nn
825   \g_@@_renderers_seq
826   { ellipsis }
827 \prop_put:Nnn
828   \g_@@_renderer_arities_prop
829   { ellipsis }
830   { 0 }
831 \ExplSyntaxOff

```

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

```

832 \def\markdownRendererNbsp{%
833   \markdownRendererNbspPrototype}%
834 \ExplSyntaxOn
835 \seq_put_right:Nn
836   \g_@@_renderers_seq
837   { nbsp }
838 \prop_put:Nnn
839   \g_@@_renderer_arities_prop
840   { nbsp }
841   { 0 }
842 \ExplSyntaxOff

```

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

```

843 \def\markdownRendererLeftBrace{%
844   \markdownRendererLeftBracePrototype}%
845 \ExplSyntaxOn
846 \seq_put_right:Nn
847   \g_@@_renderers_seq
848   { leftBrace }
849 \prop_put:Nnn
850   \g_@@_renderer_arities_prop
851   { leftBrace }

```

```

852 { 0 }
853 \ExplSyntaxOff
854 \def\markdownRendererRightBrace{%
855 \markdownRendererRightBracePrototype}%
856 \ExplSyntaxOn
857 \seq_put_right:Nn
858 \g_@@_renderers_seq
859 { rightBrace }
860 \prop_put:Nnn
861 \g_@@_renderer_arities_prop
862 { rightBrace }
863 { 0 }
864 \ExplSyntaxOff
865 \def\markdownRendererDollarSign{%
866 \markdownRendererDollarSignPrototype}%
867 \ExplSyntaxOn
868 \seq_put_right:Nn
869 \g_@@_renderers_seq
870 { dollarSign }
871 \prop_put:Nnn
872 \g_@@_renderer_arities_prop
873 { dollarSign }
874 { 0 }
875 \ExplSyntaxOff
876 \def\markdownRendererPercentSign{%
877 \markdownRendererPercentSignPrototype}%
878 \ExplSyntaxOn
879 \seq_put_right:Nn
880 \g_@@_renderers_seq
881 { percentSign }
882 \prop_put:Nnn
883 \g_@@_renderer_arities_prop
884 { percentSign }
885 { 0 }
886 \ExplSyntaxOff
887 \def\markdownRendererAmpersand{%
888 \markdownRendererAmpersandPrototype}%
889 \ExplSyntaxOn
890 \seq_put_right:Nn
891 \g_@@_renderers_seq
892 { ampersand }
893 \prop_put:Nnn
894 \g_@@_renderer_arities_prop
895 { ampersand }
896 { 0 }
897 \ExplSyntaxOff
898 \def\markdownRendererUnderscore{%

```



```

899 \markdownRendererUnderscorePrototype}%
900 \ExplSyntaxOn
901 \seq_put_right:Nn
902   \g_@@_renderers_seq
903   { underscore }
904 \prop_put:Nnn
905   \g_@@_renderer_arities_prop
906   { underscore }
907   { 0 }
908 \ExplSyntaxOff
909 \def\markdownRendererHash{%
910   \markdownRendererHashPrototype}%
911 \ExplSyntaxOn
912 \seq_put_right:Nn
913   \g_@@_renderers_seq
914   { hash }
915 \prop_put:Nnn
916   \g_@@_renderer_arities_prop
917   { hash }
918   { 0 }
919 \ExplSyntaxOff
920 \def\markdownRendererCircumflex{%
921   \markdownRendererCircumflexPrototype}%
922 \ExplSyntaxOn
923 \seq_put_right:Nn
924   \g_@@_renderers_seq
925   { circumflex }
926 \prop_put:Nnn
927   \g_@@_renderer_arities_prop
928   { circumflex }
929   { 0 }
930 \ExplSyntaxOff
931 \def\markdownRendererBackslash{%
932   \markdownRendererBackslashPrototype}%
933 \ExplSyntaxOn
934 \seq_put_right:Nn
935   \g_@@_renderers_seq
936   { backslash }
937 \prop_put:Nnn
938   \g_@@_renderer_arities_prop
939   { backslash }
940   { 0 }
941 \ExplSyntaxOff
942 \def\markdownRendererTilde{%
943   \markdownRendererTildePrototype}%
944 \ExplSyntaxOn
945 \seq_put_right:Nn

```

```

946 \g_@@_renderers_seq
947 { tilde }
948 \prop_put:Nnn
949 \g_@@_renderer_arities_prop
950 { tilde }
951 { 0 }
952 \ExplSyntaxOff
953 \def\markdownRendererPipe{%
954 \markdownRendererPipePrototype}%
955 \ExplSyntaxOn
956 \seq_put_right:Nn
957 \g_@@_renderers_seq
958 { pipe }
959 \prop_put:Nnn
960 \g_@@_renderer_arities_prop
961 { pipe }
962 { 0 }
963 \ExplSyntaxOff

```

2.2.3.8 Code Span Renderer The `\markdownRendererCodeSpan` macro represents inlined code span in the input text. It receives a single argument that corresponds to the inlined code span.

```

964 \def\markdownRendererCodeSpan{%
965 \markdownRendererCodeSpanPrototype}%
966 \ExplSyntaxOn
967 \seq_put_right:Nn
968 \g_@@_renderers_seq
969 { codeSpan }
970 \prop_put:Nnn
971 \g_@@_renderer_arities_prop
972 { codeSpan }
973 { 1 }
974 \ExplSyntaxOff

```

2.2.3.9 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.

```

975 \def\markdownRendererLink{%
976 \markdownRendererLinkPrototype}%
977 \ExplSyntaxOn
978 \seq_put_right:Nn
979 \g_@@_renderers_seq
980 { link }
981 \prop_put:Nnn
982 \g_@@_renderer_arities_prop

```

```

983 { link }
984 { 4 }
985 \ExplSyntaxOff

```

2.2.3.10 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.

```

986 \def\markdownRendererImage{%
987   \markdownRendererImagePrototype}%
988 \ExplSyntaxOn
989 \seq_put_right:Nn
990   \g_@@_renderers_seq
991   { image }
992 \prop_put:Nnn
993   \g_@@_renderer_arities_prop
994   { image }
995   { 4 }
996 \ExplSyntaxOff

```

2.2.3.11 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.

```

997 \def\markdownRendererContentBlock{%
998   \markdownRendererContentBlockPrototype}%
999 \ExplSyntaxOn
1000 \seq_put_right:Nn
1001   \g_@@_renderers_seq
1002   { contentBlock }
1003 \prop_put:Nnn
1004   \g_@@_renderer_arities_prop
1005   { contentBlock }
1006   { 4 }
1007 \ExplSyntaxOff

```

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

```

1008 \def\markdownRendererContentBlockOnlineImage{%
1009   \markdownRendererContentBlockOnlineImagePrototype}%
1010 \ExplSyntaxOn
1011 \seq_put_right:Nn

```

```

1012 \g_@@_renderers_seq
1013 { contentBlockOnlineImage }
1014 \prop_put:Nnn
1015 \g_@@_renderer_arities_prop
1016 { contentBlockOnlineImage }
1017 { 4 }
1018 \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 kpathsea⁷ contains a record (k, v) , then a non-online-image content block with the filename extension s , $s:\text{lower}() = k$ is considered to be in a known programming language v . The macro receives five arguments: the local file name extension s cast to the lower case, the language v , the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

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

```

1019 \def\markdownRendererContentBlockCode{%
1020 \markdownRendererContentBlockCodePrototype}%
1021 \ExplSyntaxOn
1022 \seq_put_right:Nn
1023 \g_@@_renderers_seq
1024 { contentBlockCode }
1025 \prop_put:Nnn
1026 \g_@@_renderer_arities_prop
1027 { contentBlockCode }
1028 { 5 }
1029 \ExplSyntaxOff

```

2.2.3.12 Bullet List Renderers The `\markdownRendererU1Begin` 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.

```

1030 \def\markdownRendererU1Begin{%
1031 \markdownRendererU1BeginPrototype}%
1032 \ExplSyntaxOn
1033 \seq_put_right:Nn
1034 \g_@@_renderers_seq

```

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

```

1035 { ulBegin }
1036 \prop_put:Nnn
1037 \g_@@_renderer_arities_prop
1038 { ulBegin }
1039 { 0 }
1040 \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.

```

1041 \def\markdownRendererUlBeginTight{%
1042   \markdownRendererUlBeginTightPrototype}%
1043 \ExplSyntaxOn
1044 \seq_put_right:Nn
1045   \g_@@_renderers_seq
1046   { ulBeginTight }
1047 \prop_put:Nnn
1048   \g_@@_renderer_arities_prop
1049   { ulBeginTight }
1050   { 0 }
1051 \ExplSyntaxOff

```

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

```

1052 \def\markdownRendererUlItem{%
1053   \markdownRendererUlItemPrototype}%
1054 \ExplSyntaxOn
1055 \seq_put_right:Nn
1056   \g_@@_renderers_seq
1057   { ulItem }
1058 \prop_put:Nnn
1059   \g_@@_renderer_arities_prop
1060   { ulItem }
1061   { 0 }
1062 \ExplSyntaxOff

```

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

```

1063 \def\markdownRendererUlItemEnd{%
1064   \markdownRendererUlItemEndPrototype}%
1065 \ExplSyntaxOn
1066 \seq_put_right:Nn
1067   \g_@@_renderers_seq
1068   { ulItemEnd }
1069 \prop_put:Nnn

```

```

1070 \g_@@_renderer_arities_prop
1071 { ulItemEnd }
1072 { 0 }
1073 \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.

```

1074 \def\markdownRendererUEnd{%
1075 \markdownRendererUEndPrototype}%
1076 \ExplSyntaxOn
1077 \seq_put_right:Nn
1078 \g_@@_renderers_seq
1079 { ulEnd }
1080 \prop_put:Nnn
1081 \g_@@_renderer_arities_prop
1082 { ulEnd }
1083 { 0 }
1084 \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.

```

1085 \def\markdownRendererUEndTight{%
1086 \markdownRendererUEndTightPrototype}%
1087 \ExplSyntaxOn
1088 \seq_put_right:Nn
1089 \g_@@_renderers_seq
1090 { ulEndTight }
1091 \prop_put:Nnn
1092 \g_@@_renderer_arities_prop
1093 { ulEndTight }
1094 { 0 }
1095 \ExplSyntaxOff

```

2.2.3.13 Ordered List Renderers The `\markdownRendererO1Begin` macro represents the beginning of an ordered list that contains an item with several paragraphs of text (the list is not tight). This macro will only be produced, when the `fancyLists` option is disabled. The macro receives no arguments.

```

1096 \def\markdownRendererO1Begin{%
1097 \markdownRendererO1BeginPrototype}%
1098 \ExplSyntaxOn
1099 \seq_put_right:Nn
1100 \g_@@_renderers_seq

```

```

1101 { olBegin }
1102 \prop_put:Nnn
1103 \g_@@_renderer_arities_prop
1104 { olBegin }
1105 { 0 }
1106 \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.

```

1107 \def\markdownRendererOlBeginTight{%
1108   \markdownRendererOlBeginTightPrototype}%
1109 \ExplSyntaxOn
1110 \seq_put_right:Nn
1111   \g_@@_renderers_seq
1112   { olBeginTight }
1113 \prop_put:Nnn
1114   \g_@@_renderer_arities_prop
1115   { olBeginTight }
1116   { 0 }
1117 \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`).

```

1118 \def\markdownRendererFancyOlBegin{%
1119   \markdownRendererFancyOlBeginPrototype}%
1120 \ExplSyntaxOn
1121 \seq_put_right:Nn
1122   \g_@@_renderers_seq
1123   { fancyOlBegin }
1124 \prop_put:Nnn
1125   \g_@@_renderer_arities_prop
1126   { fancyOlBegin }
1127   { 2 }
1128 \ExplSyntaxOff

```

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

labels, and the style of delimiters between list item labels and texts. See the `\markdownRendererFancyOlBegin` macro for the valid style values.

```

1129 \def\markdownRendererFancyOlBeginTight{%
1130   \markdownRendererFancyOlBeginTightPrototype}%
1131 \ExplSyntaxOn
1132 \seq_put_right:Nn
1133   \g_@@_renderers_seq
1134   { fancyOlBeginTight }
1135 \prop_put:Nnn
1136   \g_@@_renderer_arities_prop
1137   { fancyOlBeginTight }
1138   { 2 }
1139 \ExplSyntaxOff

```

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

```

1140 \def\markdownRendererOlItem{%
1141   \markdownRendererOlItemPrototype}%
1142 \ExplSyntaxOn
1143 \seq_put_right:Nn
1144   \g_@@_renderers_seq
1145   { olItem }
1146 \prop_put:Nnn
1147   \g_@@_renderer_arities_prop
1148   { olItem }
1149   { 0 }
1150 \ExplSyntaxOff

```

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

```

1151 \def\markdownRendererOlItemEnd{%
1152   \markdownRendererOlItemEndPrototype}%
1153 \ExplSyntaxOn
1154 \seq_put_right:Nn
1155   \g_@@_renderers_seq
1156   { olItemEnd }
1157 \prop_put:Nnn
1158   \g_@@_renderer_arities_prop
1159   { olItemEnd }
1160   { 0 }
1161 \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.

```

1162 \def\markdownRendererOItemWithNumber{%
1163   \markdownRendererOItemWithNumberPrototype}%
1164 \ExplSyntaxOn
1165 \seq_put_right:Nn
1166   \g_@@_renderers_seq
1167   { oItemWithNumber }
1168 \prop_put:Nnn
1169   \g_@@_renderer_arities_prop
1170   { oItemWithNumber }
1171   { 1 }
1172 \ExplSyntaxOff

```

The `\markdownRendererFancyOItem` 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.

```

1173 \def\markdownRendererFancyOItem{%
1174   \markdownRendererFancyOItemPrototype}%
1175 \ExplSyntaxOn
1176 \seq_put_right:Nn
1177   \g_@@_renderers_seq
1178   { fancyOItem }
1179 \prop_put:Nnn
1180   \g_@@_renderer_arities_prop
1181   { fancyOItem }
1182   { 0 }
1183 \ExplSyntaxOff

```

The `\markdownRendererFancyOItemEnd` 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.

```

1184 \def\markdownRendererFancyOItemEnd{%
1185   \markdownRendererFancyOItemEndPrototype}%
1186 \ExplSyntaxOn
1187 \seq_put_right:Nn
1188   \g_@@_renderers_seq
1189   { fancyOItemEnd }
1190 \prop_put:Nnn
1191   \g_@@_renderer_arities_prop
1192   { fancyOItemEnd }
1193   { 0 }
1194 \ExplSyntaxOff

```

The `\markdownRendererFancyOItemWithNumber` 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.

```

1195 \def\markdownRendererFancyOItemWithNumber{%
1196   \markdownRendererFancyOItemWithNumberPrototype}%
1197 \ExplSyntaxOn
1198 \seq_put_right:Nn
1199   \g_@@_renderers_seq
1200   { fancyOItemWithNumber }
1201 \prop_put:Nnn
1202   \g_@@_renderer_arities_prop
1203   { fancyOItemWithNumber }
1204   { 1 }
1205 \ExplSyntaxOff

```

The `\markdownRendererOItemEnd` 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.

```

1206 \def\markdownRendererOItemEnd{%
1207   \markdownRendererOItemEndPrototype}%
1208 \ExplSyntaxOn
1209 \seq_put_right:Nn
1210   \g_@@_renderers_seq
1211   { olEnd }
1212 \prop_put:Nnn
1213   \g_@@_renderer_arities_prop
1214   { olEnd }
1215   { 0 }
1216 \ExplSyntaxOff

```

The `\markdownRendererOItemEndTight` 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.

```

1217 \def\markdownRendererOItemEndTight{%
1218   \markdownRendererOItemEndTightPrototype}%
1219 \ExplSyntaxOn
1220 \seq_put_right:Nn
1221   \g_@@_renderers_seq
1222   { olEndTight }
1223 \prop_put:Nnn
1224   \g_@@_renderer_arities_prop
1225   { olEndTight }
1226   { 0 }
1227 \ExplSyntaxOff

```

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

```

1228 \def\markdownRendererFancyOlEnd{%
1229   \markdownRendererFancyOlEndPrototype}%
1230 \ExplSyntaxOn
1231 \seq_put_right:Nn
1232   \g_@@_renderers_seq
1233   { fancyOlEnd }
1234 \prop_put:Nnn
1235   \g_@@_renderer_arities_prop
1236   { fancyOlEnd }
1237   { 0 }
1238 \ExplSyntaxOff

```

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

```

1239 \def\markdownRendererFancyOlEndTight{%
1240   \markdownRendererFancyOlEndTightPrototype}%
1241 \ExplSyntaxOn
1242 \seq_put_right:Nn
1243   \g_@@_renderers_seq
1244   { fancyOlEndTight }
1245 \prop_put:Nnn
1246   \g_@@_renderer_arities_prop
1247   { fancyOlEndTight }
1248   { 0 }
1249 \ExplSyntaxOff

```

2.2.3.14 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.

```

1250 \def\markdownRendererDlBegin{%
1251   \markdownRendererDlBeginPrototype}%
1252 \ExplSyntaxOn
1253 \seq_put_right:Nn
1254   \g_@@_renderers_seq
1255   { dlBegin }
1256 \prop_put:Nnn

```

```

1257 \g_@@_renderer_arities_prop
1258 { dlBegin }
1259 { 0 }
1260 \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.

```

1261 \def\markdownRendererDlBeginTight{%
1262 \markdownRendererDlBeginTightPrototype}%
1263 \ExplSyntaxOn
1264 \seq_put_right:Nn
1265 \g_@@_renderers_seq
1266 { dlBeginTight }
1267 \prop_put:Nnn
1268 \g_@@_renderer_arities_prop
1269 { dlBeginTight }
1270 { 0 }
1271 \ExplSyntaxOff

```

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

```

1272 \def\markdownRendererDlItem{%
1273 \markdownRendererDlItemPrototype}%
1274 \ExplSyntaxOn
1275 \seq_put_right:Nn
1276 \g_@@_renderers_seq
1277 { dlItem }
1278 \prop_put:Nnn
1279 \g_@@_renderer_arities_prop
1280 { dlItem }
1281 { 1 }
1282 \ExplSyntaxOff

```

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

```

1283 \def\markdownRendererDlItemEnd{%
1284 \markdownRendererDlItemEndPrototype}%
1285 \ExplSyntaxOn
1286 \seq_put_right:Nn
1287 \g_@@_renderers_seq
1288 { dlItemEnd }
1289 \prop_put:Nnn
1290 \g_@@_renderer_arities_prop
1291 { dlItemEnd }

```

```

1292 { 0 }
1293 \ExplSyntaxOff

```

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

```

1294 \def\markdownRendererDlDefinitionBegin{%
1295   \markdownRendererDlDefinitionBeginPrototype}%
1296 \ExplSyntaxOn
1297 \seq_put_right:Nn
1298   \g_@@_renderers_seq
1299   { dlDefinitionBegin }
1300 \prop_put:Nnn
1301   \g_@@_renderer_arities_prop
1302   { dlDefinitionBegin }
1303   { 0 }
1304 \ExplSyntaxOff

```

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

```

1305 \def\markdownRendererDlDefinitionEnd{%
1306   \markdownRendererDlDefinitionEndPrototype}%
1307 \ExplSyntaxOn
1308 \seq_put_right:Nn
1309   \g_@@_renderers_seq
1310   { dlDefinitionEnd }
1311 \prop_put:Nnn
1312   \g_@@_renderer_arities_prop
1313   { dlDefinitionEnd }
1314   { 0 }
1315 \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.

```

1316 \def\markdownRendererDlEnd{%
1317   \markdownRendererDlEndPrototype}%
1318 \ExplSyntaxOn
1319 \seq_put_right:Nn
1320   \g_@@_renderers_seq
1321   { dlEnd }
1322 \prop_put:Nnn
1323   \g_@@_renderer_arities_prop
1324   { dlEnd }
1325   { 0 }
1326 \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.

```

1327 \def\markdownRendererDlEndTight{%
1328   \markdownRendererDlEndTightPrototype}%
1329 \ExplSyntaxOn
1330 \seq_put_right:Nn
1331   \g_@@_renderers_seq
1332   { dlEndTight }
1333 \prop_put:Nnn
1334   \g_@@_renderer_arities_prop
1335   { dlEndTight }
1336   { 0 }
1337 \ExplSyntaxOff

```

2.2.3.15 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.

```

1338 \def\markdownRendererEmphasis{%
1339   \markdownRendererEmphasisPrototype}%
1340 \ExplSyntaxOn
1341 \seq_put_right:Nn
1342   \g_@@_renderers_seq
1343   { emphasis }
1344 \prop_put:Nnn
1345   \g_@@_renderer_arities_prop
1346   { emphasis }
1347   { 1 }
1348 \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.

```

1349 \def\markdownRendererStrongEmphasis{%
1350   \markdownRendererStrongEmphasisPrototype}%
1351 \ExplSyntaxOn
1352 \seq_put_right:Nn
1353   \g_@@_renderers_seq
1354   { strongEmphasis }
1355 \prop_put:Nnn
1356   \g_@@_renderer_arities_prop
1357   { strongEmphasis }
1358   { 1 }
1359 \ExplSyntaxOff

```

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

```

1360 \def\markdownRendererBlockQuoteBegin{%
1361   \markdownRendererBlockQuoteBeginPrototype}%
1362 \ExplSyntaxOn
1363 \seq_put_right:Nn
1364   \g_@@_renderers_seq
1365   { blockQuoteBegin }
1366 \prop_put:Nnn
1367   \g_@@_renderer_arities_prop
1368   { blockQuoteBegin }
1369   { 0 }
1370 \ExplSyntaxOff

```

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

```

1371 \def\markdownRendererBlockQuoteEnd{%
1372   \markdownRendererBlockQuoteEndPrototype}%
1373 \ExplSyntaxOn
1374 \seq_put_right:Nn
1375   \g_@@_renderers_seq
1376   { blockQuoteEnd }
1377 \prop_put:Nnn
1378   \g_@@_renderer_arities_prop
1379   { blockQuoteEnd }
1380   { 0 }
1381 \ExplSyntaxOff

```

2.2.3.17 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.

```

1382 \def\markdownRendererInputVerbatim{%
1383   \markdownRendererInputVerbatimPrototype}%
1384 \ExplSyntaxOn
1385 \seq_put_right:Nn
1386   \g_@@_renderers_seq
1387   { inputVerbatim }
1388 \prop_put:Nnn
1389   \g_@@_renderer_arities_prop
1390   { inputVerbatim }
1391   { 1 }
1392 \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.

```

1393 \def\markdownRendererInputFencedCode{%
1394   \markdownRendererInputFencedCodePrototype}%
1395 \ExplSyntaxOn
1396 \seq_put_right:Nn
1397   \g_@@_renderers_seq
1398   { inputFencedCode }
1399 \prop_put:Nnn
1400   \g_@@_renderer_arities_prop
1401   { inputFencedCode }
1402   { 2 }
1403 \ExplSyntaxOff

```

2.2.3.18 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.

```

1404 \def\markdownRendererJekyllDataBegin{%
1405   \markdownRendererJekyllDataBeginPrototype}%
1406 \ExplSyntaxOn
1407 \seq_put_right:Nn
1408   \g_@@_renderers_seq
1409   { jekyllDataBegin }
1410 \prop_put:Nnn
1411   \g_@@_renderer_arities_prop
1412   { jekyllDataBegin }
1413   { 0 }
1414 \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.

```

1415 \def\markdownRendererJekyllDataEnd{%
1416   \markdownRendererJekyllDataEndPrototype}%
1417 \ExplSyntaxOn
1418 \seq_put_right:Nn
1419   \g_@@_renderers_seq
1420   { jekyllDataEnd }
1421 \prop_put:Nnn
1422   \g_@@_renderer_arities_prop
1423   { jekyllDataEnd }
1424   { 0 }
1425 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the

`jeekyllData` 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.

```

1426 \def\markdownRendererJekyllDataMappingBegin{%
1427   \markdownRendererJekyllDataMappingBeginPrototype}%
1428 \ExplSyntaxOn
1429 \seq_put_right:Nn
1430   \g_@@_renderers_seq
1431   { jeekyllDataMappingBegin }
1432 \prop_put:Nnn
1433   \g_@@_renderer_arities_prop
1434   { jeekyllDataMappingBegin }
1435   { 2 }
1436 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataMappingEnd` macro represents the end of a mapping in a YAML document. This macro will only be produced when the `jeekyllData` option is enabled. The macro receives no arguments.

```

1437 \def\markdownRendererJekyllDataMappingEnd{%
1438   \markdownRendererJekyllDataMappingEndPrototype}%
1439 \ExplSyntaxOn
1440 \seq_put_right:Nn
1441   \g_@@_renderers_seq
1442   { jeekyllDataMappingEnd }
1443 \prop_put:Nnn
1444   \g_@@_renderer_arities_prop
1445   { jeekyllDataMappingEnd }
1446   { 0 }
1447 \ExplSyntaxOff

```

The `\markdownRendererJekyllDataSequenceBegin` macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the `jeekyllData` 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.

```

1448 \def\markdownRendererJekyllDataSequenceBegin{%
1449   \markdownRendererJekyllDataSequenceBeginPrototype}%
1450 \ExplSyntaxOn
1451 \seq_put_right:Nn
1452   \g_@@_renderers_seq
1453   { jeekyllDataSequenceBegin }
1454 \prop_put:Nnn
1455   \g_@@_renderer_arities_prop
1456   { jeekyllDataSequenceBegin }
1457   { 2 }
1458 \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.

```
1459 \def\markdownRendererJekyllDataSequenceEnd{%
1460   \markdownRendererJekyllDataSequenceEndPrototype}%
1461 \ExplSyntaxOn
1462 \seq_put_right:Nn
1463   \g_@@_renderers_seq
1464   { jekyllDataSequenceEnd }
1465 \prop_put:Nnn
1466   \g_@@_renderer_arities_prop
1467   { jekyllDataSequenceEnd }
1468   { 0 }
1469 \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.

```
1470 \def\markdownRendererJekyllDataBoolean{%
1471   \markdownRendererJekyllDataBooleanPrototype}%
1472 \ExplSyntaxOn
1473 \seq_put_right:Nn
1474   \g_@@_renderers_seq
1475   { jekyllDataBoolean }
1476 \prop_put:Nnn
1477   \g_@@_renderer_arities_prop
1478   { jekyllDataBoolean }
1479   { 2 }
1480 \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.

```
1481 \def\markdownRendererJekyllDataNumber{%
1482   \markdownRendererJekyllDataNumberPrototype}%
1483 \ExplSyntaxOn
1484 \seq_put_right:Nn
1485   \g_@@_renderers_seq
1486   { jekyllDataNumber }
1487 \prop_put:Nnn
1488   \g_@@_renderer_arities_prop
```

```

1489 { jekyllDataNumber }
1490 { 2 }
1491 \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.

```

1492 \def\markdownRendererJekyllDataString{%
1493   \markdownRendererJekyllDataStringPrototype}%
1494 \ExplSyntaxOn
1495 \seq_put_right:Nn
1496   \g_@@_renderers_seq
1497   { jekyllDataString }
1498 \prop_put:Nnn
1499   \g_@@_renderer_arities_prop
1500   { jekyllDataString }
1501   { 2 }
1502 \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.

```

1503 \def\markdownRendererJekyllDataEmpty{%
1504   \markdownRendererJekyllDataEmptyPrototype}%
1505 \ExplSyntaxOn
1506 \seq_put_right:Nn
1507   \g_@@_renderers_seq
1508   { jekyllDataEmpty }
1509 \prop_put:Nnn
1510   \g_@@_renderer_arities_prop
1511   { jekyllDataEmpty }
1512   { 1 }
1513 \ExplSyntaxOff

```

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

```

1514 \def\markdownRendererHeadingOne{%
1515   \markdownRendererHeadingOnePrototype}%
1516 \ExplSyntaxOn
1517 \seq_put_right:Nn

```

```

1518 \g_@@_renderers_seq
1519 { headingOne }
1520 \prop_put:Nnn
1521 \g_@@_renderer_arities_prop
1522 { headingOne }
1523 { 1 }
1524 \ExplSyntaxOff

```

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

```

1525 \def\markdownRendererHeadingTwo{%
1526 \markdownRendererHeadingTwoPrototype}%
1527 \ExplSyntaxOn
1528 \seq_put_right:Nn
1529 \g_@@_renderers_seq
1530 { headingTwo }
1531 \prop_put:Nnn
1532 \g_@@_renderer_arities_prop
1533 { headingTwo }
1534 { 1 }
1535 \ExplSyntaxOff

```

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

```

1536 \def\markdownRendererHeadingThree{%
1537 \markdownRendererHeadingThreePrototype}%
1538 \ExplSyntaxOn
1539 \seq_put_right:Nn
1540 \g_@@_renderers_seq
1541 { headingThree }
1542 \prop_put:Nnn
1543 \g_@@_renderer_arities_prop
1544 { headingThree }
1545 { 1 }
1546 \ExplSyntaxOff

```

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

```

1547 \def\markdownRendererHeadingFour{%
1548 \markdownRendererHeadingFourPrototype}%
1549 \ExplSyntaxOn
1550 \seq_put_right:Nn
1551 \g_@@_renderers_seq
1552 { headingFour }
1553 \prop_put:Nnn
1554 \g_@@_renderer_arities_prop
1555 { headingFour }

```

```

1556 { 1 }
1557 \ExplSyntaxOff

```

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

```

1558 \def\markdownRendererHeadingFive{%
1559   \markdownRendererHeadingFivePrototype}%
1560 \ExplSyntaxOn
1561 \seq_put_right:Nn
1562   \g_@@_renderers_seq
1563   { headingFive }
1564 \prop_put:Nnn
1565   \g_@@_renderer_arities_prop
1566   { headingFive }
1567   { 1 }
1568 \ExplSyntaxOff

```

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

```

1569 \def\markdownRendererHeadingSix{%
1570   \markdownRendererHeadingSixPrototype}%
1571 \ExplSyntaxOn
1572 \seq_put_right:Nn
1573   \g_@@_renderers_seq
1574   { headingSix }
1575 \prop_put:Nnn
1576   \g_@@_renderer_arities_prop
1577   { headingSix }
1578   { 1 }
1579 \ExplSyntaxOff

```

2.2.3.20 Horizontal Rule Renderer The `\markdownRendererHorizontalRule` macro represents a horizontal rule. The macro receives no arguments.

```

1580 \def\markdownRendererHorizontalRule{%
1581   \markdownRendererHorizontalRulePrototype}%
1582 \ExplSyntaxOn
1583 \seq_put_right:Nn
1584   \g_@@_renderers_seq
1585   { horizontalRule }
1586 \prop_put:Nnn
1587   \g_@@_renderer_arities_prop
1588   { horizontalRule }
1589   { 0 }
1590 \ExplSyntaxOff

```

2.2.3.21 Footnote Renderer The `\markdownRendererFootnote` macro represents a footnote. This macro will only be produced, when the `footnotes` option is enabled. The macro receives a single argument that corresponds to the footnote text.

```

1591 \def\markdownRendererFootnote{%
1592   \markdownRendererFootnotePrototype}%
1593 \ExplSyntaxOn
1594 \seq_put_right:Nn
1595   \g_@@_renderers_seq
1596   { footnote }
1597 \prop_put:Nnn
1598   \g_@@_renderer_arities_prop
1599   { footnote }
1600   { 1 }
1601 \ExplSyntaxOff

```

2.2.3.22 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.

```

1602 \def\markdownRendererCite{%
1603   \markdownRendererCitePrototype}%
1604 \ExplSyntaxOn
1605 \seq_put_right:Nn
1606   \g_@@_renderers_seq
1607   { cite }
1608 \prop_put:Nnn
1609   \g_@@_renderer_arities_prop
1610   { cite }
1611   { 1 }
1612 \ExplSyntaxOff

```

2.2.3.23 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.

```

1613 \def\markdownRendererTextCite{%
1614   \markdownRendererTextCitePrototype}%
1615 \ExplSyntaxOn
1616 \seq_put_right:Nn
1617   \g_@@_renderers_seq

```

```

1618 { textCite }
1619 \prop_put:Nnn
1620 \g_@@_renderer_arities_prop
1621 { textCite }
1622 { 1 }
1623 \ExplSyntaxOff

```

2.2.3.24 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.

```

1624 \def\markdownRendererTable{%
1625   \markdownRendererTablePrototype}%
1626 \ExplSyntaxOn
1627 \seq_put_right:Nn
1628   \g_@@_renderers_seq
1629   { table }
1630 \prop_put:Nnn
1631   \g_@@_renderer_arities_prop
1632   { table }
1633   { 3 }
1634 \ExplSyntaxOff

```

2.2.3.25 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.

```

1635 \def\markdownRendererInlineHtmlComment{%
1636   \markdownRendererInlineHtmlCommentPrototype}%
1637 \ExplSyntaxOn
1638 \seq_put_right:Nn
1639   \g_@@_renderers_seq
1640   { inlineHtmlComment }

```

```

1641 \prop_put:Nnn
1642   \g_@@_renderer_arities_prop
1643   { inlineHtmlComment }
1644   { 1 }
1645 \ExplSyntaxOff
1646 \def\markdownRendererBlockHtmlCommentBegin{%
1647   \markdownRendererBlockHtmlCommentBeginPrototype}%
1648 \ExplSyntaxOn
1649 \seq_put_right:Nn
1650   \g_@@_renderers_seq
1651   { blockHtmlCommentBegin }
1652 \prop_put:Nnn
1653   \g_@@_renderer_arities_prop
1654   { blockHtmlCommentBegin }
1655   { 0 }
1656 \ExplSyntaxOff
1657 \def\markdownRendererBlockHtmlCommentEnd{%
1658   \markdownRendererBlockHtmlCommentEndPrototype}%
1659 \ExplSyntaxOn
1660 \seq_put_right:Nn
1661   \g_@@_renderers_seq
1662   { blockHtmlCommentEnd }
1663 \prop_put:Nnn
1664   \g_@@_renderer_arities_prop
1665   { blockHtmlCommentEnd }
1666   { 0 }
1667 \ExplSyntaxOff

```

2.2.3.26 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.

```

1668 \def\markdownRendererInlineHtmlTag{%
1669   \markdownRendererInlineHtmlTagPrototype}%
1670 \ExplSyntaxOn
1671 \seq_put_right:Nn
1672   \g_@@_renderers_seq
1673   { inlineHtmlTag }
1674 \prop_put:Nnn
1675   \g_@@_renderer_arities_prop
1676   { inlineHtmlTag }

```



```

1677 { 1 }
1678 \ExplSyntaxOff
1679 \def\markdownRendererInputBlockHtmlElement{%
1680 \markdownRendererInputBlockHtmlElementPrototype}%
1681 \ExplSyntaxOn
1682 \seq_put_right:Nn
1683 \g_@@_renderers_seq
1684 { inputBlockHtmlElement }
1685 \prop_put:Nnn
1686 \g_@@_renderer_arities_prop
1687 { inputBlockHtmlElement }
1688 { 1 }
1689 \ExplSyntaxOff

```

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

`\markdownRendererAttributeIdentifier` represents the $\langle identifier \rangle$ of a mark-down element (`id=" $\langle identifier \rangle$ "` in HTML and `# $\langle identifier \rangle$` 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 mark-down element (`class=" $\langle class name \rangle$..."` in HTML and `.. $\langle class name \rangle$` 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.

```

1690 \def\markdownRendererAttributeIdentifier{%
1691 \markdownRendererAttributeIdentifierPrototype}%
1692 \ExplSyntaxOn
1693 \seq_put_right:Nn
1694 \g_@@_renderers_seq
1695 { attributeIdentifier }
1696 \prop_put:Nnn
1697 \g_@@_renderer_arities_prop
1698 { attributeIdentifier }
1699 { 1 }
1700 \ExplSyntaxOff
1701 \def\markdownRendererAttributeClassName{%
1702 \markdownRendererAttributeClassNamePrototype}%
1703 \ExplSyntaxOn
1704 \seq_put_right:Nn
1705 \g_@@_renderers_seq
1706 { attributeClassName }
1707 \prop_put:Nnn

```

```

1708 \g_@@_renderer_arities_prop
1709 { attributeClassName }
1710 { 1 }
1711 \ExplSyntaxOff
1712 \def\markdownRendererAttributeKeyValue{%
1713 \markdownRendererAttributeKeyValuePrototype}%
1714 \ExplSyntaxOn
1715 \seq_put_right:Nn
1716 \g_@@_renderers_seq
1717 { attributeKeyValue }
1718 \prop_put:Nnn
1719 \g_@@_renderer_arities_prop
1720 { attributeKeyValue }
1721 { 2 }
1722 \ExplSyntaxOff

```

2.2.3.28 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.

```

1723 \def\markdownRendererHeaderAttributeContextBegin{%
1724 \markdownRendererHeaderAttributeContextBeginPrototype}%
1725 \ExplSyntaxOn
1726 \seq_put_right:Nn
1727 \g_@@_renderers_seq
1728 { headerAttributeContextBegin }
1729 \prop_put:Nnn
1730 \g_@@_renderer_arities_prop
1731 { headerAttributeContextBegin }
1732 { 0 }
1733 \ExplSyntaxOff
1734 \def\markdownRendererHeaderAttributeContextEnd{%
1735 \markdownRendererHeaderAttributeContextEndPrototype}%
1736 \ExplSyntaxOn
1737 \seq_put_right:Nn
1738 \g_@@_renderers_seq
1739 { headerAttributeContextEnd }
1740 \prop_put:Nnn
1741 \g_@@_renderer_arities_prop
1742 { headerAttributeContextEnd }
1743 { 0 }
1744 \ExplSyntaxOff

```

2.2.3.29 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.

```

1745 \def\markdownRendererStrikeThrough{%
1746   \markdownRendererStrikeThroughPrototype}%
1747 \ExplSyntaxOn
1748 \seq_put_right:Nn
1749   \g_@@_renderers_seq
1750   { strikeThrough }
1751 \prop_put:Nnn
1752   \g_@@_renderer_arities_prop
1753   { strikeThrough }
1754   { 1 }
1755 \ExplSyntaxOff

```

2.2.3.30 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.

```

1756 \def\markdownRendererSuperscript{%
1757   \markdownRendererSuperscriptPrototype}%
1758 \ExplSyntaxOn
1759 \seq_put_right:Nn
1760   \g_@@_renderers_seq
1761   { superscript }
1762 \prop_put:Nnn
1763   \g_@@_renderer_arities_prop
1764   { superscript }
1765   { 1 }
1766 \ExplSyntaxOff

```

2.2.3.31 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.

```

1767 \def\markdownRendererSubscript{%
1768   \markdownRendererSubscriptPrototype}%
1769 \ExplSyntaxOn
1770 \seq_put_right:Nn
1771   \g_@@_renderers_seq
1772   { subscript }
1773 \prop_put:Nnn
1774   \g_@@_renderer_arities_prop

```

```

1775 { subscript }
1776 { 1 }
1777 \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.

```

1778 \ExplSyntaxOn
1779 \keys_define:nn
1780 { markdown/jekyllData }
1781 { }
1782 \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).

```

1783 \ExplSyntaxOn
1784 \cs_new:Nn \@@_plaintex_define_renderer_prototypes:
1785 {
1786   \seq_map_function:NN
1787   \g_@@_renderers_seq
1788   \@@_plaintex_define_renderer_prototype:n
1789   \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
1790   \let\markdownRendererBlockHtmlCommentBegin=\iffalse
1791   \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
1792   \let\markdownRendererBlockHtmlCommentEnd=\fi
1793 }
1794 \cs_new:Nn \@@_plaintex_define_renderer_prototype:n
1795 {
1796   \@@_renderer_prototype_tl_to_csname:nN
1797   { #1 }
1798   \l_tmpa_tl
1799   \prop_get:NnN
1800   \g_@@_renderer_arities_prop
1801   { #1 }
1802   \l_tmpb_tl
1803   \@@_plaintex_define_renderer_prototype:cV
1804   { \l_tmpa_tl }
1805   \l_tmpb_tl
1806 }
1807 \cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
1808 {

```

```

1809     \tl_set:Nn
1810     \l_tmpa_tl
1811     { \str_uppercase:n { #1 } }
1812     \tl_set:Nx
1813     #2
1814     {
1815         markdownRenderer
1816         \tl_head:f { \l_tmpa_tl }
1817         \tl_tail:n { #1 }
1818         Prototype
1819     }
1820 }
1821 \cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
1822 {
1823     \cs_generate_from_arg_count:NNnn
1824     #1
1825     \cs_set:Npn
1826     { #2 }
1827     { }
1828 }
1829 \cs_generate_variant:Nn
1830 \@@_plaintex_define_renderer_prototype:Nn
1831 { cV }
1832 \@@_plaintex_define_renderer_prototypes:
1833 \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 T_EX engine that does not support direct Lua access is starting to buffer a text. The plain T_EX implementation changes the category code of plain T_EX special characters to other, but there may be other active characters that may break the output. This macro should temporarily change the category of these to *other*.

```

1834 \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

T_EX 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.

```
1835 \let\markdownReadAndConvert\relax
1836 \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*.

```
1837 \catcode`\|=0\catcode`\=12%
1838 |gdef|markdownBegin{%
1839 |markdownReadAndConvert{\markdownEnd}%
1840 |markdownEnd}%
1841 |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 T_EX 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 T_EX 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.

```
1842 \ExplSyntaxOn
1843 \cs_if_exist:NF
1844 |markdownMode
1845 {
1846 |file_if_exist:nTF
1847 |lt3luabridge.tex }
1848 {
1849 |cs_new:Npn
1850 |markdownMode
1851 |3 }
1852 }
1853 {
1854 |cs_if_exist:NTF
```

```

1855     \directlua
1856     {
1857         \cs_new:Npn
1858             \markdownMode
1859             { 2 }
1860     }
1861     {
1862         \cs_new:Npn
1863             \markdownMode
1864             { 0 }
1865     }
1866 }
1867 }

```

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

```

1868 \def\markdownLuaRegisterIBCallback#1{\relax}%
1869 \def\markdownLuaUnregisterIBCallback#1{\relax}%

```

2.3 L^AT_EX Interface

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

The L^AT_EX implementation redefines the plain T_EX logging macros (see Section 3.2.1) to use the L^AT_EX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

```

1870 \newcommand\markdownInfo[1]{\PackageInfo{markdown}{#1}}%
1871 \newcommand\markdownWarning[1]{\PackageWarning{markdown}{#1}}%
1872 \newcommand\markdownError[2]{\PackageError{markdown}{#1}{#2.}}%
1873 \input markdown/markdown

```

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

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

where *<options>* are the L^AT_EX interface options (see Section 2.3.2). 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^AT_EX 2_ε parses package options.

2.3.1 Typesetting Markdown

The interface exposes the `markdown` and `markdown*` L^AT_EX environments, and redefines the `\markdownInput` command.

The `markdown` and `markdown*` L^AT_EX environments are used to typeset markdown document fragments. The starred version of the `markdown` environment accepts L^AT_EX interface options (see Section 2.3.2) as its only argument. These options will only influence this markdown document fragment.

```
1874 \newenvironment{markdown}\relax\relax
1875 \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^AT_EX environment (and likewise for the starred version).

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

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

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

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

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

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

2.3.2 Options

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

Except for the `plain` option described in Section 2.3.2.1, and the \LaTeX themes described in Section 2.3.2.2, and the \LaTeX setup snippets described in Section 2.3.2.3, \LaTeX options map directly to the options recognized by the plain \TeX interface (see Section 2.2.2) and to the markdown token renderers and their prototypes recognized by the plain \TeX interface (see Sections 2.2.3 and 2.2.4).

The \LaTeX options may be specified when loading the \LaTeX package, when using the `markdown*` \LaTeX environment or the `\markdownInput` macro (see Section 2.3), or via the `\markdownSetup` macro. The `\markdownSetup` macro receives the options to set up as its only argument:

```
1876 \ExplSyntaxOn
1877 \cs_new:Nn
1878   \@@_setup:n
1879   {
1880     \keys_set:nn
1881       { markdown/latex-options }
1882       { #1 }
1883   }
1884 \let\markdownSetup=\@@_setup:n
1885 \ExplSyntaxOff
```

We may also store \LaTeX 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:

```
1886 \newcommand\markdownSetupSnippet[2]{%
1887   \markdownIfSnippetExists{#1}%
1888   {%
1889     \markdownWarning
1890       {Redefined setup snippet \markdownLaTeXThemeName#1}%
1891     \csname markdownLaTeXSetupSnippet%
1892       \markdownLaTeXThemeName#1\endcsname={#2}%
1893   }{%
1894     \newtoks\next
1895     \next={#2}%
1896     \expandafter\let\csname markdownLaTeXSetupSnippet%
1897       \markdownLaTeXThemeName#1\endcsname=\next
1898   }%
}
```

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

```
1899 \newcommand\markdownIfSnippetExists[3]{%
1900   \@ifundefined
```

```

1901     {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
1902     {#3}{#2}}%

```

See Section 2.3.2.2 for information on interactions between setup snippets and L^AT_EX themes. See Section 2.3.2.3 for information about invoking the stored setup snippets.

To enable the enumeration of L^AT_EX options, we will maintain the `\g_@@_latex_options_seq` sequence.

```

1903 \ExplSyntaxOn
1904 \seq_new:N \g_@@_latex_options_seq

```

To enable the reflection of default L^AT_EX options and their types, we will maintain the `\g_@@_default_latex_options_prop` and `\g_@@_latex_option_types_prop` property lists, respectively.

```

1905 \prop_new:N \g_@@_latex_option_types_prop
1906 \prop_new:N \g_@@_default_latex_options_prop
1907 \tl_const:Nn \c_@@_option_layer_latex_tl { latex }
1908 \seq_put_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
1909 \cs_new:Nn
1910   \@@_add_latex_option:nnn
1911   {
1912     \@@_add_option:Vnnn
1913     \c_@@_option_layer_latex_tl
1914     { #1 }
1915     { #2 }
1916     { #3 }
1917   }

```

2.3.2.1 No default token renderer prototypes Default token renderer prototypes require L^AT_EX 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 T_EX implementation (see Section 3.2.2) and prevent the soft L^AT_EX 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}
```

```

1918 \@@_add_latex_option:nnn
1919   { plain }
1920   { boolean }
1921   { false }
1922 \ExplSyntaxOff

```

2.3.2.2 L^AT_EX themes User-defined L^AT_EX themes for the Markdown package provide a domain-specific interpretation of Markdown tokens. Similarly to L^AT_EX packages, themes allow the authors to achieve a specific look and other high-level goals without low-level programming.

The L^AT_EX option `theme=<theme name>` loads a L^AT_EX 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^AT_EX package, which provides similar functionality with its `\usetheme` macro [6, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single L^AT_EX document class or for a single L^AT_EX 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^AT_EX packages are identified only by their filenames, not by their pathnames. [7] 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^AT_EX package named `markdownthemewitiko_beamer_MU.sty`.

If the L^AT_EX 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^AT_EX package has been loaded. Otherwise, the theme(s) will be loaded immediately. For example, there is a theme named `witiko/dot`, which typesets fenced code blocks with the `dot` infostring as images of directed graphs rendered by the Graphviz tools. The following code would first load the Markdown package, then the `markdownthemewitiko_beamer_MU.sty` L^AT_EX package, and finally the `markdownthemewitiko_dot.sty` L^AT_EX package:

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

```
1923 \newif\ifmarkdownLaTeXLoaded
1924 \markdownLaTeXLoadedfalse
1925 \AtEndOfPackage{\markdownLaTeXLoadedtrue}
1926 \ExplSyntaxOn
1927 \tl_new:N \markdownLaTeXThemePackageName
```

```

1928 \cs_new:Nn
1929   \@@_set_latex_theme:n
1930   {
1931     \str_if_in:nnF
1932       { #1 }
1933       { / }
1934       {
1935         \markdownError
1936         { Won't~load~theme~with~unqualified~name~#1 }
1937         { Theme~names~must~contain~at~least~one~forward~slash }
1938       }
1939     \str_if_in:nnT
1940       { #1 }
1941       { _ }
1942       {
1943         \markdownError
1944         { Won't~load~theme~with~an~underscore~in~its~name~#1 }
1945         { Theme~names~must~not~contain~underscores~in~their~names }
1946       }
1947     \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
1948     \str_replace_all:Nnn
1949       \markdownLaTeXThemePackageName
1950       { / }
1951       { _ }
1952     \edef\markdownLaTeXThemePackageName{
1953       markdowntheme\markdownLaTeXThemePackageName}
1954     \expandafter\markdownLaTeXThemeLoad\expandafter{
1955       \markdownLaTeXThemePackageName}{#1/}
1956   }
1957 \keys_define:nn
1958   { markdown/latex-options }
1959   {
1960     theme .code:n = { \@@_set_latex_theme:n { #1 } },
1961   }
1962 \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.

```

1963 \ExplSyntaxOn
1964 \@onlypreamble
1965   \@@_set_latex_theme:n
1966 \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.

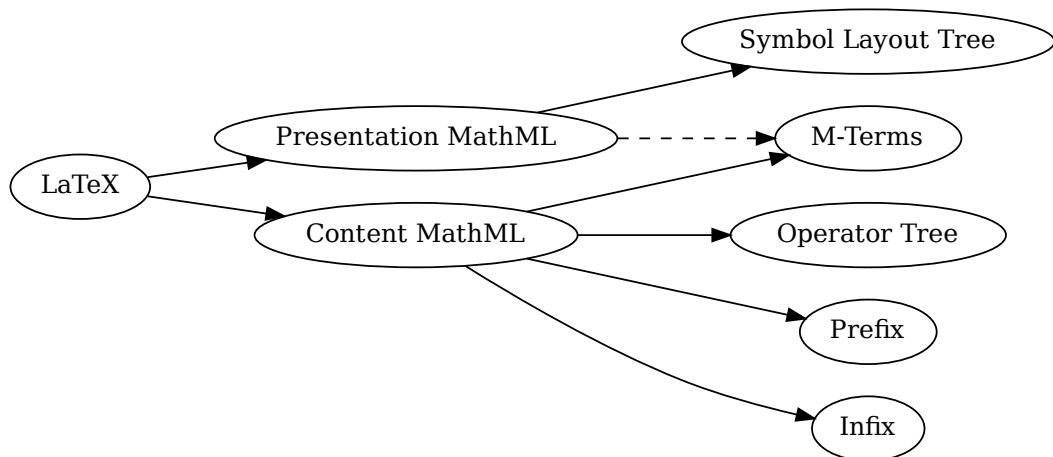


Figure 4: Various formats of mathematical formulae

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

1967 `\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}

  "The banner of the Markdown package"
\end{markdown}
\end{document}

```

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

1968 `\ProvidesPackage{markdownthemewitiko_graphicx_http}[2021/03/22]%`

```

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

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

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

```



Chapter 1

Introduction

1.1 Section

1.1.1 Subsection

Hello *Markdown*!

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

Table 1.1: Table

Figure 5: The banner of the Markdown package

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

```

\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”.

1969 \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 *<value>*:

1970 \ExplSyntaxOn

```

1971 \keys_define:nn
1972   { markdown/latex-options }
1973   {
1974     snippet .code:n = {
1975       \markdownIfSnippetExists{#1}
1976       {
1977         \expandafter\markdownSetup\expandafter{
1978           \the\csname markdownLaTeXSetupSnippet
1979             \markdownLaTeXThemeName#1\endcsname}
1980       }{
1981         \markdownError
1982           {Can't~invoke~setup~snippet~#1}
1983           {The~setup~snippet~is~undefined}
1984       }
1985     }
1986   }
1987 \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.


```

1988 \ExplSyntaxOn
1989 \cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
1990 {
1991   \seq_map_inline:Nn
1992     \g_@@_latex_options_seq
1993     {
1994       \@@_plain_tex_define_option_command:n
1995       { ##1 }
1996     }

```

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.

```

1997   \seq_map_inline:Nn
1998     \g_@@_option_layers_seq
1999     {
2000       \seq_map_inline:cn
2001       { g_@@_ ##1 _options_seq }
2002       {
2003         \@@_latex_define_option_keyval:nn
2004         { ##1 }
2005         { #####1 }
2006       }
2007     }
2008 }
2009 \cs_new:Nn \@@_latex_define_option_keyval:nn
2010 {
2011   \prop_get:cnN
2012     { g_@@_ #1 _option_types_prop }
2013     { #2 }
2014     \l_tmpa_tl
2015   \keys_define:nn
2016     { markdown/latex-options }
2017     {
2018       #2 .code:n = {
2019         \@@_set_option_value:nn
2020         { #2 }
2021         { ##1 }
2022       },
2023     }
2024   \str_if_eq:VVT
2025     \l_tmpa_tl
2026     \c_@@_option_type_boolean_tl
2027     {
2028       \keys_define:nn
2029         { markdown/latex-options }
2030         {
2031           #2 .default:n = { true },

```

```

2032     }
2033 }

```

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.

```

2034     \str_if_eq:VVT
2035     \l_tmpa_tl
2036     \c_@@_option_type_clist_tl
2037     {
2038         \tl_set:Nn
2039             \l_tmpa_tl
2040             { #2 }
2041         \tl_reverse:N
2042             \l_tmpa_tl
2043         \str_if_eq:enF
2044             {
2045                 \tl_head:V
2046                 \l_tmpa_tl
2047             }
2048         { s }
2049         {
2050             \msg_error:nnn
2051                 { @@ }
2052                 { malformed-name-for-clist-option }
2053                 { #2 }
2054         }
2055         \tl_set:Nx
2056             \l_tmpa_tl
2057             {
2058                 \tl_tail:V
2059                 \l_tmpa_tl
2060             }
2061         \tl_reverse:N
2062             \l_tmpa_tl
2063         \tl_put_right:Nn
2064             \l_tmpa_tl
2065             {
2066                 .code:n = {
2067                     \@@_get_option_value:nN
2068                         { #2 }
2069                     \l_tmpa_tl
2070                     \clist_set:NV
2071                         \l_tmpa_clist
2072                     { \l_tmpa_tl, { ##1 } }

```

```

2073         \@@_set_option_value:nV
2074         { #2 }
2075         \l_tmpa_clist
2076     }
2077 }
2078 \keys_define:nV
2079 { markdown/latex-options }
2080 \l_tmpa_tl
2081 }
2082 }
2083 \cs_generate_variant:Nn
2084 \clist_set:Nn
2085 { NV }
2086 \cs_generate_variant:Nn
2087 \keys_define:nn
2088 { nV }
2089 \cs_generate_variant:Nn
2090 \@@_set_option_value:nn
2091 { nV }
2092 \prg_generate_conditional_variant:Nnn
2093 \str_if_eq:nn
2094 { en }
2095 { F }
2096 \msg_new:nnn
2097 { @@ }
2098 { malformed-name-for-clist-option }
2099 {
2100     Clist~option~name~#1~does~not~end~with~-s.
2101 }
2102 \@@_latex_define_option_commands_and_keyvals:
2103 \ExplSyntaxOff

```

The `\markdownOptionFinalizeCache` and `\markdownOptionFrozenCache` plain TeX options are exposed through L^AT_EX options with keys `finalizeCache` and `frozenCache`.

To ensure compatibility with the `minted` package [8, Section 5.1], which supports the `finalizcache` and `frozenscache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document class options. By passing `finalizcache` and `frozenscache` as document class options, you may conveniently control the behavior of both packages at once:

```

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

```

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

```
2104 \DeclareOption{finalizcache}{\markdownSetup{finalizeCache}}
2105 \DeclareOption{frozenscache}{\markdownSetup{frozenCache}}
```

The following example L^AT_EX code showcases a possible configuration of plain T_EX interface options `\markdownOptionHybrid`, `\markdownOptionSmartEllipses`, and `\markdownOptionCacheDir`.

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

2.3.2.5 Plain T_EX Markdown Token Renderers The L^AT_EX 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 T_EX interface (see Section 2.2.3).

```
2106 \ExplSyntaxOn
2107 \cs_new:Nn \@@_latex_define_renderers:
2108 {
2109   \seq_map_function:NN
2110     \g_@@_renderers_seq
2111     \@@_latex_define_renderer:n
2112 }
2113 \cs_new:Nn \@@_latex_define_renderer:n
2114 {
2115   \@@_renderer_tl_to_csname:nN
2116     { #1 }
2117     \l_tmpa_tl
2118   \prop_get:NnN
2119     \g_@@_renderer_arities_prop
2120     { #1 }
2121     \l_tmpb_tl
2122   \@@_latex_define_renderer:ncV
2123     { #1 }
2124     { \l_tmpa_tl }
2125     \l_tmpb_tl
2126 }
2127 \cs_new:Nn \@@_renderer_tl_to_csname:nN
2128 {
2129   \tl_set:Nn
2130     \l_tmpa_tl
```

```

2131     { \str_uppercase:n { #1 } }
2132   \tl_set:Nx
2133     #2
2134     {
2135       markdownRenderer
2136       \tl_head:f { \l_tmpa_tl }
2137       \tl_tail:n { #1 }
2138     }
2139   }
2140   \cs_new:Nn \@@_latex_define_renderer:nNn
2141     {
2142       \keys_define:nn
2143         { markdown/latex-options/renderers }
2144         {
2145           #1 .code:n = {
2146             \cs_generate_from_arg_count:NNnn
2147               #2
2148               \cs_set:Npn
2149                 { #3 }
2150                 { ##1 }
2151           },
2152         }
2153     }
2154   \cs_generate_variant:Nn
2155     \@@_latex_define_renderer:nNn
2156     { ncV }
2157   \ExplSyntaxOff

```

The following example \LaTeX 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 \TeX Markdown Token Renderer Prototypes The \LaTeX 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 \TeX interface (see Section 2.2.4).

```

2158 \ExplSyntaxOn
2159 \cs_new:Nn \@@_latex_define_renderer_prototypes:

```

```

2160 {
2161   \seq_map_function:NN
2162   \g_@@_renderers_seq
2163   \@@_latex_define_renderer_prototype:n
2164 }
2165 \cs_new:Nn \@@_latex_define_renderer_prototype:n
2166 {
2167   \@@_renderer_prototype_tl_to_csname:nN
2168   { #1 }
2169   \l_tmpa_tl
2170   \prop_get:NnN
2171   \g_@@_renderer_arities_prop
2172   { #1 }
2173   \l_tmpb_tl
2174   \@@_latex_define_renderer_prototype:ncV
2175   { #1 }
2176   { \l_tmpa_tl }
2177   \l_tmpb_tl
2178 }
2179 \cs_new:Nn \@@_latex_define_renderer_prototype:nNn
2180 {
2181   \keys_define:nn
2182   { markdown/latex-options/renderer-prototypes }
2183   {
2184     #1 .code:n = {
2185       \cs_generate_from_arg_count:NNnn
2186       #2
2187       \cs_set:Npn
2188       { #3 }
2189       { ##1 }
2190     },
2191   }
2192 }
2193 \cs_generate_variant:Nn
2194 \@@_latex_define_renderer_prototype:nNn
2195 { ncV }
2196 \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).

```
2197 \writestatus{loading}{ConTEXt User Module / markdown}%
2198 \startmodule[markdown]
2199 \unprotect
```

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

```
2200 \def\markdownInfo#1{\writestatus{markdown}{#1.}}%
2201 \def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
2202 \def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
2203 \do\#\do\~\do\_ \do\% \do\~}%
2204 \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 `\input`ting 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.

```
2205 \let\startmarkdown\relax
2206 \let\stopmarkdown\relax
2207 \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 \TeX . 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:

```

2208 \ExplSyntaxOn
2209 \cs_new:Nn
2210   \@@_setup:n
2211   {
2212     \keys_set:nn
2213       { markdown/context-options }
2214       { #1 }
2215   }
2216 \long\def\setupmarkdown[#1]
2217   {
2218     \@@_setup:n

```



```

2219     { #1 }
2220   }
2221 \ExplSyntaxOff

```

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

```

2222 \ExplSyntaxOn
2223 \cs_new:Nn \@@_context_define_option_commands_and_keyvals:
2224 {
2225   \seq_map_inline:Nn
2226     \g_@@_option_layers_seq
2227     {
2228       \seq_map_inline:cn
2229         { g_@@_ ##1 _options_seq }
2230         {
2231           \@@_context_define_option_keyval:nn
2232             { ##1 }
2233             { ####1 }
2234         }
2235     }
2236 }
2237 \cs_new:Nn \@@_context_define_option_keyval:nn
2238 {
2239   \prop_get:cnN
2240     { g_@@_ #1 _option_types_prop }
2241     { #2 }
2242   \l_tmpa_tl
2243   \keys_define:nn
2244     { markdown/context-options }
2245     {
2246       #2 .code:n = {
2247         \tl_set:Nx
2248           \l_tmpa_tl
2249           {
2250             \str_case:nnF
2251               { ##1 }
2252               {
2253                 { yes } { true }
2254                 { no } { false }
2255               }
2256             { ##1 }
2257           }
2258       \@@_set_option_value:nV
2259         { #2 }
2260         \l_tmpa_tl
2261     },

```

```

2262     }
2263     \str_if_eq:VVT
2264     \l_tmpa_tl
2265     \c_@@_option_type_boolean_tl
2266     {
2267         \keys_define:nn
2268             { markdown/context-options }
2269             {
2270                 #2 .default:n = { true },
2271             }
2272     }
2273 }
2274 \cs_generate_variant:Nn
2275     \@@_set_option_value:nn
2276     { nV }
2277 \@@_context_define_option_commands_and_keyvals:
2278 \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 \TeX writer were hidden behind the converter functions exposed by the Lua interface (see Section 2.1).

```

2279 local upper, gsub, format, length =
2280     string.upper, string.gsub, string.format, string.len
2281 local P, R, S, V, C, Cg, Cb, Cmt, Cc, Ct, B, Cs, any =
2282     lpeg.P, lpeg.R, lpeg.S, lpeg.V, lpeg.C, lpeg.Cg, lpeg.Cb,
2283     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.

```
2284 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.

```
2285 function util.err(msg, exit_code)
2286   io.stderr:write("markdown.lua: " .. msg .. "\n")
2287   os.exit(exit_code or 1)
2288 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.

```
2289 function util.cache(dir, string, salt, transform, suffix)
2290   local digest = md5.sumhexa(string .. (salt or ""))
2291   local name = util.pathname(dir, digest .. suffix)
2292   local file = io.open(name, "r")
2293   if file == nil then -- If no cache entry exists, then create a new one.
2294     file = assert(io.open(name, "w"),
2295       [[Could not open file ]] .. name .. [[ for writing]])
2296     local result = string
2297     if transform ~= nil then
2298       result = transform(result)
2299     end
2300     assert(file:write(result))
2301     assert(file:close())
2302   end
2303   return name
2304 end
```

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

```
2305 function util.table_copy(t)
2306   local u = { }
2307   for k, v in pairs(t) do u[k] = v end
2308   return setmetatable(u, getmetatable(t))
2309 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 T_EX directory structure. Further options for `kpathsea` can be specified in table `options`. [1, Section 10.7.4]

```

2310 util.lookup_files = (function()
2311     local ran_ok, kpse = pcall(require, "kpse")
2312     if ran_ok then
2313         kpse.set_program_name("luatex")
2314     else
2315         kpse = { lookup = function(f, _) return f end }
2316     end
2317
2318     local function lookup_files(f, options)
2319         return kpse.lookup(f, options)
2320     end
2321
2322     return lookup_files
2323 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 [9, Chapter 21].

```

2324 function util.expand_tabs_in_line(s, tabstop)
2325     local tab = tabstop or 4
2326     local corr = 0
2327     return (s:gsub("()\t", function(p)
2328         local sp = tab - (p - 1 + corr) % tab
2329         corr = corr - 1 + sp
2330         return string.rep(" ", sp)
2331     end))
2332 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.

```

2333 function util.walk(t, f)
2334     local typ = type(t)
2335     if typ == "string" then
2336         f(t)
2337     elseif typ == "table" then
2338         local i = 1
2339         local n
2340         n = t[i]
2341         while n do
2342             util.walk(n, f)
2343             i = i + 1
2344             n = t[i]
2345         end
2346     elseif typ == "function" then

```

```

2347     local ok, val = pcall(t)
2348     if ok then
2349         util.walk(val,f)
2350     end
2351 else
2352     f(tostring(t))
2353 end
2354 end

```

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

```

2355 function util.flatten(ary)
2356     local new = {}
2357     for _,v in ipairs(ary) do
2358         if type(v) == "table" then
2359             for _,w in ipairs(util.flatten(v)) do
2360                 new[#new + 1] = w
2361             end
2362         else
2363             new[#new + 1] = v
2364         end
2365     end
2366     return new
2367 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.

```

2368 function util.rope_to_string(rope)
2369     local buffer = {}
2370     util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2371     return table.concat(buffer)
2372 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.

```

2373 function util.rope_last(rope)
2374     if #rope == 0 then
2375         return nil
2376     else
2377         local l = rope[#rope]
2378         if type(l) == "table" then
2379             return util.rope_last(l)
2380         else
2381             return l
2382         end
2383     end
2384 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 $1 \leq i \leq \#ary$.

```

2385 function util.intersperse(ary, x)
2386   local new = {}
2387   local l = #ary
2388   for i,v in ipairs(ary) do
2389     local n = #new
2390     new[n + 1] = v
2391     if i ~= l then
2392       new[n + 2] = x
2393     end
2394   end
2395   return new
2396 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 $1 \leq i \leq \#ary$.

```

2397 function util.map(ary, f)
2398   local new = {}
2399   for i,v in ipairs(ary) do
2400     new[i] = f(v)
2401   end
2402   return new
2403 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.

```

2404 function util.escaper(char_escapes, string_escapes)

```

Build a string of escapable characters.

```

2405   local char_escapes_list = ""
2406   for i,_ in pairs(char_escapes) do
2407     char_escapes_list = char_escapes_list .. i
2408   end

```

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

```

2409   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.

```

2410 if string_escapes then
2411   for k,v in pairs(string_escapes) do
2412     escapable = P(k) / v + escapable
2413   end
2414 end

```

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

```

2415 local escape_string = Cs((escapable + any)^0)

```

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

```

2416 return function(s)
2417   return lpeg.match(escape_string, s)
2418 end
2419 end

```

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

```

2420 function util.pathname(dir, file)
2421   if #dir == 0 then
2422     return file
2423   else
2424     return dir .. "/" .. file
2425   end
2426 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.

```

2427 local entities = {}
2428
2429 local character_entities = {
2430   ["Tab"] = 9,
2431   ["NewLine"] = 10,
2432   ["excl"] = 33,
2433   ["quot"] = 34,
2434   ["QUOT"] = 34,
2435   ["num"] = 35,
2436   ["dollar"] = 36,
2437   ["percent"] = 37,

```

```

2438 ["amp"] = 38,
2439 ["AMP"] = 38,
2440 ["apos"] = 39,
2441 ["lpar"] = 40,
2442 ["rpar"] = 41,
2443 ["ast"] = 42,
2444 ["midast"] = 42,
2445 ["plus"] = 43,
2446 ["comma"] = 44,
2447 ["period"] = 46,
2448 ["sol"] = 47,
2449 ["colon"] = 58,
2450 ["semi"] = 59,
2451 ["lt"] = 60,
2452 ["LT"] = 60,
2453 ["equals"] = 61,
2454 ["gt"] = 62,
2455 ["GT"] = 62,
2456 ["quest"] = 63,
2457 ["commat"] = 64,
2458 ["lsqb"] = 91,
2459 ["lbrack"] = 91,
2460 ["bsol"] = 92,
2461 ["rsqb"] = 93,
2462 ["rbrack"] = 93,
2463 ["Hat"] = 94,
2464 ["lowbar"] = 95,
2465 ["grave"] = 96,
2466 ["DiacriticalGrave"] = 96,
2467 ["lcub"] = 123,
2468 ["lbrace"] = 123,
2469 ["verbar"] = 124,
2470 ["vert"] = 124,
2471 ["VerticalLine"] = 124,
2472 ["rcub"] = 125,
2473 ["rbrace"] = 125,
2474 ["nbsp"] = 160,
2475 ["NonBreakingSpace"] = 160,
2476 ["iexcl"] = 161,
2477 ["cent"] = 162,
2478 ["pound"] = 163,
2479 ["curren"] = 164,
2480 ["yen"] = 165,
2481 ["brvbar"] = 166,
2482 ["sect"] = 167,
2483 ["Dot"] = 168,
2484 ["die"] = 168,

```


2485 ["DoubleDot"] = 168,
 2486 ["uml"] = 168,
 2487 ["copy"] = 169,
 2488 ["COPY"] = 169,
 2489 ["ordf"] = 170,
 2490 ["laquo"] = 171,
 2491 ["not"] = 172,
 2492 ["shy"] = 173,
 2493 ["reg"] = 174,
 2494 ["circledR"] = 174,
 2495 ["REG"] = 174,
 2496 ["macr"] = 175,
 2497 ["OverBar"] = 175,
 2498 ["strns"] = 175,
 2499 ["deg"] = 176,
 2500 ["plusmn"] = 177,
 2501 ["pm"] = 177,
 2502 ["PlusMinus"] = 177,
 2503 ["sup2"] = 178,
 2504 ["sup3"] = 179,
 2505 ["acute"] = 180,
 2506 ["DiacriticalAcute"] = 180,
 2507 ["micro"] = 181,
 2508 ["para"] = 182,
 2509 ["middot"] = 183,
 2510 ["centerdot"] = 183,
 2511 ["CenterDot"] = 183,
 2512 ["cedil"] = 184,
 2513 ["Cedilla"] = 184,
 2514 ["sup1"] = 185,
 2515 ["ordm"] = 186,
 2516 ["raquo"] = 187,
 2517 ["frac14"] = 188,
 2518 ["frac12"] = 189,
 2519 ["half"] = 189,
 2520 ["frac34"] = 190,
 2521 ["iquest"] = 191,
 2522 ["Agrave"] = 192,
 2523 ["Aacute"] = 193,
 2524 ["Acirc"] = 194,
 2525 ["Atilde"] = 195,
 2526 ["Auml"] = 196,
 2527 ["Aring"] = 197,
 2528 ["AElig"] = 198,
 2529 ["Ccedil"] = 199,
 2530 ["Egrave"] = 200,
 2531 ["Eacute"] = 201,

2532 ["Ecirc"] = 202,
 2533 ["Euml"] = 203,
 2534 ["Igrave"] = 204,
 2535 ["Iacute"] = 205,
 2536 ["Icirc"] = 206,
 2537 ["Iuml"] = 207,
 2538 ["ETH"] = 208,
 2539 ["Ntilde"] = 209,
 2540 ["Ograve"] = 210,
 2541 ["Oacute"] = 211,
 2542 ["Ocirc"] = 212,
 2543 ["Otilde"] = 213,
 2544 ["Ouml"] = 214,
 2545 ["times"] = 215,
 2546 ["Oslash"] = 216,
 2547 ["Ugrave"] = 217,
 2548 ["Uacute"] = 218,
 2549 ["Ucirc"] = 219,
 2550 ["Uuml"] = 220,
 2551 ["Yacute"] = 221,
 2552 ["THORN"] = 222,
 2553 ["szlig"] = 223,
 2554 ["agrave"] = 224,
 2555 ["aacute"] = 225,
 2556 ["acirc"] = 226,
 2557 ["atilde"] = 227,
 2558 ["auml"] = 228,
 2559 ["aring"] = 229,
 2560 ["aelig"] = 230,
 2561 ["ccedil"] = 231,
 2562 ["egrave"] = 232,
 2563 ["eacute"] = 233,
 2564 ["ecirc"] = 234,
 2565 ["euml"] = 235,
 2566 ["igrave"] = 236,
 2567 ["iacute"] = 237,
 2568 ["icirc"] = 238,
 2569 ["iuml"] = 239,
 2570 ["eth"] = 240,
 2571 ["ntilde"] = 241,
 2572 ["ograve"] = 242,
 2573 ["oacute"] = 243,
 2574 ["ocirc"] = 244,
 2575 ["otilde"] = 245,
 2576 ["ouml"] = 246,
 2577 ["divide"] = 247,
 2578 ["div"] = 247,

2579 ["oslash"] = 248,
 2580 ["ugrave"] = 249,
 2581 ["uacute"] = 250,
 2582 ["ucirc"] = 251,
 2583 ["uuml"] = 252,
 2584 ["yacute"] = 253,
 2585 ["thorn"] = 254,
 2586 ["yuml"] = 255,
 2587 ["Amacr"] = 256,
 2588 ["amacr"] = 257,
 2589 ["Abreve"] = 258,
 2590 ["abreve"] = 259,
 2591 ["Aogon"] = 260,
 2592 ["aogon"] = 261,
 2593 ["Cacute"] = 262,
 2594 ["cacute"] = 263,
 2595 ["Ccirc"] = 264,
 2596 ["ccirc"] = 265,
 2597 ["Cdot"] = 266,
 2598 ["cdot"] = 267,
 2599 ["Ccaron"] = 268,
 2600 ["ccaron"] = 269,
 2601 ["Dcaron"] = 270,
 2602 ["dcaron"] = 271,
 2603 ["Dstrok"] = 272,
 2604 ["dstrok"] = 273,
 2605 ["Emacr"] = 274,
 2606 ["emacr"] = 275,
 2607 ["Edot"] = 278,
 2608 ["edot"] = 279,
 2609 ["Eogon"] = 280,
 2610 ["eogon"] = 281,
 2611 ["Ecaron"] = 282,
 2612 ["ecaron"] = 283,
 2613 ["Gcirc"] = 284,
 2614 ["gcirc"] = 285,
 2615 ["Gbreve"] = 286,
 2616 ["gbreve"] = 287,
 2617 ["Gdot"] = 288,
 2618 ["gdot"] = 289,
 2619 ["Gcedil"] = 290,
 2620 ["Hcirc"] = 292,
 2621 ["hcirc"] = 293,
 2622 ["Hstrok"] = 294,
 2623 ["hstrok"] = 295,
 2624 ["Itilde"] = 296,
 2625 ["itilde"] = 297,

```

2626 ["Imacr"] = 298,
2627 ["imacr"] = 299,
2628 ["Iogon"] = 302,
2629 ["iogon"] = 303,
2630 ["Idot"] = 304,
2631 ["imath"] = 305,
2632 ["inodot"] = 305,
2633 ["IJlig"] = 306,
2634 ["ijlig"] = 307,
2635 ["Jcirc"] = 308,
2636 ["jcirc"] = 309,
2637 ["Kcedil"] = 310,
2638 ["kcedil"] = 311,
2639 ["kgreen"] = 312,
2640 ["Lacute"] = 313,
2641 ["lacute"] = 314,
2642 ["Lcedil"] = 315,
2643 ["lcedil"] = 316,
2644 ["Lcaron"] = 317,
2645 ["lcaron"] = 318,
2646 ["Lmidot"] = 319,
2647 ["lmidot"] = 320,
2648 ["Lstrok"] = 321,
2649 ["lstrok"] = 322,
2650 ["Nacute"] = 323,
2651 ["nacute"] = 324,
2652 ["Ncedil"] = 325,
2653 ["ncedil"] = 326,
2654 ["Ncaron"] = 327,
2655 ["ncaron"] = 328,
2656 ["napos"] = 329,
2657 ["ENG"] = 330,
2658 ["eng"] = 331,
2659 ["Omacr"] = 332,
2660 ["omacr"] = 333,
2661 ["Odblac"] = 336,
2662 ["odblac"] = 337,
2663 ["OElig"] = 338,
2664 ["oelig"] = 339,
2665 ["Racute"] = 340,
2666 ["racute"] = 341,
2667 ["Rcedil"] = 342,
2668 ["rcedil"] = 343,
2669 ["Rcaron"] = 344,
2670 ["rcaron"] = 345,
2671 ["Sacute"] = 346,
2672 ["sacute"] = 347,

```

2673 ["Scirc"] = 348,
 2674 ["scirc"] = 349,
 2675 ["Scedil"] = 350,
 2676 ["scedil"] = 351,
 2677 ["Scaron"] = 352,
 2678 ["scaron"] = 353,
 2679 ["Tcedil"] = 354,
 2680 ["tcedil"] = 355,
 2681 ["Tcaron"] = 356,
 2682 ["tcaron"] = 357,
 2683 ["Tstrok"] = 358,
 2684 ["tstrok"] = 359,
 2685 ["Utilde"] = 360,
 2686 ["utilde"] = 361,
 2687 ["Umacr"] = 362,
 2688 ["umacr"] = 363,
 2689 ["Ubreve"] = 364,
 2690 ["ubreve"] = 365,
 2691 ["Uring"] = 366,
 2692 ["uring"] = 367,
 2693 ["Udblac"] = 368,
 2694 ["udblac"] = 369,
 2695 ["Uogon"] = 370,
 2696 ["uogon"] = 371,
 2697 ["Wcirc"] = 372,
 2698 ["wcirc"] = 373,
 2699 ["Ycirc"] = 374,
 2700 ["ycirc"] = 375,
 2701 ["Yuml"] = 376,
 2702 ["Zacute"] = 377,
 2703 ["zacute"] = 378,
 2704 ["Zdot"] = 379,
 2705 ["zdot"] = 380,
 2706 ["Zcaron"] = 381,
 2707 ["zcaron"] = 382,
 2708 ["fnof"] = 402,
 2709 ["imped"] = 437,
 2710 ["gacute"] = 501,
 2711 ["jmath"] = 567,
 2712 ["circ"] = 710,
 2713 ["caron"] = 711,
 2714 ["Hacek"] = 711,
 2715 ["breve"] = 728,
 2716 ["Breve"] = 728,
 2717 ["dot"] = 729,
 2718 ["DiacriticalDot"] = 729,
 2719 ["ring"] = 730,

```

2720 ["ogon"] = 731,
2721 ["tilde"] = 732,
2722 ["DiacriticalTilde"] = 732,
2723 ["dblac"] = 733,
2724 ["DiacriticalDoubleAcute"] = 733,
2725 ["DownBreve"] = 785,
2726 ["UnderBar"] = 818,
2727 ["Alpha"] = 913,
2728 ["Beta"] = 914,
2729 ["Gamma"] = 915,
2730 ["Delta"] = 916,
2731 ["Epsilon"] = 917,
2732 ["Zeta"] = 918,
2733 ["Eta"] = 919,
2734 ["Theta"] = 920,
2735 ["Iota"] = 921,
2736 ["Kappa"] = 922,
2737 ["Lambda"] = 923,
2738 ["Mu"] = 924,
2739 ["Nu"] = 925,
2740 ["Xi"] = 926,
2741 ["Omicron"] = 927,
2742 ["Pi"] = 928,
2743 ["Rho"] = 929,
2744 ["Sigma"] = 931,
2745 ["Tau"] = 932,
2746 ["Upsilon"] = 933,
2747 ["Phi"] = 934,
2748 ["Chi"] = 935,
2749 ["Psi"] = 936,
2750 ["Omega"] = 937,
2751 ["alpha"] = 945,
2752 ["beta"] = 946,
2753 ["gamma"] = 947,
2754 ["delta"] = 948,
2755 ["epsiv"] = 949,
2756 ["varepsilon"] = 949,
2757 ["epsilon"] = 949,
2758 ["zeta"] = 950,
2759 ["eta"] = 951,
2760 ["theta"] = 952,
2761 ["iota"] = 953,
2762 ["kappa"] = 954,
2763 ["lambda"] = 955,
2764 ["mu"] = 956,
2765 ["nu"] = 957,
2766 ["xi"] = 958,

```

```

2767 ["omicron"] = 959,
2768 ["pi"] = 960,
2769 ["rho"] = 961,
2770 ["sigmav"] = 962,
2771 ["varsigma"] = 962,
2772 ["sigmaf"] = 962,
2773 ["sigma"] = 963,
2774 ["tau"] = 964,
2775 ["upsilon"] = 965,
2776 ["upsilon"] = 965,
2777 ["phi"] = 966,
2778 ["phiv"] = 966,
2779 ["varphi"] = 966,
2780 ["chi"] = 967,
2781 ["psi"] = 968,
2782 ["omega"] = 969,
2783 ["thetav"] = 977,
2784 ["vartheta"] = 977,
2785 ["thetasym"] = 977,
2786 ["Upsilon"] = 978,
2787 ["upsih"] = 978,
2788 ["straightphi"] = 981,
2789 ["piv"] = 982,
2790 ["varpi"] = 982,
2791 ["Gammad"] = 988,
2792 ["gammad"] = 989,
2793 ["digamma"] = 989,
2794 ["kappav"] = 1008,
2795 ["varkappa"] = 1008,
2796 ["rhov"] = 1009,
2797 ["varrho"] = 1009,
2798 ["epsi"] = 1013,
2799 ["straightepsilon"] = 1013,
2800 ["bepsi"] = 1014,
2801 ["backepsilon"] = 1014,
2802 ["IOcy"] = 1025,
2803 ["DJcy"] = 1026,
2804 ["GJcy"] = 1027,
2805 ["Jukcy"] = 1028,
2806 ["DScy"] = 1029,
2807 ["Iukcy"] = 1030,
2808 ["YIcy"] = 1031,
2809 ["Jsercy"] = 1032,
2810 ["LJcy"] = 1033,
2811 ["NJcy"] = 1034,
2812 ["TSHcy"] = 1035,
2813 ["KJcy"] = 1036,

```

```

2814 ["Ubrcy"] = 1038,
2815 ["DZcy"] = 1039,
2816 ["Acy"] = 1040,
2817 ["Bcy"] = 1041,
2818 ["Vcy"] = 1042,
2819 ["Gcy"] = 1043,
2820 ["Dcy"] = 1044,
2821 ["IEcy"] = 1045,
2822 ["ZHcy"] = 1046,
2823 ["Zcy"] = 1047,
2824 ["Icy"] = 1048,
2825 ["Jcy"] = 1049,
2826 ["Kcy"] = 1050,
2827 ["Lcy"] = 1051,
2828 ["Mcy"] = 1052,
2829 ["Ncy"] = 1053,
2830 ["Ocy"] = 1054,
2831 ["Pcy"] = 1055,
2832 ["Rcy"] = 1056,
2833 ["Scy"] = 1057,
2834 ["Tcy"] = 1058,
2835 ["Ucy"] = 1059,
2836 ["Fcy"] = 1060,
2837 ["KHcy"] = 1061,
2838 ["TScy"] = 1062,
2839 ["CHcy"] = 1063,
2840 ["SHcy"] = 1064,
2841 ["SHCHcy"] = 1065,
2842 ["HARDcy"] = 1066,
2843 ["Ycy"] = 1067,
2844 ["SOFTcy"] = 1068,
2845 ["Ecy"] = 1069,
2846 ["YUcy"] = 1070,
2847 ["YAcy"] = 1071,
2848 ["acy"] = 1072,
2849 ["bcy"] = 1073,
2850 ["vcy"] = 1074,
2851 ["gcy"] = 1075,
2852 ["dcy"] = 1076,
2853 ["iecy"] = 1077,
2854 ["zhcy"] = 1078,
2855 ["zcy"] = 1079,
2856 ["icy"] = 1080,
2857 ["jcy"] = 1081,
2858 ["kcy"] = 1082,
2859 ["lcy"] = 1083,
2860 ["mcy"] = 1084,

```



```

2861 ["ncy"] = 1085,
2862 ["ocy"] = 1086,
2863 ["pcy"] = 1087,
2864 ["rcy"] = 1088,
2865 ["scy"] = 1089,
2866 ["tcy"] = 1090,
2867 ["ucy"] = 1091,
2868 ["fcy"] = 1092,
2869 ["khcy"] = 1093,
2870 ["tscy"] = 1094,
2871 ["chcy"] = 1095,
2872 ["shcy"] = 1096,
2873 ["shchcy"] = 1097,
2874 ["hardcy"] = 1098,
2875 ["ycy"] = 1099,
2876 ["softcy"] = 1100,
2877 ["ecy"] = 1101,
2878 ["yucy"] = 1102,
2879 ["yacy"] = 1103,
2880 ["iocy"] = 1105,
2881 ["djcy"] = 1106,
2882 ["gjcy"] = 1107,
2883 ["jukcy"] = 1108,
2884 ["dscy"] = 1109,
2885 ["iukcy"] = 1110,
2886 ["yicy"] = 1111,
2887 ["jsercy"] = 1112,
2888 ["ljcy"] = 1113,
2889 ["njcy"] = 1114,
2890 ["tshcy"] = 1115,
2891 ["kjcy"] = 1116,
2892 ["ubrcy"] = 1118,
2893 ["dzcyc"] = 1119,
2894 ["ensp"] = 8194,
2895 ["emsp"] = 8195,
2896 ["emsp13"] = 8196,
2897 ["emsp14"] = 8197,
2898 ["numsp"] = 8199,
2899 ["puncsp"] = 8200,
2900 ["thinsp"] = 8201,
2901 ["ThinSpace"] = 8201,
2902 ["hairsp"] = 8202,
2903 ["VeryThinSpace"] = 8202,
2904 ["ZeroWidthSpace"] = 8203,
2905 ["NegativeVeryThinSpace"] = 8203,
2906 ["NegativeThinSpace"] = 8203,
2907 ["NegativeMediumSpace"] = 8203,

```

2908 ["NegativeThickSpace"] = 8203,
 2909 ["zwnj"] = 8204,
 2910 ["zwj"] = 8205,
 2911 ["lrm"] = 8206,
 2912 ["rlm"] = 8207,
 2913 ["hyphen"] = 8208,
 2914 ["dash"] = 8208,
 2915 ["ndash"] = 8211,
 2916 ["mdash"] = 8212,
 2917 ["horbar"] = 8213,
 2918 ["Verbar"] = 8214,
 2919 ["Vert"] = 8214,
 2920 ["lsquo"] = 8216,
 2921 ["OpenCurlyQuote"] = 8216,
 2922 ["rsquo"] = 8217,
 2923 ["rsquor"] = 8217,
 2924 ["CloseCurlyQuote"] = 8217,
 2925 ["lsquor"] = 8218,
 2926 ["sbquo"] = 8218,
 2927 ["ldquo"] = 8220,
 2928 ["OpenCurlyDoubleQuote"] = 8220,
 2929 ["rdquo"] = 8221,
 2930 ["rdquor"] = 8221,
 2931 ["CloseCurlyDoubleQuote"] = 8221,
 2932 ["ldquor"] = 8222,
 2933 ["bdquo"] = 8222,
 2934 ["dagger"] = 8224,
 2935 ["Dagger"] = 8225,
 2936 ["ddagger"] = 8225,
 2937 ["bull"] = 8226,
 2938 ["bullet"] = 8226,
 2939 ["nldr"] = 8229,
 2940 ["hellip"] = 8230,
 2941 ["mldr"] = 8230,
 2942 ["permil"] = 8240,
 2943 ["pertenk"] = 8241,
 2944 ["prime"] = 8242,
 2945 ["Prime"] = 8243,
 2946 ["tprime"] = 8244,
 2947 ["bprime"] = 8245,
 2948 ["backprime"] = 8245,
 2949 ["lsaquo"] = 8249,
 2950 ["rsaquo"] = 8250,
 2951 ["oline"] = 8254,
 2952 ["caret"] = 8257,
 2953 ["hybull"] = 8259,
 2954 ["frasl"] = 8260,

```

2955 ["bsemi"] = 8271,
2956 ["qprime"] = 8279,
2957 ["MediumSpace"] = 8287,
2958 ["NoBreak"] = 8288,
2959 ["ApplyFunction"] = 8289,
2960 ["af"] = 8289,
2961 ["InvisibleTimes"] = 8290,
2962 ["it"] = 8290,
2963 ["InvisibleComma"] = 8291,
2964 ["ic"] = 8291,
2965 ["euro"] = 8364,
2966 ["tdot"] = 8411,
2967 ["TripleDot"] = 8411,
2968 ["DotDot"] = 8412,
2969 ["Copf"] = 8450,
2970 ["complexes"] = 8450,
2971 ["incare"] = 8453,
2972 ["gscr"] = 8458,
2973 ["hamilt"] = 8459,
2974 ["HilbertSpace"] = 8459,
2975 ["Hscr"] = 8459,
2976 ["Hfr"] = 8460,
2977 ["Poincareplane"] = 8460,
2978 ["quaternions"] = 8461,
2979 ["Hopf"] = 8461,
2980 ["planckh"] = 8462,
2981 ["planck"] = 8463,
2982 ["hbar"] = 8463,
2983 ["plankv"] = 8463,
2984 ["hslash"] = 8463,
2985 ["Iscr"] = 8464,
2986 ["imagline"] = 8464,
2987 ["image"] = 8465,
2988 ["Im"] = 8465,
2989 ["imagpart"] = 8465,
2990 ["Ifr"] = 8465,
2991 ["Lscr"] = 8466,
2992 ["lagran"] = 8466,
2993 ["Laplacetrif"] = 8466,
2994 ["ell"] = 8467,
2995 ["Nopf"] = 8469,
2996 ["naturals"] = 8469,
2997 ["numero"] = 8470,
2998 ["copyysr"] = 8471,
2999 ["weierp"] = 8472,
3000 ["wp"] = 8472,
3001 ["Popf"] = 8473,

```

```

3002 ["primes"] = 8473,
3003 ["rationals"] = 8474,
3004 ["Qopf"] = 8474,
3005 ["Rscr"] = 8475,
3006 ["realine"] = 8475,
3007 ["real"] = 8476,
3008 ["Re"] = 8476,
3009 ["realpart"] = 8476,
3010 ["Rfr"] = 8476,
3011 ["reals"] = 8477,
3012 ["Ropf"] = 8477,
3013 ["rx"] = 8478,
3014 ["trade"] = 8482,
3015 ["TRADE"] = 8482,
3016 ["integers"] = 8484,
3017 ["Zopf"] = 8484,
3018 ["ohm"] = 8486,
3019 ["mho"] = 8487,
3020 ["Zfr"] = 8488,
3021 ["zeetrf"] = 8488,
3022 ["iiota"] = 8489,
3023 ["angst"] = 8491,
3024 ["bernou"] = 8492,
3025 ["Bernoullis"] = 8492,
3026 ["Bscr"] = 8492,
3027 ["Cfr"] = 8493,
3028 ["Cayleys"] = 8493,
3029 ["escr"] = 8495,
3030 ["Escr"] = 8496,
3031 ["expectation"] = 8496,
3032 ["Fscr"] = 8497,
3033 ["Fouriertrf"] = 8497,
3034 ["phmmat"] = 8499,
3035 ["Mellintrf"] = 8499,
3036 ["Mscr"] = 8499,
3037 ["order"] = 8500,
3038 ["orderof"] = 8500,
3039 ["oscr"] = 8500,
3040 ["alefsym"] = 8501,
3041 ["aleph"] = 8501,
3042 ["beth"] = 8502,
3043 ["gimel"] = 8503,
3044 ["daleth"] = 8504,
3045 ["CapitalDifferentialD"] = 8517,
3046 ["DD"] = 8517,
3047 ["DifferentialD"] = 8518,
3048 ["dd"] = 8518,

```

```

3049 ["ExponentialE"] = 8519,
3050 ["exponentiale"] = 8519,
3051 ["ee"] = 8519,
3052 ["ImaginaryI"] = 8520,
3053 ["ii"] = 8520,
3054 ["frac13"] = 8531,
3055 ["frac23"] = 8532,
3056 ["frac15"] = 8533,
3057 ["frac25"] = 8534,
3058 ["frac35"] = 8535,
3059 ["frac45"] = 8536,
3060 ["frac16"] = 8537,
3061 ["frac56"] = 8538,
3062 ["frac18"] = 8539,
3063 ["frac38"] = 8540,
3064 ["frac58"] = 8541,
3065 ["frac78"] = 8542,
3066 ["larr"] = 8592,
3067 ["leftarrow"] = 8592,
3068 ["LeftArrow"] = 8592,
3069 ["slarr"] = 8592,
3070 ["ShortLeftArrow"] = 8592,
3071 ["uarr"] = 8593,
3072 ["uparrow"] = 8593,
3073 ["UpArrow"] = 8593,
3074 ["ShortUpArrow"] = 8593,
3075 ["rarr"] = 8594,
3076 ["rightarrow"] = 8594,
3077 ["RightArrow"] = 8594,
3078 ["srarr"] = 8594,
3079 ["ShortRightArrow"] = 8594,
3080 ["darr"] = 8595,
3081 ["downarrow"] = 8595,
3082 ["DownArrow"] = 8595,
3083 ["ShortDownArrow"] = 8595,
3084 ["harr"] = 8596,
3085 ["leftrightarrow"] = 8596,
3086 ["LeftRightArrow"] = 8596,
3087 ["varr"] = 8597,
3088 ["updownarrow"] = 8597,
3089 ["UpDownArrow"] = 8597,
3090 ["nwarr"] = 8598,
3091 ["UpperLeftArrow"] = 8598,
3092 ["nwarrow"] = 8598,
3093 ["nearr"] = 8599,
3094 ["UpperRightArrow"] = 8599,
3095 ["nearrow"] = 8599,

```

```

3096 ["searr"] = 8600,
3097 ["searrow"] = 8600,
3098 ["LowerRightArrow"] = 8600,
3099 ["swarr"] = 8601,
3100 ["swarrow"] = 8601,
3101 ["LowerLeftArrow"] = 8601,
3102 ["nlarr"] = 8602,
3103 ["nleftarrow"] = 8602,
3104 ["nrarr"] = 8603,
3105 ["nrightarrow"] = 8603,
3106 ["rarrw"] = 8605,
3107 ["rightsquigarrow"] = 8605,
3108 ["Larr"] = 8606,
3109 ["twoheadleftarrow"] = 8606,
3110 ["Uarr"] = 8607,
3111 ["Rarr"] = 8608,
3112 ["twoheadrightarrow"] = 8608,
3113 ["Darr"] = 8609,
3114 ["larrtl"] = 8610,
3115 ["leftarrowtail"] = 8610,
3116 ["rarrtl"] = 8611,
3117 ["rightarrowtail"] = 8611,
3118 ["LeftTeeArrow"] = 8612,
3119 ["mapstoleft"] = 8612,
3120 ["UpTeeArrow"] = 8613,
3121 ["mapstoup"] = 8613,
3122 ["map"] = 8614,
3123 ["RightTeeArrow"] = 8614,
3124 ["mapsto"] = 8614,
3125 ["DownTeeArrow"] = 8615,
3126 ["mapstodown"] = 8615,
3127 ["larrhk"] = 8617,
3128 ["hookleftarrow"] = 8617,
3129 ["rarrhk"] = 8618,
3130 ["hookrightarrow"] = 8618,
3131 ["larrlp"] = 8619,
3132 ["looparrowleft"] = 8619,
3133 ["rarrlp"] = 8620,
3134 ["looparrowright"] = 8620,
3135 ["harrw"] = 8621,
3136 ["leftrightsquigarrow"] = 8621,
3137 ["nharr"] = 8622,
3138 ["nletrightarrow"] = 8622,
3139 ["lsh"] = 8624,
3140 ["Lsh"] = 8624,
3141 ["rsh"] = 8625,
3142 ["Rsh"] = 8625,

```

```

3143 ["ldsh"] = 8626,
3144 ["rdsh"] = 8627,
3145 ["crarr"] = 8629,
3146 ["cularr"] = 8630,
3147 ["curvearrowleft"] = 8630,
3148 ["curarr"] = 8631,
3149 ["curvearrowright"] = 8631,
3150 ["olarr"] = 8634,
3151 ["circlearrowleft"] = 8634,
3152 ["orarr"] = 8635,
3153 ["circlearrowright"] = 8635,
3154 ["lharu"] = 8636,
3155 ["LeftVector"] = 8636,
3156 ["leftharpoonup"] = 8636,
3157 ["lhard"] = 8637,
3158 ["leftharpoondown"] = 8637,
3159 ["DownLeftVector"] = 8637,
3160 ["uharr"] = 8638,
3161 ["upharpoonright"] = 8638,
3162 ["RightUpVector"] = 8638,
3163 ["uharl"] = 8639,
3164 ["upharpoonleft"] = 8639,
3165 ["LeftUpVector"] = 8639,
3166 ["rharu"] = 8640,
3167 ["RightVector"] = 8640,
3168 ["rightharpoonup"] = 8640,
3169 ["rhard"] = 8641,
3170 ["rightharpoondown"] = 8641,
3171 ["DownRightVector"] = 8641,
3172 ["dharr"] = 8642,
3173 ["RightDownVector"] = 8642,
3174 ["downharpoonright"] = 8642,
3175 ["dharl"] = 8643,
3176 ["LeftDownVector"] = 8643,
3177 ["downharpoonleft"] = 8643,
3178 ["rlarr"] = 8644,
3179 ["rightleftarrows"] = 8644,
3180 ["RightArrowLeftArrow"] = 8644,
3181 ["udarr"] = 8645,
3182 ["UpArrowDownArrow"] = 8645,
3183 ["lrarr"] = 8646,
3184 ["leftrightarrows"] = 8646,
3185 ["LeftArrowRightArrow"] = 8646,
3186 ["llarr"] = 8647,
3187 ["leftleftarrows"] = 8647,
3188 ["uuarr"] = 8648,
3189 ["upuparrows"] = 8648,

```

```

3190 ["rrarr"] = 8649,
3191 ["rightrightarrows"] = 8649,
3192 ["ddarr"] = 8650,
3193 ["downdownarrows"] = 8650,
3194 ["lrhar"] = 8651,
3195 ["ReverseEquilibrium"] = 8651,
3196 ["leftrightharpoons"] = 8651,
3197 ["rlhar"] = 8652,
3198 ["rightleftharpoons"] = 8652,
3199 ["Equilibrium"] = 8652,
3200 ["nlArr"] = 8653,
3201 ["nLeftarrow"] = 8653,
3202 ["nhArr"] = 8654,
3203 ["nLeftrightarrow"] = 8654,
3204 ["nrArr"] = 8655,
3205 ["nRightarrow"] = 8655,
3206 ["lArr"] = 8656,
3207 ["Leftarrow"] = 8656,
3208 ["DoubleLeftArrow"] = 8656,
3209 ["uArr"] = 8657,
3210 ["Uparrow"] = 8657,
3211 ["DoubleUpArrow"] = 8657,
3212 ["rArr"] = 8658,
3213 ["Rightarrow"] = 8658,
3214 ["Implies"] = 8658,
3215 ["DoubleRightArrow"] = 8658,
3216 ["dArr"] = 8659,
3217 ["Downarrow"] = 8659,
3218 ["DoubleDownArrow"] = 8659,
3219 ["hArr"] = 8660,
3220 ["Leftrightarrow"] = 8660,
3221 ["DoubleLeftRightArrow"] = 8660,
3222 ["iff"] = 8660,
3223 ["vArr"] = 8661,
3224 ["Updownarrow"] = 8661,
3225 ["DoubleUpDownArrow"] = 8661,
3226 ["nwArr"] = 8662,
3227 ["neArr"] = 8663,
3228 ["seArr"] = 8664,
3229 ["swArr"] = 8665,
3230 ["lAarr"] = 8666,
3231 ["Lleftarrow"] = 8666,
3232 ["rAarr"] = 8667,
3233 ["Rrightarrow"] = 8667,
3234 ["zigrarr"] = 8669,
3235 ["larrb"] = 8676,
3236 ["LeftArrowBar"] = 8676,

```



```

3237 ["rarrb"] = 8677,
3238 ["RightArrowBar"] = 8677,
3239 ["duarr"] = 8693,
3240 ["DownArrowUpArrow"] = 8693,
3241 ["loarr"] = 8701,
3242 ["roarr"] = 8702,
3243 ["hoarr"] = 8703,
3244 ["forall"] = 8704,
3245 ["ForAll"] = 8704,
3246 ["comp"] = 8705,
3247 ["complement"] = 8705,
3248 ["part"] = 8706,
3249 ["PartialD"] = 8706,
3250 ["exist"] = 8707,
3251 ["Exists"] = 8707,
3252 ["nexist"] = 8708,
3253 ["NotExists"] = 8708,
3254 ["nexists"] = 8708,
3255 ["empty"] = 8709,
3256 ["emptyset"] = 8709,
3257 ["emptyv"] = 8709,
3258 ["varnothing"] = 8709,
3259 ["nabla"] = 8711,
3260 ["Del"] = 8711,
3261 ["isin"] = 8712,
3262 ["isinv"] = 8712,
3263 ["Element"] = 8712,
3264 ["in"] = 8712,
3265 ["notin"] = 8713,
3266 ["NotElement"] = 8713,
3267 ["notinva"] = 8713,
3268 ["niv"] = 8715,
3269 ["ReverseElement"] = 8715,
3270 ["ni"] = 8715,
3271 ["SuchThat"] = 8715,
3272 ["notni"] = 8716,
3273 ["notniva"] = 8716,
3274 ["NotReverseElement"] = 8716,
3275 ["prod"] = 8719,
3276 ["Product"] = 8719,
3277 ["coprod"] = 8720,
3278 ["Coproduct"] = 8720,
3279 ["sum"] = 8721,
3280 ["Sum"] = 8721,
3281 ["minus"] = 8722,
3282 ["mnplus"] = 8723,
3283 ["mp"] = 8723,

```

```

3284 ["MinusPlus"] = 8723,
3285 ["plusdo"] = 8724,
3286 ["dotplus"] = 8724,
3287 ["setmn"] = 8726,
3288 ["setminus"] = 8726,
3289 ["Backslash"] = 8726,
3290 ["ssetmn"] = 8726,
3291 ["smallsetminus"] = 8726,
3292 ["lowast"] = 8727,
3293 ["compfn"] = 8728,
3294 ["SmallCircle"] = 8728,
3295 ["radic"] = 8730,
3296 ["Sqrt"] = 8730,
3297 ["prop"] = 8733,
3298 ["propto"] = 8733,
3299 ["Proportional"] = 8733,
3300 ["vprop"] = 8733,
3301 ["varpropto"] = 8733,
3302 ["infin"] = 8734,
3303 ["angrt"] = 8735,
3304 ["ang"] = 8736,
3305 ["angle"] = 8736,
3306 ["angmsd"] = 8737,
3307 ["measuredangle"] = 8737,
3308 ["angsph"] = 8738,
3309 ["mid"] = 8739,
3310 ["VerticalBar"] = 8739,
3311 ["smid"] = 8739,
3312 ["shortmid"] = 8739,
3313 ["nmid"] = 8740,
3314 ["NotVerticalBar"] = 8740,
3315 ["nsmid"] = 8740,
3316 ["nshortmid"] = 8740,
3317 ["par"] = 8741,
3318 ["parallel"] = 8741,
3319 ["DoubleVerticalBar"] = 8741,
3320 ["spar"] = 8741,
3321 ["shortparallel"] = 8741,
3322 ["npar"] = 8742,
3323 ["nparallel"] = 8742,
3324 ["NotDoubleVerticalBar"] = 8742,
3325 ["nspar"] = 8742,
3326 ["nshortparallel"] = 8742,
3327 ["and"] = 8743,
3328 ["wedge"] = 8743,
3329 ["or"] = 8744,
3330 ["vee"] = 8744,

```

```

3331 ["cap"] = 8745,
3332 ["cup"] = 8746,
3333 ["int"] = 8747,
3334 ["Integral"] = 8747,
3335 ["Int"] = 8748,
3336 ["tint"] = 8749,
3337 ["iiint"] = 8749,
3338 ["conint"] = 8750,
3339 ["oint"] = 8750,
3340 ["ContourIntegral"] = 8750,
3341 ["Conint"] = 8751,
3342 ["DoubleContourIntegral"] = 8751,
3343 ["Cconint"] = 8752,
3344 ["cwint"] = 8753,
3345 ["cwconint"] = 8754,
3346 ["ClockwiseContourIntegral"] = 8754,
3347 ["awconint"] = 8755,
3348 ["CounterClockwiseContourIntegral"] = 8755,
3349 ["there4"] = 8756,
3350 ["therefore"] = 8756,
3351 ["Therefore"] = 8756,
3352 ["becaus"] = 8757,
3353 ["because"] = 8757,
3354 ["Because"] = 8757,
3355 ["ratio"] = 8758,
3356 ["Colon"] = 8759,
3357 ["Proportion"] = 8759,
3358 ["minusd"] = 8760,
3359 ["dotminus"] = 8760,
3360 ["mDDot"] = 8762,
3361 ["homtht"] = 8763,
3362 ["sim"] = 8764,
3363 ["Tilde"] = 8764,
3364 ["thksim"] = 8764,
3365 ["thicksim"] = 8764,
3366 ["bsim"] = 8765,
3367 ["backsim"] = 8765,
3368 ["ac"] = 8766,
3369 ["mstpos"] = 8766,
3370 ["acd"] = 8767,
3371 ["wreath"] = 8768,
3372 ["VerticalTilde"] = 8768,
3373 ["wr"] = 8768,
3374 ["nsim"] = 8769,
3375 ["NotTilde"] = 8769,
3376 ["esim"] = 8770,
3377 ["EqualTilde"] = 8770,

```

```

3378 ["eqsim"] = 8770,
3379 ["sime"] = 8771,
3380 ["TildeEqual"] = 8771,
3381 ["simeq"] = 8771,
3382 ["nsime"] = 8772,
3383 ["nsimeq"] = 8772,
3384 ["NotTildeEqual"] = 8772,
3385 ["cong"] = 8773,
3386 ["TildeFullEqual"] = 8773,
3387 ["simne"] = 8774,
3388 ["ncong"] = 8775,
3389 ["NotTildeFullEqual"] = 8775,
3390 ["asymp"] = 8776,
3391 ["ap"] = 8776,
3392 ["TildeTilde"] = 8776,
3393 ["approx"] = 8776,
3394 ["thkap"] = 8776,
3395 ["thickapprox"] = 8776,
3396 ["nap"] = 8777,
3397 ["NotTildeTilde"] = 8777,
3398 ["napprox"] = 8777,
3399 ["ape"] = 8778,
3400 ["approxseq"] = 8778,
3401 ["apid"] = 8779,
3402 ["bcong"] = 8780,
3403 ["backcong"] = 8780,
3404 ["asympeq"] = 8781,
3405 ["CupCap"] = 8781,
3406 ["bump"] = 8782,
3407 ["HumpDownHump"] = 8782,
3408 ["Bumpeq"] = 8782,
3409 ["bumpe"] = 8783,
3410 ["HumpEqual"] = 8783,
3411 ["bumpeq"] = 8783,
3412 ["esdot"] = 8784,
3413 ["DotEqual"] = 8784,
3414 ["doteq"] = 8784,
3415 ["eDot"] = 8785,
3416 ["doteqdot"] = 8785,
3417 ["efDot"] = 8786,
3418 ["fallingdotseq"] = 8786,
3419 ["erDot"] = 8787,
3420 ["risingdotseq"] = 8787,
3421 ["colone"] = 8788,
3422 ["coloneq"] = 8788,
3423 ["Assign"] = 8788,
3424 ["ecolon"] = 8789,

```

```

3425 ["eqcolon"] = 8789,
3426 ["ecir"] = 8790,
3427 ["eqcirc"] = 8790,
3428 ["cire"] = 8791,
3429 ["circeq"] = 8791,
3430 ["wedgeq"] = 8793,
3431 ["veeeq"] = 8794,
3432 ["trie"] = 8796,
3433 ["triangleq"] = 8796,
3434 ["equest"] = 8799,
3435 ["questeq"] = 8799,
3436 ["ne"] = 8800,
3437 ["NotEqual"] = 8800,
3438 ["equiv"] = 8801,
3439 ["Congruent"] = 8801,
3440 ["nequiv"] = 8802,
3441 ["NotCongruent"] = 8802,
3442 ["le"] = 8804,
3443 ["leq"] = 8804,
3444 ["ge"] = 8805,
3445 ["GreaterEqual"] = 8805,
3446 ["geq"] = 8805,
3447 ["lE"] = 8806,
3448 ["LessFullEqual"] = 8806,
3449 ["leqq"] = 8806,
3450 ["gE"] = 8807,
3451 ["GreaterFullEqual"] = 8807,
3452 ["geqq"] = 8807,
3453 ["lnE"] = 8808,
3454 ["lneqq"] = 8808,
3455 ["gnE"] = 8809,
3456 ["gneqq"] = 8809,
3457 ["Lt"] = 8810,
3458 ["NestedLessLess"] = 8810,
3459 ["ll"] = 8810,
3460 ["Gt"] = 8811,
3461 ["NestedGreaterGreater"] = 8811,
3462 ["gg"] = 8811,
3463 ["twixt"] = 8812,
3464 ["between"] = 8812,
3465 ["NotCupCap"] = 8813,
3466 ["nlt"] = 8814,
3467 ["NotLess"] = 8814,
3468 ["nless"] = 8814,
3469 ["ngt"] = 8815,
3470 ["NotGreater"] = 8815,
3471 ["ngtr"] = 8815,

```

```

3472 ["nle"] = 8816,
3473 ["NotLessEqual"] = 8816,
3474 ["nleq"] = 8816,
3475 ["nge"] = 8817,
3476 ["NotGreaterEqual"] = 8817,
3477 ["ngeq"] = 8817,
3478 ["lsim"] = 8818,
3479 ["LessTilde"] = 8818,
3480 ["lesssim"] = 8818,
3481 ["gsim"] = 8819,
3482 ["gtrsim"] = 8819,
3483 ["GreaterTilde"] = 8819,
3484 ["nlsim"] = 8820,
3485 ["NotLessTilde"] = 8820,
3486 ["ngsim"] = 8821,
3487 ["NotGreaterTilde"] = 8821,
3488 ["lg"] = 8822,
3489 ["lessgtr"] = 8822,
3490 ["LessGreater"] = 8822,
3491 ["gl"] = 8823,
3492 ["gtrless"] = 8823,
3493 ["GreaterLess"] = 8823,
3494 ["ntlg"] = 8824,
3495 ["NotLessGreater"] = 8824,
3496 ["ntgl"] = 8825,
3497 ["NotGreaterLess"] = 8825,
3498 ["pr"] = 8826,
3499 ["Precedes"] = 8826,
3500 ["prec"] = 8826,
3501 ["sc"] = 8827,
3502 ["Succeeds"] = 8827,
3503 ["succ"] = 8827,
3504 ["prcue"] = 8828,
3505 ["PrecedesSlantEqual"] = 8828,
3506 ["preccurlyeq"] = 8828,
3507 ["sccue"] = 8829,
3508 ["SucceedsSlantEqual"] = 8829,
3509 ["succcurlyeq"] = 8829,
3510 ["prsim"] = 8830,
3511 ["precsim"] = 8830,
3512 ["PrecedesTilde"] = 8830,
3513 ["scsim"] = 8831,
3514 ["succsim"] = 8831,
3515 ["SucceedsTilde"] = 8831,
3516 ["npr"] = 8832,
3517 ["nprec"] = 8832,
3518 ["NotPrecedes"] = 8832,

```

```

3519 ["nsc"] = 8833,
3520 ["nsucc"] = 8833,
3521 ["NotSucceeds"] = 8833,
3522 ["sub"] = 8834,
3523 ["subset"] = 8834,
3524 ["sup"] = 8835,
3525 ["supset"] = 8835,
3526 ["Superset"] = 8835,
3527 ["nsub"] = 8836,
3528 ["nsup"] = 8837,
3529 ["sube"] = 8838,
3530 ["SubsetEqual"] = 8838,
3531 ["subseteq"] = 8838,
3532 ["supe"] = 8839,
3533 ["supseteq"] = 8839,
3534 ["SupersetEqual"] = 8839,
3535 ["nsube"] = 8840,
3536 ["nsubseteq"] = 8840,
3537 ["NotSubsetEqual"] = 8840,
3538 ["nsupe"] = 8841,
3539 ["nsupseteq"] = 8841,
3540 ["NotSupersetEqual"] = 8841,
3541 ["subne"] = 8842,
3542 ["subsetneq"] = 8842,
3543 ["supne"] = 8843,
3544 ["supsetneq"] = 8843,
3545 ["cupdot"] = 8845,
3546 ["uplus"] = 8846,
3547 ["UnionPlus"] = 8846,
3548 ["sqsub"] = 8847,
3549 ["SquareSubset"] = 8847,
3550 ["sqsubset"] = 8847,
3551 ["sqsup"] = 8848,
3552 ["SquareSuperset"] = 8848,
3553 ["sqsupset"] = 8848,
3554 ["sqsube"] = 8849,
3555 ["SquareSubsetEqual"] = 8849,
3556 ["sqsubseteq"] = 8849,
3557 ["sqsupe"] = 8850,
3558 ["SquareSupersetEqual"] = 8850,
3559 ["sqsupseteq"] = 8850,
3560 ["sqcap"] = 8851,
3561 ["SquareIntersection"] = 8851,
3562 ["sqcup"] = 8852,
3563 ["SquareUnion"] = 8852,
3564 ["oplus"] = 8853,
3565 ["CirclePlus"] = 8853,

```

```

3566 ["ominus"] = 8854,
3567 ["CircleMinus"] = 8854,
3568 ["otimes"] = 8855,
3569 ["CircleTimes"] = 8855,
3570 ["osol"] = 8856,
3571 ["odot"] = 8857,
3572 ["CircleDot"] = 8857,
3573 ["ocir"] = 8858,
3574 ["circledcirc"] = 8858,
3575 ["oast"] = 8859,
3576 ["circledast"] = 8859,
3577 ["odash"] = 8861,
3578 ["circleddash"] = 8861,
3579 ["plusb"] = 8862,
3580 ["boxplus"] = 8862,
3581 ["minusb"] = 8863,
3582 ["boxminus"] = 8863,
3583 ["timesb"] = 8864,
3584 ["boxtimes"] = 8864,
3585 ["sdotb"] = 8865,
3586 ["dotsquare"] = 8865,
3587 ["vdash"] = 8866,
3588 ["RightTee"] = 8866,
3589 ["dashv"] = 8867,
3590 ["LeftTee"] = 8867,
3591 ["top"] = 8868,
3592 ["DownTee"] = 8868,
3593 ["bottom"] = 8869,
3594 ["bot"] = 8869,
3595 ["perp"] = 8869,
3596 ["UpTee"] = 8869,
3597 ["models"] = 8871,
3598 ["vDash"] = 8872,
3599 ["DoubleRightTee"] = 8872,
3600 ["Vdash"] = 8873,
3601 ["Vvdash"] = 8874,
3602 ["VDash"] = 8875,
3603 ["nvdash"] = 8876,
3604 ["nvDash"] = 8877,
3605 ["nVdash"] = 8878,
3606 ["nVDash"] = 8879,
3607 ["prurel"] = 8880,
3608 ["vltri"] = 8882,
3609 ["vartriangleleft"] = 8882,
3610 ["LeftTriangle"] = 8882,
3611 ["vrtri"] = 8883,
3612 ["vartriangleright"] = 8883,

```



```

3613 ["RightTriangle"] = 8883,
3614 ["ltrie"] = 8884,
3615 ["trianglelefteq"] = 8884,
3616 ["LeftTriangleEqual"] = 8884,
3617 ["rtrie"] = 8885,
3618 ["trianglerighteq"] = 8885,
3619 ["RightTriangleEqual"] = 8885,
3620 ["origof"] = 8886,
3621 ["imof"] = 8887,
3622 ["mumap"] = 8888,
3623 ["multimap"] = 8888,
3624 ["hercon"] = 8889,
3625 ["intcal"] = 8890,
3626 ["intercal"] = 8890,
3627 ["veebar"] = 8891,
3628 ["barvee"] = 8893,
3629 ["angrtvb"] = 8894,
3630 ["lrtri"] = 8895,
3631 ["xwedge"] = 8896,
3632 ["Wedge"] = 8896,
3633 ["bigwedge"] = 8896,
3634 ["xvee"] = 8897,
3635 ["Vee"] = 8897,
3636 ["bigvee"] = 8897,
3637 ["xcap"] = 8898,
3638 ["Intersection"] = 8898,
3639 ["bigcap"] = 8898,
3640 ["xcup"] = 8899,
3641 ["Union"] = 8899,
3642 ["bigcup"] = 8899,
3643 ["diam"] = 8900,
3644 ["diamond"] = 8900,
3645 ["Diamond"] = 8900,
3646 ["sdot"] = 8901,
3647 ["sstarf"] = 8902,
3648 ["Star"] = 8902,
3649 ["divonx"] = 8903,
3650 ["divideontimes"] = 8903,
3651 ["bowtie"] = 8904,
3652 ["ltimes"] = 8905,
3653 ["rtimes"] = 8906,
3654 ["lthree"] = 8907,
3655 ["leftthreetimes"] = 8907,
3656 ["rthree"] = 8908,
3657 ["rightthreetimes"] = 8908,
3658 ["bsime"] = 8909,
3659 ["backsimeq"] = 8909,

```

```

3660 ["cuvee"] = 8910,
3661 ["curlyvee"] = 8910,
3662 ["cuwed"] = 8911,
3663 ["curlywedge"] = 8911,
3664 ["Sub"] = 8912,
3665 ["Subset"] = 8912,
3666 ["Sup"] = 8913,
3667 ["Supset"] = 8913,
3668 ["Cap"] = 8914,
3669 ["Cup"] = 8915,
3670 ["fork"] = 8916,
3671 ["pitchfork"] = 8916,
3672 ["epar"] = 8917,
3673 ["ltdot"] = 8918,
3674 ["lessdot"] = 8918,
3675 ["gtdot"] = 8919,
3676 ["gtrdot"] = 8919,
3677 ["Ll"] = 8920,
3678 ["Gg"] = 8921,
3679 ["ggg"] = 8921,
3680 ["leg"] = 8922,
3681 ["LessEqualGreater"] = 8922,
3682 ["lesseqgtr"] = 8922,
3683 ["gel"] = 8923,
3684 ["gtreqless"] = 8923,
3685 ["GreaterEqualLess"] = 8923,
3686 ["cuepr"] = 8926,
3687 ["curlyeqprec"] = 8926,
3688 ["cuesc"] = 8927,
3689 ["curlyeqsucc"] = 8927,
3690 ["nprcue"] = 8928,
3691 ["NotPrecedesSlantEqual"] = 8928,
3692 ["nsccue"] = 8929,
3693 ["NotSucceedsSlantEqual"] = 8929,
3694 ["nsqsube"] = 8930,
3695 ["NotSquareSubsetEqual"] = 8930,
3696 ["nsqsupe"] = 8931,
3697 ["NotSquareSupersetEqual"] = 8931,
3698 ["lnsim"] = 8934,
3699 ["gnsim"] = 8935,
3700 ["prnsim"] = 8936,
3701 ["precnsim"] = 8936,
3702 ["scnsim"] = 8937,
3703 ["succnsim"] = 8937,
3704 ["nltri"] = 8938,
3705 ["ntriangleleft"] = 8938,
3706 ["NotLeftTriangle"] = 8938,

```

```

3707 ["nrtri"] = 8939,
3708 ["ntriangleright"] = 8939,
3709 ["NotRightTriangle"] = 8939,
3710 ["nltrie"] = 8940,
3711 ["ntrianglelefteq"] = 8940,
3712 ["NotLeftTriangleEqual"] = 8940,
3713 ["nrtrie"] = 8941,
3714 ["ntrianglerighteq"] = 8941,
3715 ["NotRightTriangleEqual"] = 8941,
3716 ["vellip"] = 8942,
3717 ["ctdot"] = 8943,
3718 ["utdot"] = 8944,
3719 ["dtdot"] = 8945,
3720 ["disin"] = 8946,
3721 ["isinsv"] = 8947,
3722 ["isins"] = 8948,
3723 ["isindot"] = 8949,
3724 ["notinvc"] = 8950,
3725 ["notinvb"] = 8951,
3726 ["isinE"] = 8953,
3727 ["nisd"] = 8954,
3728 ["xnis"] = 8955,
3729 ["nis"] = 8956,
3730 ["notnivc"] = 8957,
3731 ["notnivb"] = 8958,
3732 ["barwed"] = 8965,
3733 ["barwedge"] = 8965,
3734 ["Barwed"] = 8966,
3735 ["doublebarwedge"] = 8966,
3736 ["lceil"] = 8968,
3737 ["LeftCeiling"] = 8968,
3738 ["rceil"] = 8969,
3739 ["RightCeiling"] = 8969,
3740 ["lfloor"] = 8970,
3741 ["LeftFloor"] = 8970,
3742 ["rfloor"] = 8971,
3743 ["RightFloor"] = 8971,
3744 ["drcrop"] = 8972,
3745 ["dlcrop"] = 8973,
3746 ["urcrop"] = 8974,
3747 ["ulcrop"] = 8975,
3748 ["bnot"] = 8976,
3749 ["proflin"] = 8978,
3750 ["profsurf"] = 8979,
3751 ["telrec"] = 8981,
3752 ["target"] = 8982,
3753 ["ulcorn"] = 8988,

```

```

3754 ["ulcorner"] = 8988,
3755 ["urcorn"] = 8989,
3756 ["urcorner"] = 8989,
3757 ["dlcorn"] = 8990,
3758 ["llcorner"] = 8990,
3759 ["drcorn"] = 8991,
3760 ["lrcorner"] = 8991,
3761 ["frown"] = 8994,
3762 ["sfrown"] = 8994,
3763 ["smile"] = 8995,
3764 ["ssmile"] = 8995,
3765 ["cylcty"] = 9005,
3766 ["profalar"] = 9006,
3767 ["topbot"] = 9014,
3768 ["ovbar"] = 9021,
3769 ["solbar"] = 9023,
3770 ["angzarr"] = 9084,
3771 ["lmoust"] = 9136,
3772 ["lmoustache"] = 9136,
3773 ["rmoust"] = 9137,
3774 ["rmoustache"] = 9137,
3775 ["tbrk"] = 9140,
3776 ["OverBracket"] = 9140,
3777 ["bbrk"] = 9141,
3778 ["UnderBracket"] = 9141,
3779 ["bbrktbrk"] = 9142,
3780 ["OverParenthesis"] = 9180,
3781 ["UnderParenthesis"] = 9181,
3782 ["OverBrace"] = 9182,
3783 ["UnderBrace"] = 9183,
3784 ["trpezium"] = 9186,
3785 ["elinters"] = 9191,
3786 ["blank"] = 9251,
3787 ["oS"] = 9416,
3788 ["circledS"] = 9416,
3789 ["boxh"] = 9472,
3790 ["HorizontalLine"] = 9472,
3791 ["boxv"] = 9474,
3792 ["boxdr"] = 9484,
3793 ["boxdl"] = 9488,
3794 ["boxur"] = 9492,
3795 ["boxul"] = 9496,
3796 ["boxvr"] = 9500,
3797 ["boxvl"] = 9508,
3798 ["boxhd"] = 9516,
3799 ["boxhu"] = 9524,
3800 ["boxvh"] = 9532,

```

```

3801 ["boxH"] = 9552,
3802 ["boxV"] = 9553,
3803 ["boxdR"] = 9554,
3804 ["boxDr"] = 9555,
3805 ["boxDR"] = 9556,
3806 ["boxdL"] = 9557,
3807 ["boxDl"] = 9558,
3808 ["boxDL"] = 9559,
3809 ["boxuR"] = 9560,
3810 ["boxUr"] = 9561,
3811 ["boxUR"] = 9562,
3812 ["boxuL"] = 9563,
3813 ["boxUl"] = 9564,
3814 ["boxUL"] = 9565,
3815 ["boxvR"] = 9566,
3816 ["boxVr"] = 9567,
3817 ["boxVR"] = 9568,
3818 ["boxvL"] = 9569,
3819 ["boxVl"] = 9570,
3820 ["boxVL"] = 9571,
3821 ["boxHd"] = 9572,
3822 ["boxhD"] = 9573,
3823 ["boxHD"] = 9574,
3824 ["boxHu"] = 9575,
3825 ["boxhU"] = 9576,
3826 ["boxHU"] = 9577,
3827 ["boxvH"] = 9578,
3828 ["boxVh"] = 9579,
3829 ["boxVH"] = 9580,
3830 ["uhblk"] = 9600,
3831 ["lhblk"] = 9604,
3832 ["block"] = 9608,
3833 ["blk14"] = 9617,
3834 ["blk12"] = 9618,
3835 ["blk34"] = 9619,
3836 ["squ"] = 9633,
3837 ["square"] = 9633,
3838 ["Square"] = 9633,
3839 ["squf"] = 9642,
3840 ["squarf"] = 9642,
3841 ["blacksquare"] = 9642,
3842 ["FilledVerySmallSquare"] = 9642,
3843 ["EmptyVerySmallSquare"] = 9643,
3844 ["rect"] = 9645,
3845 ["marker"] = 9646,
3846 ["fltns"] = 9649,
3847 ["xutri"] = 9651,

```

```

3848 ["bigtriangleup"] = 9651,
3849 ["utrif"] = 9652,
3850 ["blacktriangle"] = 9652,
3851 ["utri"] = 9653,
3852 ["triangle"] = 9653,
3853 ["rtrif"] = 9656,
3854 ["blacktriangleright"] = 9656,
3855 ["rtri"] = 9657,
3856 ["triangleright"] = 9657,
3857 ["xdtri"] = 9661,
3858 ["bigtriangledown"] = 9661,
3859 ["dtrif"] = 9662,
3860 ["blacktriangledown"] = 9662,
3861 ["dtri"] = 9663,
3862 ["triangledown"] = 9663,
3863 ["ltrif"] = 9666,
3864 ["blacktriangleleft"] = 9666,
3865 ["ltri"] = 9667,
3866 ["triangleleft"] = 9667,
3867 ["loz"] = 9674,
3868 ["lozenge"] = 9674,
3869 ["cir"] = 9675,
3870 ["tridot"] = 9708,
3871 ["xcirc"] = 9711,
3872 ["bigcirc"] = 9711,
3873 ["ultri"] = 9720,
3874 ["urtri"] = 9721,
3875 ["lltri"] = 9722,
3876 ["EmptySmallSquare"] = 9723,
3877 ["FilledSmallSquare"] = 9724,
3878 ["starf"] = 9733,
3879 ["bigstar"] = 9733,
3880 ["star"] = 9734,
3881 ["phone"] = 9742,
3882 ["female"] = 9792,
3883 ["male"] = 9794,
3884 ["spades"] = 9824,
3885 ["spadesuit"] = 9824,
3886 ["clubs"] = 9827,
3887 ["clubsuit"] = 9827,
3888 ["hearts"] = 9829,
3889 ["heartsuit"] = 9829,
3890 ["diams"] = 9830,
3891 ["diamondsuit"] = 9830,
3892 ["sung"] = 9834,
3893 ["flat"] = 9837,
3894 ["natur"] = 9838,

```

```

3895 ["natural"] = 9838,
3896 ["sharp"] = 9839,
3897 ["check"] = 10003,
3898 ["checkmark"] = 10003,
3899 ["cross"] = 10007,
3900 ["malt"] = 10016,
3901 ["maltese"] = 10016,
3902 ["sext"] = 10038,
3903 ["VerticalSeparator"] = 10072,
3904 ["lbrk"] = 10098,
3905 ["rbrk"] = 10099,
3906 ["lobrk"] = 10214,
3907 ["LeftDoubleBracket"] = 10214,
3908 ["robrk"] = 10215,
3909 ["RightDoubleBracket"] = 10215,
3910 ["lang"] = 10216,
3911 ["LeftAngleBracket"] = 10216,
3912 ["langle"] = 10216,
3913 ["rang"] = 10217,
3914 ["RightAngleBracket"] = 10217,
3915 ["rangle"] = 10217,
3916 ["Lang"] = 10218,
3917 ["Rang"] = 10219,
3918 ["loang"] = 10220,
3919 ["roang"] = 10221,
3920 ["xlarr"] = 10229,
3921 ["longleftarrow"] = 10229,
3922 ["LongLeftArrow"] = 10229,
3923 ["xrarr"] = 10230,
3924 ["longrightarrow"] = 10230,
3925 ["LongRightArrow"] = 10230,
3926 ["xharr"] = 10231,
3927 ["longleftrightharrow"] = 10231,
3928 ["LongLeftRightArrow"] = 10231,
3929 ["xlArr"] = 10232,
3930 ["Longleftarrow"] = 10232,
3931 ["DoubleLongLeftArrow"] = 10232,
3932 ["xrArr"] = 10233,
3933 ["Longrightarrow"] = 10233,
3934 ["DoubleLongRightArrow"] = 10233,
3935 ["xhArr"] = 10234,
3936 ["Longleftrightharrow"] = 10234,
3937 ["DoubleLongLeftRightArrow"] = 10234,
3938 ["xmap"] = 10236,
3939 ["longmapsto"] = 10236,
3940 ["dzigrarr"] = 10239,
3941 ["nvlArr"] = 10498,

```

```

3942 ["nvrArr"] = 10499,
3943 ["nvHarr"] = 10500,
3944 ["Map"] = 10501,
3945 ["lbarr"] = 10508,
3946 ["rbarr"] = 10509,
3947 ["bkarow"] = 10509,
3948 ["lBarr"] = 10510,
3949 ["rBarr"] = 10511,
3950 ["dbkarow"] = 10511,
3951 ["RBarr"] = 10512,
3952 ["drbkarow"] = 10512,
3953 ["DDotrahd"] = 10513,
3954 ["UpArrowBar"] = 10514,
3955 ["DownArrowBar"] = 10515,
3956 ["Rarrtl"] = 10518,
3957 ["latail"] = 10521,
3958 ["ratail"] = 10522,
3959 ["lAtail"] = 10523,
3960 ["rAtail"] = 10524,
3961 ["larrfs"] = 10525,
3962 ["rarrfs"] = 10526,
3963 ["larrbfs"] = 10527,
3964 ["rarrbfs"] = 10528,
3965 ["nwarhk"] = 10531,
3966 ["nearhk"] = 10532,
3967 ["searhk"] = 10533,
3968 ["hksearow"] = 10533,
3969 ["swarhk"] = 10534,
3970 ["hkswarow"] = 10534,
3971 ["nwnear"] = 10535,
3972 ["nesear"] = 10536,
3973 ["toea"] = 10536,
3974 ["seswar"] = 10537,
3975 ["tosa"] = 10537,
3976 ["swnwar"] = 10538,
3977 ["rarrc"] = 10547,
3978 ["cudarr"] = 10549,
3979 ["ldca"] = 10550,
3980 ["rdca"] = 10551,
3981 ["cudarrrl"] = 10552,
3982 ["larrpl"] = 10553,
3983 ["curarrm"] = 10556,
3984 ["cularrp"] = 10557,
3985 ["rarrpl"] = 10565,
3986 ["harrcir"] = 10568,
3987 ["Uarroccir"] = 10569,
3988 ["lurdshar"] = 10570,

```



```

3989 ["ldrushar"] = 10571,
3990 ["LeftRightVector"] = 10574,
3991 ["RightUpDownVector"] = 10575,
3992 ["DownLeftRightVector"] = 10576,
3993 ["LeftUpDownVector"] = 10577,
3994 ["LeftVectorBar"] = 10578,
3995 ["RightVectorBar"] = 10579,
3996 ["RightUpVectorBar"] = 10580,
3997 ["RightDownVectorBar"] = 10581,
3998 ["DownLeftVectorBar"] = 10582,
3999 ["DownRightVectorBar"] = 10583,
4000 ["LeftUpVectorBar"] = 10584,
4001 ["LeftDownVectorBar"] = 10585,
4002 ["LeftTeeVector"] = 10586,
4003 ["RightTeeVector"] = 10587,
4004 ["RightUpTeeVector"] = 10588,
4005 ["RightDownTeeVector"] = 10589,
4006 ["DownLeftTeeVector"] = 10590,
4007 ["DownRightTeeVector"] = 10591,
4008 ["LeftUpTeeVector"] = 10592,
4009 ["LeftDownTeeVector"] = 10593,
4010 ["lHar"] = 10594,
4011 ["uHar"] = 10595,
4012 ["rHar"] = 10596,
4013 ["dHar"] = 10597,
4014 ["luruhar"] = 10598,
4015 ["ldrdhar"] = 10599,
4016 ["ruluhar"] = 10600,
4017 ["rdldhar"] = 10601,
4018 ["lharul"] = 10602,
4019 ["llhard"] = 10603,
4020 ["rharul"] = 10604,
4021 ["lrhard"] = 10605,
4022 ["udhar"] = 10606,
4023 ["UpEquilibrium"] = 10606,
4024 ["duhar"] = 10607,
4025 ["ReverseUpEquilibrium"] = 10607,
4026 ["RoundImplies"] = 10608,
4027 ["erarr"] = 10609,
4028 ["simrarr"] = 10610,
4029 ["larrsim"] = 10611,
4030 ["rarrsim"] = 10612,
4031 ["rarrap"] = 10613,
4032 ["ltlarr"] = 10614,
4033 ["gtrarr"] = 10616,
4034 ["subrarr"] = 10617,
4035 ["suplarr"] = 10619,

```

```

4036 ["lfisht"] = 10620,
4037 ["rfisht"] = 10621,
4038 ["ufisht"] = 10622,
4039 ["dfisht"] = 10623,
4040 ["lopar"] = 10629,
4041 ["ropar"] = 10630,
4042 ["lbrke"] = 10635,
4043 ["rbrke"] = 10636,
4044 ["lbrkslu"] = 10637,
4045 ["rbrksld"] = 10638,
4046 ["lbrksld"] = 10639,
4047 ["rbrkslu"] = 10640,
4048 ["langd"] = 10641,
4049 ["rangd"] = 10642,
4050 ["lparlt"] = 10643,
4051 ["rpargt"] = 10644,
4052 ["gtlPar"] = 10645,
4053 ["ltrPar"] = 10646,
4054 ["vzigzag"] = 10650,
4055 ["vangrt"] = 10652,
4056 ["angrtvbd"] = 10653,
4057 ["ange"] = 10660,
4058 ["range"] = 10661,
4059 ["dwangle"] = 10662,
4060 ["uwangle"] = 10663,
4061 ["angmsdaa"] = 10664,
4062 ["angmsdab"] = 10665,
4063 ["angmsdac"] = 10666,
4064 ["angmsdad"] = 10667,
4065 ["angmsdae"] = 10668,
4066 ["angmsdaf"] = 10669,
4067 ["angmsdag"] = 10670,
4068 ["angmsdah"] = 10671,
4069 ["bemptyv"] = 10672,
4070 ["demptyv"] = 10673,
4071 ["cemptyv"] = 10674,
4072 ["raemptyv"] = 10675,
4073 ["laemptyv"] = 10676,
4074 ["ohbar"] = 10677,
4075 ["omid"] = 10678,
4076 ["opar"] = 10679,
4077 ["operp"] = 10681,
4078 ["olcross"] = 10683,
4079 ["odsold"] = 10684,
4080 ["olcir"] = 10686,
4081 ["ofcir"] = 10687,
4082 ["olt"] = 10688,

```

```

4083 ["ogt"] = 10689,
4084 ["cirscir"] = 10690,
4085 ["cirE"] = 10691,
4086 ["solb"] = 10692,
4087 ["bsolb"] = 10693,
4088 ["boxbox"] = 10697,
4089 ["trisb"] = 10701,
4090 ["rtriltri"] = 10702,
4091 ["LeftTriangleBar"] = 10703,
4092 ["RightTriangleBar"] = 10704,
4093 ["race"] = 10714,
4094 ["iinfin"] = 10716,
4095 ["infintie"] = 10717,
4096 ["nvinfin"] = 10718,
4097 ["eparsl"] = 10723,
4098 ["smeparsl"] = 10724,
4099 ["eqvparsl"] = 10725,
4100 ["lozf"] = 10731,
4101 ["blacklozenge"] = 10731,
4102 ["RuleDelayed"] = 10740,
4103 ["dsol"] = 10742,
4104 ["xodot"] = 10752,
4105 ["bigodot"] = 10752,
4106 ["xoplus"] = 10753,
4107 ["bigoplus"] = 10753,
4108 ["xotime"] = 10754,
4109 ["bigotimes"] = 10754,
4110 ["xuplus"] = 10756,
4111 ["biguplus"] = 10756,
4112 ["xsqcup"] = 10758,
4113 ["bigsqcup"] = 10758,
4114 ["qint"] = 10764,
4115 ["iiiint"] = 10764,
4116 ["fpartint"] = 10765,
4117 ["cirfnint"] = 10768,
4118 ["awint"] = 10769,
4119 ["rppointint"] = 10770,
4120 ["scpointint"] = 10771,
4121 ["npointint"] = 10772,
4122 ["pointint"] = 10773,
4123 ["quatint"] = 10774,
4124 ["intlarhk"] = 10775,
4125 ["pluscir"] = 10786,
4126 ["plusacir"] = 10787,
4127 ["simplus"] = 10788,
4128 ["plusdu"] = 10789,
4129 ["plussim"] = 10790,

```

```

4130 ["plustwo"] = 10791,
4131 ["mcomma"] = 10793,
4132 ["minusdu"] = 10794,
4133 ["loplus"] = 10797,
4134 ["roplus"] = 10798,
4135 ["Cross"] = 10799,
4136 ["timesd"] = 10800,
4137 ["timesbar"] = 10801,
4138 ["smashp"] = 10803,
4139 ["lotimes"] = 10804,
4140 ["rotimes"] = 10805,
4141 ["otimesas"] = 10806,
4142 ["Otimes"] = 10807,
4143 ["odiv"] = 10808,
4144 ["triplus"] = 10809,
4145 ["triminus"] = 10810,
4146 ["tritime"] = 10811,
4147 ["iprod"] = 10812,
4148 ["intprod"] = 10812,
4149 ["amalg"] = 10815,
4150 ["capdot"] = 10816,
4151 ["ncup"] = 10818,
4152 ["ncap"] = 10819,
4153 ["capand"] = 10820,
4154 ["cupor"] = 10821,
4155 ["cupcap"] = 10822,
4156 ["capcup"] = 10823,
4157 ["cupbrcap"] = 10824,
4158 ["capbrcup"] = 10825,
4159 ["cupcup"] = 10826,
4160 ["capcap"] = 10827,
4161 ["ccups"] = 10828,
4162 ["ccaps"] = 10829,
4163 ["ccupssm"] = 10832,
4164 ["And"] = 10835,
4165 ["Or"] = 10836,
4166 ["andand"] = 10837,
4167 ["oror"] = 10838,
4168 ["orslope"] = 10839,
4169 ["andslope"] = 10840,
4170 ["andv"] = 10842,
4171 ["orv"] = 10843,
4172 ["andd"] = 10844,
4173 ["ord"] = 10845,
4174 ["wedbar"] = 10847,
4175 ["sdote"] = 10854,
4176 ["simdot"] = 10858,

```

```

4177 ["congdot"] = 10861,
4178 ["easter"] = 10862,
4179 ["apacir"] = 10863,
4180 ["apE"] = 10864,
4181 ["eplus"] = 10865,
4182 ["pluse"] = 10866,
4183 ["Esim"] = 10867,
4184 ["Colone"] = 10868,
4185 ["Equal"] = 10869,
4186 ["eDDot"] = 10871,
4187 ["ddotseq"] = 10871,
4188 ["equivDD"] = 10872,
4189 ["ltcir"] = 10873,
4190 ["gtcir"] = 10874,
4191 ["ltquest"] = 10875,
4192 ["gtquest"] = 10876,
4193 ["les"] = 10877,
4194 ["LessSlantEqual"] = 10877,
4195 ["leqslant"] = 10877,
4196 ["ges"] = 10878,
4197 ["GreaterSlantEqual"] = 10878,
4198 ["geqslant"] = 10878,
4199 ["lesdot"] = 10879,
4200 ["gesdot"] = 10880,
4201 ["lesdoto"] = 10881,
4202 ["gesdoto"] = 10882,
4203 ["lesdotor"] = 10883,
4204 ["gesdoto1"] = 10884,
4205 ["lap"] = 10885,
4206 ["lessapprox"] = 10885,
4207 ["gap"] = 10886,
4208 ["gtrapprox"] = 10886,
4209 ["lne"] = 10887,
4210 ["lneq"] = 10887,
4211 ["gne"] = 10888,
4212 ["gneq"] = 10888,
4213 ["lnap"] = 10889,
4214 ["lnapprox"] = 10889,
4215 ["gnap"] = 10890,
4216 ["gnapprox"] = 10890,
4217 ["lEg"] = 10891,
4218 ["lesseqqgtr"] = 10891,
4219 ["gEl"] = 10892,
4220 ["gtreqqless"] = 10892,
4221 ["lsime"] = 10893,
4222 ["gsime"] = 10894,
4223 ["lsimg"] = 10895,

```

```

4224 ["gsiml"] = 10896,
4225 ["lgE"] = 10897,
4226 ["glE"] = 10898,
4227 ["lesges"] = 10899,
4228 ["gesles"] = 10900,
4229 ["els"] = 10901,
4230 ["eqslantless"] = 10901,
4231 ["egs"] = 10902,
4232 ["eqslantgtr"] = 10902,
4233 ["elsdot"] = 10903,
4234 ["egsdot"] = 10904,
4235 ["el"] = 10905,
4236 ["eg"] = 10906,
4237 ["siml"] = 10909,
4238 ["simg"] = 10910,
4239 ["simlE"] = 10911,
4240 ["simgE"] = 10912,
4241 ["LessLess"] = 10913,
4242 ["GreaterGreater"] = 10914,
4243 ["glj"] = 10916,
4244 ["gla"] = 10917,
4245 ["ltcc"] = 10918,
4246 ["gtcc"] = 10919,
4247 ["lescc"] = 10920,
4248 ["gescc"] = 10921,
4249 ["smt"] = 10922,
4250 ["lat"] = 10923,
4251 ["smtE"] = 10924,
4252 ["late"] = 10925,
4253 ["bumpE"] = 10926,
4254 ["pre"] = 10927,
4255 ["preceq"] = 10927,
4256 ["PrecedesEqual"] = 10927,
4257 ["sce"] = 10928,
4258 ["succeq"] = 10928,
4259 ["SucceedsEqual"] = 10928,
4260 ["prE"] = 10931,
4261 ["scE"] = 10932,
4262 ["prnE"] = 10933,
4263 ["precneqq"] = 10933,
4264 ["scnE"] = 10934,
4265 ["succneqq"] = 10934,
4266 ["prap"] = 10935,
4267 ["precapprox"] = 10935,
4268 ["scap"] = 10936,
4269 ["succapprox"] = 10936,
4270 ["prnap"] = 10937,

```

```

4271 ["precnapprox"] = 10937,
4272 ["scnap"] = 10938,
4273 ["succnapprox"] = 10938,
4274 ["Pr"] = 10939,
4275 ["Sc"] = 10940,
4276 ["subdot"] = 10941,
4277 ["supdot"] = 10942,
4278 ["subplus"] = 10943,
4279 ["supplus"] = 10944,
4280 ["submult"] = 10945,
4281 ["supmult"] = 10946,
4282 ["subedot"] = 10947,
4283 ["supedot"] = 10948,
4284 ["subE"] = 10949,
4285 ["subseteqq"] = 10949,
4286 ["supE"] = 10950,
4287 ["supseteqq"] = 10950,
4288 ["subsim"] = 10951,
4289 ["supsim"] = 10952,
4290 ["subnE"] = 10955,
4291 ["subsetneqq"] = 10955,
4292 ["supnE"] = 10956,
4293 ["supsetneqq"] = 10956,
4294 ["csub"] = 10959,
4295 ["csup"] = 10960,
4296 ["csube"] = 10961,
4297 ["csupe"] = 10962,
4298 ["subsup"] = 10963,
4299 ["supsub"] = 10964,
4300 ["subsub"] = 10965,
4301 ["supsup"] = 10966,
4302 ["suphsub"] = 10967,
4303 ["supdsub"] = 10968,
4304 ["forkv"] = 10969,
4305 ["topfork"] = 10970,
4306 ["mlcp"] = 10971,
4307 ["Dashv"] = 10980,
4308 ["DoubleLeftTee"] = 10980,
4309 ["Vdashl"] = 10982,
4310 ["Barv"] = 10983,
4311 ["vBar"] = 10984,
4312 ["vBarv"] = 10985,
4313 ["Vbar"] = 10987,
4314 ["Not"] = 10988,
4315 ["bNot"] = 10989,
4316 ["rnmid"] = 10990,
4317 ["cirmid"] = 10991,

```

```

4318 ["midcir"] = 10992,
4319 ["topcir"] = 10993,
4320 ["nhpar"] = 10994,
4321 ["parsim"] = 10995,
4322 ["parsl"] = 11005,
4323 ["fflig"] = 64256,
4324 ["filig"] = 64257,
4325 ["fllig"] = 64258,
4326 ["ffilig"] = 64259,
4327 ["ffllig"] = 64260,
4328 ["Ascr"] = 119964,
4329 ["Cscr"] = 119966,
4330 ["Dscr"] = 119967,
4331 ["Gscr"] = 119970,
4332 ["Jscr"] = 119973,
4333 ["Kscr"] = 119974,
4334 ["Nscr"] = 119977,
4335 ["Oscr"] = 119978,
4336 ["Pscr"] = 119979,
4337 ["Qscr"] = 119980,
4338 ["Sscr"] = 119982,
4339 ["Tscr"] = 119983,
4340 ["Uscr"] = 119984,
4341 ["Vscr"] = 119985,
4342 ["Wscr"] = 119986,
4343 ["Xscr"] = 119987,
4344 ["Yscr"] = 119988,
4345 ["Zscr"] = 119989,
4346 ["ascr"] = 119990,
4347 ["bscr"] = 119991,
4348 ["cscr"] = 119992,
4349 ["dscr"] = 119993,
4350 ["fscr"] = 119995,
4351 ["hscr"] = 119997,
4352 ["iscr"] = 119998,
4353 ["jscr"] = 119999,
4354 ["kscr"] = 120000,
4355 ["lscr"] = 120001,
4356 ["mscr"] = 120002,
4357 ["nscr"] = 120003,
4358 ["pscr"] = 120005,
4359 ["qscr"] = 120006,
4360 ["rscr"] = 120007,
4361 ["sscr"] = 120008,
4362 ["tscr"] = 120009,
4363 ["uscr"] = 120010,
4364 ["vscr"] = 120011,

```



```

4365 ["wscr"] = 120012,
4366 ["xscr"] = 120013,
4367 ["yscr"] = 120014,
4368 ["zscr"] = 120015,
4369 ["Afr"] = 120068,
4370 ["Bfr"] = 120069,
4371 ["Dfr"] = 120071,
4372 ["Efr"] = 120072,
4373 ["Ffr"] = 120073,
4374 ["Gfr"] = 120074,
4375 ["Jfr"] = 120077,
4376 ["Kfr"] = 120078,
4377 ["Lfr"] = 120079,
4378 ["Mfr"] = 120080,
4379 ["Nfr"] = 120081,
4380 ["Ofr"] = 120082,
4381 ["Pfr"] = 120083,
4382 ["Qfr"] = 120084,
4383 ["Sfr"] = 120086,
4384 ["Tfr"] = 120087,
4385 ["Ufr"] = 120088,
4386 ["Vfr"] = 120089,
4387 ["Wfr"] = 120090,
4388 ["Xfr"] = 120091,
4389 ["Yfr"] = 120092,
4390 ["afr"] = 120094,
4391 ["bfr"] = 120095,
4392 ["cfr"] = 120096,
4393 ["dfr"] = 120097,
4394 ["efr"] = 120098,
4395 ["ffr"] = 120099,
4396 ["gfr"] = 120100,
4397 ["hfr"] = 120101,
4398 ["ifr"] = 120102,
4399 ["jfr"] = 120103,
4400 ["kfr"] = 120104,
4401 ["lfr"] = 120105,
4402 ["mfr"] = 120106,
4403 ["nfr"] = 120107,
4404 ["ofr"] = 120108,
4405 ["pfr"] = 120109,
4406 ["qfr"] = 120110,
4407 ["rfr"] = 120111,
4408 ["sfr"] = 120112,
4409 ["tfr"] = 120113,
4410 ["ufr"] = 120114,
4411 ["vfr"] = 120115,

```

```

4412 ["wfr"] = 120116,
4413 ["xfr"] = 120117,
4414 ["yfr"] = 120118,
4415 ["zfr"] = 120119,
4416 ["Aopf"] = 120120,
4417 ["Bopf"] = 120121,
4418 ["Dopf"] = 120123,
4419 ["Eopf"] = 120124,
4420 ["Fopf"] = 120125,
4421 ["Gopf"] = 120126,
4422 ["Iopf"] = 120128,
4423 ["Jopf"] = 120129,
4424 ["Kopf"] = 120130,
4425 ["Lopf"] = 120131,
4426 ["Mopf"] = 120132,
4427 ["Oopf"] = 120134,
4428 ["Sopf"] = 120138,
4429 ["Topf"] = 120139,
4430 ["Uopf"] = 120140,
4431 ["Vopf"] = 120141,
4432 ["Wopf"] = 120142,
4433 ["Xopf"] = 120143,
4434 ["Yopf"] = 120144,
4435 ["aopf"] = 120146,
4436 ["bopf"] = 120147,
4437 ["copf"] = 120148,
4438 ["dopf"] = 120149,
4439 ["eopf"] = 120150,
4440 ["fopf"] = 120151,
4441 ["gopf"] = 120152,
4442 ["hopf"] = 120153,
4443 ["iopf"] = 120154,
4444 ["jopf"] = 120155,
4445 ["kopf"] = 120156,
4446 ["lopf"] = 120157,
4447 ["mopf"] = 120158,
4448 ["nopf"] = 120159,
4449 ["oopf"] = 120160,
4450 ["popf"] = 120161,
4451 ["qopf"] = 120162,
4452 ["ropf"] = 120163,
4453 ["sopf"] = 120164,
4454 ["topf"] = 120165,
4455 ["uopf"] = 120166,
4456 ["vopf"] = 120167,
4457 ["wopf"] = 120168,
4458 ["xopf"] = 120169,

```

```

4459 ["yopf"] = 120170,
4460 ["zopf"] = 120171,
4461 }

```

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

```

4462 function entities.dec_entity(s)
4463     return unicode.utf8.char(tonumber(s))
4464 end

```

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

```

4465 function entities.hex_entity(s)
4466     return unicode.utf8.char(tonumber("0x"..s))
4467 end

```

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

```

4468 function entities.char_entity(s)
4469     local n = character_entities[s]
4470     if n == nil then
4471         return "&" .. s .. ";"
4472     end
4473     return unicode.utf8.char(n)
4474 end

```

3.1.3 Plain T_EX Writer

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

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

```

4475 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.

```

4476 function M.writer.new(options)
4477   local self = {}

   Make options available as writer->options, so that it is accessible from exten-
   sions.

4478   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.
4479   local slice_specifiers = {}
4480   for specifier in options.slice:gmatch("[^%s]+") do
4481     table.insert(slice_specifiers, specifier)
4482   end
4483
4484   if #slice_specifiers == 2 then
4485     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4486     local slice_begin_type = self.slice_begin:sub(1, 1)
4487     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4488       self.slice_begin = "^" .. self.slice_begin
4489     end
4490     local slice_end_type = self.slice_end:sub(1, 1)
4491     if slice_end_type ~= "^" and slice_end_type ~= "$" then
4492       self.slice_end = "$" .. self.slice_end
4493     end
4494   elseif #slice_specifiers == 1 then
4495     self.slice_begin = "^" .. slice_specifiers[1]
4496     self.slice_end = "$" .. slice_specifiers[1]
4497   end
4498
4499   if self.slice_begin == "^" and self.slice_end ~= "^" then
4500     self.is_writing = true
4501   else
4502     self.is_writing = false
4503   end

   Define writer->suffix as the suffix of the produced cache files.
4504   self.suffix = ".tex"

   Define writer->space as the output format of a space character.
4505   self.space = " "

   Define writer->nbspace as the output format of a non-breaking space character.
4506   self.nbspace = "\\markdownRendererNbspace{"

   Define writer->plain as a function that will transform an input plain text block
   s to the output format.
4507   function self.plain(s)
4508     return s
4509   end

```

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

```

4510 function self.paragraph(s)
4511     if not self.is_writing then return "" end
4512     return s
4513 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.

```

4514 function self.pack(name)
4515     return [[\input ]] .. name .. [[\relax]]
4516 end

```

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

```

4517 function self.interblocksep()
4518     if not self.is_writing then return "" end
4519     return "\\markdownRendererInterblockSeparator\n{"
4520 end

```

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

```

4521 self.linebreak = "\\markdownRendererLineBreak\n{"

```

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

```

4522 self.ellipsis = "\\markdownRendererEllipsis{"

```

Define `writer->hrule` as the output format of a horizontal rule.

```

4523 function self.hrule()
4524     if not self.is_writing then return "" end
4525     return "\\markdownRendererHorizontalRule{"
4526 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.

```

4527 self.escaped_uri_chars = {
4528     [{""] = "\\markdownRendererLeftBrace{"},
4529     ["}"] = "\\markdownRendererRightBrace{"},
4530     [%"] = "\\markdownRendererPercentSign{"},
4531     [\\"] = "\\markdownRendererBackslash{"},
4532 }
4533 self.escaped_minimal_strings = {
4534     ["^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
4535     ["☒"] = "\\markdownRendererTickedBox{"},
4536     ["◻"] = "\\markdownRendererHalfTickedBox{"},
4537     ["□"] = "\\markdownRendererUntickedBox{"},
4538 }

```

Define a table `writer->escaped_chars` containing the mapping from special plain \TeX characters (including the active pipe character (`|`) of \ConTeXt) that need to be escaped for typeset content.

```

4539 self.escaped_chars = {
4540     ["{"] = "\\markdownRendererLeftBrace{}",
4541     ["}"] = "\\markdownRendererRightBrace{}",
4542     [%] = "\\markdownRendererPercentSign{}",
4543     [\\] = "\\markdownRendererBackslash{}",
4544     [#] = "\\markdownRendererHash{}",
4545     [$] = "\\markdownRendererDollarSign{}",
4546     [&] = "\\markdownRendererAmpersand{}",
4547     [_] = "\\markdownRendererUnderscore{}",
4548     [^] = "\\markdownRendererCircumflex{}",
4549     [~] = "\\markdownRendererTilde{}",
4550     [|] = "\\markdownRendererPipe{}",
4551 }

```

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

```

4552 self.escape = util.escaper(self.escaped_chars, self.escaped_minimal_strings)
4553 self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
4554 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.

```

4555 if options.hybrid then
4556     self.string = self.escape_minimal
4557     self.uri = self.escape_minimal
4558 else
4559     self.string = self.escape
4560     self.uri = self.escape_uri
4561 end

```

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

```

4562 function self.code(s)
4563     return {"\\markdownRendererCodeSpan{",self.escape(s),"}"}
4564 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.

```

4565 function self.link(lab,src,tit)
4566     return {"\\markdownRendererLink{",lab,"}",
4567             "{",self.escape(src),"}",
4568             "{",self.uri(src),"}",
4569             "{",self.string(tit or ""),"}"}

```

4570 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.

```
4571 function self.image(lab,src,tit)
4572     return {"\\markdownRendererImage{" ,lab,""},
4573             "{" ,self.string(src),""},
4574             "{" ,self.uri(src),""},
4575             "{" ,self.string(tit or ""),""}
4576 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.

```
4577 function self.bulletlist(items,tight)
4578     if not self.is_writing then return "" end
4579     local buffer = {}
4580     for _,item in ipairs(items) do
4581         buffer[#buffer + 1] = self.bulletitem(item)
4582     end
4583     local contents = util.intersperse(buffer,"\n")
4584     if tight and options.tightLists then
4585         return {"\\markdownRendererULBeginTight\n",contents,
4586                 "\n\\markdownRendererULEndTight "}
4587     else
4588         return {"\\markdownRendererULBegin\n",contents,
4589                 "\n\\markdownRendererULEnd "}
4590     end
4591 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.

```
4592 function self.bulletitem(s)
4593     return {"\\markdownRendererULItem ",s,
4594             "\\markdownRendererULItemEnd "}
4595 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.

```
4596 function self.orderedlist(items,tight,startnum)
4597     if not self.is_writing then return "" end
4598     local buffer = {}
4599     local num = startnum
4600     for _,item in ipairs(items) do
4601         buffer[#buffer + 1] = self.ordereditem(item,num)
```

```

4602         if num ~= nil then
4603             num = num + 1
4604         end
4605     end
4606     local contents = util.intersperse(buffer,"\n")
4607     if tight and options.tightLists then
4608         return {"\\markdownRendererOlBeginTight\n",contents,
4609             "\n\\markdownRendererOlEndTight "}
4610     else
4611         return {"\\markdownRendererOlBegin\n",contents,
4612             "\n\\markdownRendererOlEnd "}
4613     end
4614 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.

```

4615 function self.ordereditem(s,num)
4616     if num ~= nil then
4617         return {"\\markdownRendererOlItemWithNumber{" ,num,"} ",s,
4618             "\\markdownRendererOlItemEnd "}
4619     else
4620         return {"\\markdownRendererOlItem ",s,
4621             "\\markdownRendererOlItemEnd "}
4622     end
4623 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.

```

4624 function self.inline_html_comment(contents)
4625     return {"\\markdownRendererInlineHtmlComment{" ,contents,"}"}
4626 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.

```

4627 function self.block_html_comment(contents)
4628     if not self.is_writing then return "" end
4629     return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
4630         "\n\\markdownRendererBlockHtmlCommentEnd "}
4631 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.

```

4632 function self.inline_html_tag(contents)
4633     return {"\\markdownRendererInlineHtmlTag{" ,self.string(contents),"}"}

```


4634 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.

```
4635 function self.block_html_element(s)
4636   if not self.is_writing then return "" end
4637   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4638   return {"\\markdownRendererInputBlockHtmlElement{" ,name,"}"}
4639 end
```

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

```
4640 function self.emphasis(s)
4641   return {"\\markdownRendererEmphasis{" ,s,"}"}
4642 end
```

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

```
4643 function self.checkbox(f)
4644   if f == 1.0 then
4645     return "☒ "
4646   elseif f == 0.0 then
4647     return "☐ "
4648   else
4649     return "◻ "
4650   end
4651 end
```

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

```
4652 function self.strong(s)
4653   return {"\\markdownRendererStrongEmphasis{" ,s,"}"}
4654 end
```

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

```
4655 function self.blockquote(s)
4656   if #util.ropetostring(s) == 0 then return "" end
4657   return {"\\markdownRendererBlockQuoteBegin\n",s,
4658     "\n\\markdownRendererBlockQuoteEnd "}
4659 end
```

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

```
4660 function self.verbatim(s)
4661   if not self.is_writing then return "" end
4662   s = string.gsub(s, '[\r\n%s]*$', '')
4663   local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
```

```

4664     return {"\\markdownRendererInputVerbatim{" ,name,""}
4665 end

```

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

```

4666 function self.document(d)
4667     local active_attributes = self.active_attributes
4668     local buf = {"\\markdownRendererDocumentBegin\n", d}
4669
4670     -- pop attributes for sections that have ended
4671     if options.headerAttributes and self.is_writing then
4672         while #active_attributes > 0 do
4673             local attributes = active_attributes[#active_attributes]
4674             if #attributes > 0 then
4675                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4676             end
4677             table.remove(active_attributes, #active_attributes)
4678         end
4679     end
4680
4681     table.insert(buf, "\\markdownRendererDocumentEnd")
4682
4683     return buf
4684 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.

```

4685 self.active_attributes = {}

```

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

```

4686 function self.heading(s, level, attributes)
4687     attributes = attributes or {}
4688     for i = 1, #attributes do
4689         attributes[attributes[i]] = true
4690     end
4691
4692     local active_attributes = self.active_attributes
4693     local slice_begin_type = self.slice_begin:sub(1, 1)
4694     local slice_begin_identifier = self.slice_begin:sub(2) or ""
4695     local slice_end_type = self.slice_end:sub(1, 1)
4696     local slice_end_identifier = self.slice_end:sub(2) or ""
4697
4698     local buf = {}
4699
4700     -- push empty attributes for implied sections
4701     while #active_attributes < level-1 do
4702         table.insert(active_attributes, {})

```

```

4703     end
4704
4705     -- pop attributes for sections that have ended
4706     while #active_attributes >= level do
4707         local active_identifiers = active_attributes[#active_attributes]
4708         -- tear down all active attributes at slice end
4709         if active_identifiers["#" .. slice_end_identifier] ~= nil
4710             and slice_end_type == "$" then
4711             for header_level = #active_attributes, 1, -1 do
4712                 if options.headerAttributes and #active_attributes[header_level] > 0 then
4713                     table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4714                 end
4715             end
4716             self.is_writing = false
4717         end
4718         table.remove(active_attributes, #active_attributes)
4719         if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
4720             table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4721         end
4722         -- apply all active attributes at slice beginning
4723         if active_identifiers["#" .. slice_begin_identifier] ~= nil
4724             and slice_begin_type == "$" then
4725             for header_level = 1, #active_attributes do
4726                 if options.headerAttributes and #active_attributes[header_level] > 0 then
4727                     table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4728                 end
4729             end
4730             self.is_writing = true
4731         end
4732     end
4733
4734     -- tear down all active attributes at slice end
4735     if attributes["#" .. slice_end_identifier] ~= nil
4736         and slice_end_type == "^" then
4737         for header_level = #active_attributes, 1, -1 do
4738             if options.headerAttributes and #active_attributes[header_level] > 0 then
4739                 table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
4740             end
4741         end
4742         self.is_writing = false
4743     end
4744
4745     -- push attributes for the new section
4746     table.insert(active_attributes, attributes)
4747     if self.is_writing and options.headerAttributes and #attributes > 0 then
4748         table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4749     end

```

```

4750
4751 -- apply all active attributes at slice beginning
4752 if attributes["#" .. slice_begin_identifier] ~= nil
4753     and slice_begin_type == "^" then
4754     for header_level = 1, #active_attributes do
4755         if options.headerAttributes and #active_attributes[header_level] > 0 then
4756             table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
4757         end
4758     end
4759     self.is_writing = true
4760 end
4761
4762 if self.is_writing then
4763     table.sort(attributes)
4764     local key, value
4765     for i = 1, #attributes do
4766         if attributes[i]:sub(1, 1) == "#" then
4767             table.insert(buf, {"\\markdownRendererAttributeIdentifier{" ,
4768                 attributes[i]:sub(2), "}"})
4769         elseif attributes[i]:sub(1, 1) == "." then
4770             table.insert(buf, {"\\markdownRendererAttributeClassName{" ,
4771                 attributes[i]:sub(2), "}"})
4772         else
4773             key, value = attributes[i]:match("([~= ]+)%s*=%s*(.*)")
4774             table.insert(buf, {"\\markdownRendererAttributeKeyValue{" ,
4775                 key, "{", value, "}"})
4776         end
4777     end
4778 end
4779
4780 local cmd
4781 level = level + options.shiftHeadings
4782 if level <= 1 then
4783     cmd = "\\markdownRendererHeadingOne"
4784 elseif level == 2 then
4785     cmd = "\\markdownRendererHeadingTwo"
4786 elseif level == 3 then
4787     cmd = "\\markdownRendererHeadingThree"
4788 elseif level == 4 then
4789     cmd = "\\markdownRendererHeadingFour"
4790 elseif level == 5 then
4791     cmd = "\\markdownRendererHeadingFive"
4792 elseif level >= 6 then
4793     cmd = "\\markdownRendererHeadingSix"
4794 else
4795     cmd = ""
4796 end

```

```

4797     if self.is_writing then
4798         table.insert(buf, {cmd, "{", s, "}"})
4799     end
4800
4801     return buf
4802 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.

```

4803 function self.get_state()
4804     return {
4805         is_writing=self.is_writing,
4806         active_attributes={table.unpack(self.active_attributes)},
4807     }
4808 end

```

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

```

4809 function self.set_state(s)
4810     local previous_state = self.get_state()
4811     for key, value in pairs(s) do
4812         self[key] = value
4813     end
4814     return previous_state
4815 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`.

```

4816 function self.defer_call(f)
4817     local previous_state = self.get_state()
4818     return function(...)
4819         local state = self.set_state(previous_state)
4820         local return_value = f(...)
4821         self.set_state(state)
4822         return return_value
4823     end
4824 end
4825
4826 return self
4827 end

```

3.1.4 Parsers

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

```

4828 local parsers = {}

```

3.1.4.1 Basic Parsers

```
4829 parsers.percent          = P("%")
4830 parsers.at                 = P("@")
4831 parsers.comma              = P(",")
4832 parsers.asterisk           = P("*")
4833 parsers.dash                = P("-")
4834 parsers.plus                = P("+")
4835 parsers.underscore         = P("_")
4836 parsers.period              = P(".")
4837 parsers.hash                = P("#")
4838 parsers.ampersand           = P("&")
4839 parsers.backtick            = P("`")
4840 parsers.less                = P("<")
4841 parsers.more                = P(">")
4842 parsers.space               = P(" ")
4843 parsers.squote              = P("'")
4844 parsers.dquote              = P('"')
4845 parsers.lparent             = P("(")
4846 parsers.rparent             = P(")")
4847 parsers.lbracket            = P("[")
4848 parsers.rbracket            = P("]")
4849 parsers.lbrace              = P("{")
4850 parsers.rbrace              = P("}")
4851 parsers.circumflex          = P("^")
4852 parsers.slash               = P("/")
4853 parsers.equal               = P("=")
4854 parsers.colon               = P(":")
4855 parsers.semicolon           = P(";")
4856 parsers.exclamation         = P("!")
4857 parsers.pipe                = P("|")
4858 parsers.tilde               = P("~")
4859 parsers.backslash           = P("\\")
4860 parsers.tab                  = P("\t")
4861 parsers.newline              = P("\n")
4862 parsers.tightblocksep       = P("\001")
4863
4864 parsers.digit                = R("09")
4865 parsers.hexdigit            = R("09", "af", "AF")
4866 parsers.letter              = R("AZ", "az")
4867 parsers.alphanumeric        = R("AZ", "az", "09")
4868 parsers.keyword             = parsers.letter
4869                             * parsers.alphanumeric^0
4870 parsers.internal_punctuation = S(":,.;?")
4871
4872 parsers.doubleasterisks      = P("**")
4873 parsers.doubleunderscores    = P("__")
4874 parsers.doubletildes         = P("~~")
```

```

4875 parsers.fourspace      = P("    ")
4876
4877 parsers.any              = P(1)
4878 parsers.fail             = parsers.any - 1
4879
4880 parsers.escapable        = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
4881 parsers.anyescaped       = parsers.backslash / "\"" * parsers.escapable
4882 + parsers.any
4883
4884 parsers.spacechar        = S("\t ")
4885 parsers.spacing          = S(" \n\r\t")
4886 parsers.nonspacechar     = parsers.any - parsers.spacing
4887 parsers.optionalspace    = parsers.spacechar^0
4888
4889 parsers.normalchar       = parsers.any - (V("SpecialChar")
4890                                           + parsers.spacing
4891                                           + parsers.tightblocksep)
4892
4893 parsers.eof              = -parsers.any
4894 parsers.nonindentSPACE   = parsers.space^-3 * - parsers.spacechar
4895 parsers.indent           = parsers.space^-3 * parsers.tab
4896 parsers.linechar         = parsers.fourspace / "\""
4897
4898 parsers.blankline        = parsers.optionalspace
4899 * parsers.newline / "\n"
4900 parsers.blanklines       = parsers.blankline^0
4901 parsers.skipblanklines    = (parsers.optionalspace * parsers.newline)^0
4902 parsers.indentedline     = parsers.indent / "\""
4903 * C(parsers.linechar^1 * parsers.newline^-
4904    1)
4905 parsers.optionallyindentedline = parsers.indent^-1 / "\""
4906 * C(parsers.linechar^1 * parsers.newline^-
4907    1)
4908 parsers.sp               = parsers.spacing^0
4909 parsers.spnl             = parsers.optionalspace
4910 * (parsers.newline * parsers.optionalspace)^-
4911    1
4912 parsers.line             = parsers.linechar^0 * parsers.newline
4913 parsers.nonemptyline     = parsers.line - parsers.blankline
4914
4915 The parsers.commented_line^1 parser recognizes the regular language of TEX
4916 comments, see an equivalent finite automaton in Figure 6.
4917
4918 parsers.commented_line_letter = parsers.linechar
4919 + parsers.newline
4920 - parsers.backslash
4921 - parsers.percent
4922 parsers.commented_line      = Cg(Cc(""), "backslashes")

```

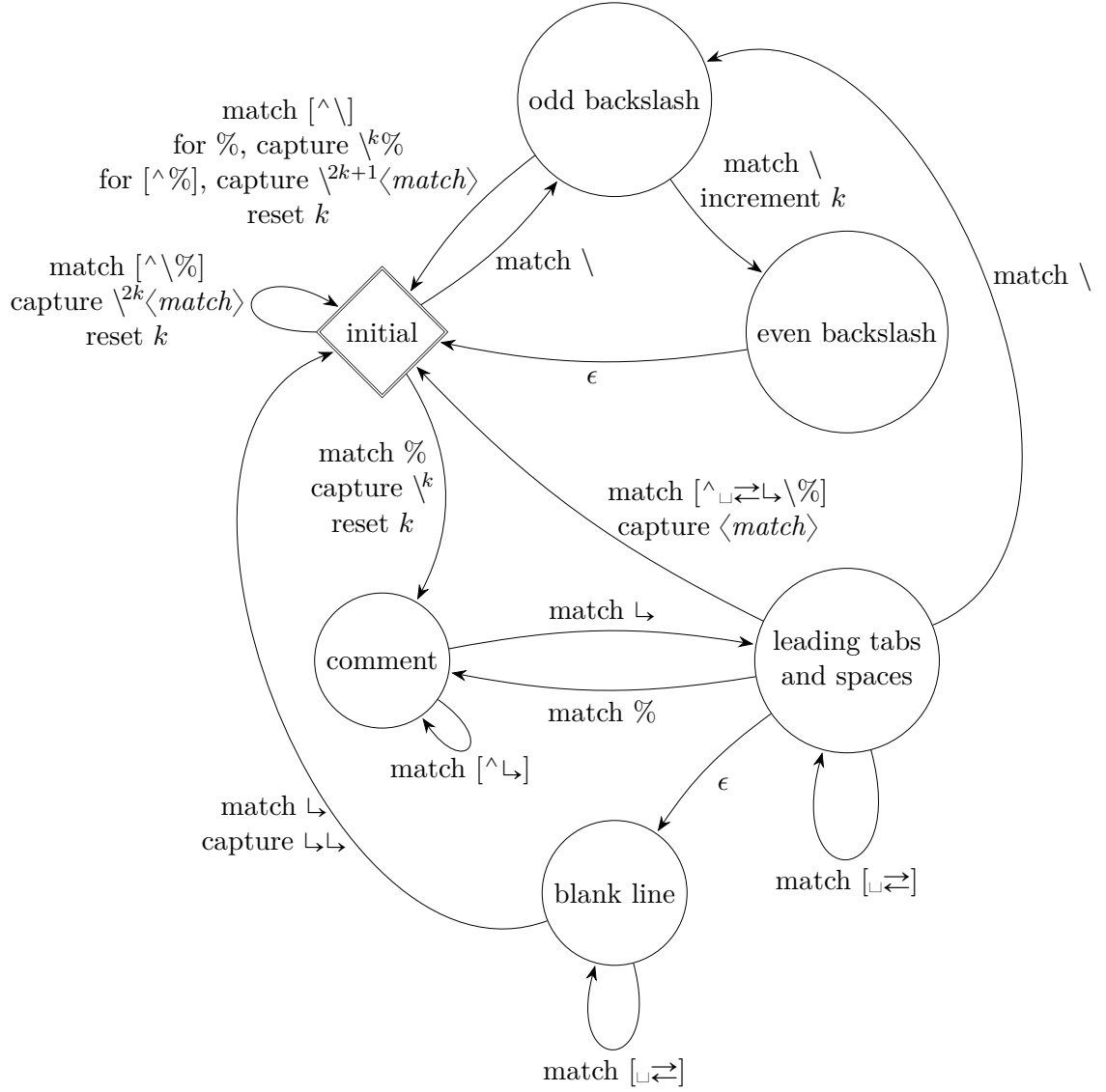


Figure 6: A pushdown automaton that recognizes \TeX comments


```

4916 * ((#(parsers.comment_line_letter
4917   - parsers.newline)
4918   * Cb("backslashes")
4919   * Cs(parsers.comment_line_letter
4920     - parsers.newline)^1 -- initial
4921   * Cg(Cc(""), "backslashes"))
4922 + #(parsers.backslash * parsers.backslash)
4923 * Cg((parsers.backslash -- even backslash
4924   * parsers.backslash)^1, "backslashes")
4925 + (parsers.backslash
4926   * (#parsers.percent
4927     * Cb("backslashes")
4928     / function(backslashes)
4929       return string.rep("\\", #backslashes / 2)
4930     end
4931     * C(parsers.percent)
4932     + #parsers.comment_line_letter
4933     * Cb("backslashes")
4934     * Cc("\\")
4935     * C(parsers.comment_line_letter))
4936   * Cg(Cc(""), "backslashes"))^0
4937 * (#parsers.percent
4938   * Cb("backslashes")
4939   / function(backslashes)
4940     return string.rep("\\", #backslashes / 2)
4941   end
4942 * ((parsers.percent -- comment
4943   * parsers.line
4944   * #parsers.blankline) -- blank line
4945   / "\n"
4946   + parsers.percent -- comment
4947   * parsers.line
4948   * parsers.optionalspace) -- leading tabs and spaces
4949 + #(parsers.newline)
4950 * Cb("backslashes")
4951 * C(parsers.newline))
4952
4953 parsers.chunk = parsers.line * (parsers.optionallyindentedline
4954   - parsers.blankline)^0
4955
4956 parsers.attribute_key_char = parsers.alphanumeric + S("_-")
4957 parsers.attribute_key = (parsers.attribute_key_char
4958   - parsers.dash - parsers.digit)
4959 * parsers.attribute_key_char^0
4960 parsers.attribute_value = ( (parsers.dquote / "\"")
4961   * (parsers.anyescaped - parsers.dquote)^0
4962   * (parsers.dquote / "\""))

```

```

4963                                     + ( parsers.anyescaped - parsers.dquote - parsers.rbrace
4964                                     - parsers.space)^0
4965
4966 -- block followed by 0 or more optionally
4967 -- indented blocks with first line indented.
4968 parsers.indented_blocks = function(bl)
4969   return Cs( bl
4970             * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
4971             * (parsers.blankline^1 + parsers.eof) )
4972 end

```

3.1.4.2 Parsers Used for Markdown Lists

```

4973 parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)
4974
4975 parsers.bullet = ( parsers.bulletchar * #parsers.spacing
4976                  * (parsers.tab + parsers.space^-3)
4977                  + parsers.space * parsers.bulletchar * #parsers.spacing
4978                  * (parsers.tab + parsers.space^-2)
4979                  + parsers.space * parsers.space * parsers.bulletchar
4980                  * #parsers.spacing
4981                  * (parsers.tab + parsers.space^-1)
4982                  + parsers.space * parsers.space * parsers.space
4983                  * parsers.bulletchar * #parsers.spacing
4984                  )
4985
4986 local function tickbox(interior)
4987   return parsers.optionalspace * parsers.lbracket
4988   * interior * parsers.rbracket * parsers.spacechar^1
4989 end
4990
4991 parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
4992 parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
4993 parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)
4994

```

3.1.4.3 Parsers Used for Markdown Code Spans

```

4995 parsers.openticks = Cg(parsers.backtick^1, "ticks")
4996
4997 local function captures_equal_length(_,i,a,b)
4998   return #a == #b and i
4999 end
5000
5001 parsers.closeticks = parsers.space^-1
5002                  * Cmt(C(parsers.backtick^1)
5003                  * Cb("ticks"), captures_equal_length)
5004

```

```

5005 parsers.intickschar = (parsers.any - S(" \n\r`"))
5006                       + (parsers.newline * -parsers.blankline)
5007                       + (parsers.space - parsers.closeticks)
5008                       + (parsers.backtick^1 - parsers.closeticks)
5009
5010 parsers.inticks       = parsers.openticks * parsers.space^-1
5011                       * C(parsers.intickschar^0) * parsers.closeticks

```

3.1.4.4 Parsers Used for Fenced Code Blocks

```

5012 local function captures_geq_length(_,i,a,b)
5013   return #a >= #b and i
5014 end
5015
5016 parsers.infostring     = (parsers.linechar - (parsers.backtick
5017   + parsers.space^1 * (parsers.newline + parsers.eof)))^0
5018
5019 local fenceindent
5020 parsers.fencehead      = function(char)
5021   return               C(parsers.nonindentospace) / function(s) fenceindent = #s end
5022   * Cg(char^3, "fencelength")
5023   * parsers.optionalspace * C(parsers.infostring)
5024   * parsers.optionalspace * (parsers.newline + parsers.eof)
5025 end
5026
5027 parsers.fencetail      = function(char)
5028   return               parsers.nonindentospace
5029   * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
5030   * parsers.optionalspace * (parsers.newline + parsers.eof)
5031   + parsers.eof
5032 end
5033
5034 parsers.fencedline     = function(char)
5035   return               C(parsers.line - parsers.fencetail(char))
5036   / function(s)
5037     local i = 1
5038     local remaining = fenceindent
5039     while true do
5040       local c = s:sub(i, i)
5041       if c == " " and remaining > 0 then
5042         remaining = remaining - 1
5043         i = i + 1
5044       elseif c == "\t" and remaining > 3 then
5045         remaining = remaining - 4
5046         i = i + 1
5047       else
5048         break

```

```

5049             end
5050         end
5051         return s:sub(i)
5052     end
5053 end

```

3.1.4.5 Parsers Used for Markdown Tags and Links

```

5054 parsers.leader      = parsers.space~-3
5055
5056 -- content in balanced brackets, parentheses, or quotes:
5057 parsers.bracketed    = P{ parsers.lbracket
5058     * (( parsers.backslash / "\"" * parsers.rbracket
5059         + parsers.any - (parsers.lbracket
5060                         + parsers.rbracket
5061                         + parsers.blankline^2)
5062         ) + V(1))^0
5063     * parsers.rbracket }
5064
5065 parsers.inparens     = P{ parsers.lparent
5066     * ((parsers.anyescaped - (parsers.lparent
5067                             + parsers.rparent
5068                             + parsers.blankline^2)
5069         ) + V(1))^0
5070     * parsers.rparent }
5071
5072 parsers.squoted      = P{ parsers.squote * parsers.alphanumeric
5073     * ((parsers.anyescaped - (parsers.squote
5074                             + parsers.blankline^2)
5075         ) + V(1))^0
5076     * parsers.squote }
5077
5078 parsers.dquoted      = P{ parsers.dquote * parsers.alphanumeric
5079     * ((parsers.anyescaped - (parsers.dquote
5080                             + parsers.blankline^2)
5081         ) + V(1))^0
5082     * parsers.dquote }
5083
5084 -- bracketed tag for markdown links, allowing nested brackets:
5085 parsers.tag          = parsers.lbracket
5086     * Cs((parsers.alphanumeric^1
5087         + parsers.bracketed
5088         + parsers.inticks
5089         + ( parsers.backslash / "\"" * parsers.rbracket
5090           + parsers.any
5091           - (parsers.rbracket + parsers.blankline^2))))^0)
5092     * parsers.rbracket

```

```

5093
5094 -- url for markdown links, allowing nested brackets:
5095 parsers.url      = parsers.less * Cs((parsers.anyescaped
5096                               - parsers.more)^0)
5097                               * parsers.more
5098                               + Cs((parsers.inparens + (parsers.anyescaped
5099                               - parsers.spacing
5100                               - parsers.rparent))^1)
5101
5102 -- quoted text, possibly with nested quotes:
5103 parsers.title_s   = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
5104                               + parsers.squoted)^0)
5105                               * parsers.squote
5106
5107 parsers.title_d   = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5108                               + parsers.dquoted)^0)
5109                               * parsers.dquote
5110
5111 parsers.title_p   = parsers.lparent
5112                               * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5113                               * parsers.rparent
5114
5115 parsers.title     = parsers.title_d + parsers.title_s + parsers.title_p
5116
5117 parsers.optionaltitle
5118                               = parsers.spnl * parsers.title * parsers.spacechar^0
5119                               + Cc("")

```

3.1.4.6 Parsers Used for HTML

```

5120 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5121 parsers.keyword_exact = function(s)
5122   local parser = P(0)
5123   for i=1,#s do
5124     local c = s:sub(i,i)
5125     local m = c .. upper(c)
5126     parser = parser * S(m)
5127   end
5128   return parser
5129 end
5130
5131 parsers.block_keyword =
5132   parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5133   parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5134   parsers.keyword_exact("div") + parsers.keyword_exact("div") +
5135   parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5136   parsers.keyword_exact("li") + parsers.keyword_exact("ol") +

```

```

5137 parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5138 parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5139 parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5140 parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5141 parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5142 parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5143 parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5144 parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5145 parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5146 parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5147 parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5148 parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5149 parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5150
5151 -- There is no reason to support bad html, so we expect quoted attributes
5152 parsers.htmlattributevalue
5153         = parsers.squote * (parsers.any - (parsers.blankline
5154                                           + parsers.squote))^0
5155           * parsers.squote
5156         + parsers.dquote * (parsers.any - (parsers.blankline
5157                                           + parsers.dquote))^0
5158           * parsers.dquote
5159
5160 parsers.htmlattribute = parsers.spacing^1
5161                       * (parsers.alphanumeric + S("_-"))^1
5162                       * parsers.sp * parsers.equal * parsers.sp
5163                       * parsers.htmlattributevalue
5164
5165 parsers.htmlcomment = P("<!--")
5166                     * parsers.optionalspace
5167                     * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5168                     * parsers.optionalspace
5169                     * P("-->")
5170
5171 parsers.htmlinstruction = P("<?") * (parsers.any - P(">"))^0 * P(">")
5172
5173 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5174                   * parsers.sp * parsers.more
5175
5176 parsers.openelt_exact = function(s)
5177   return parsers.less * parsers.sp * parsers.keyword_exact(s)
5178         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5179 end
5180
5181 parsers.openelt_block = parsers.sp * parsers.block_keyword
5182                   * parsers.htmlattribute^0 * parsers.sp * parsers.more
5183

```

```

5184 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5185         * parsers.keyword * parsers.sp * parsers.more
5186
5187 parsers.closeelt_exact = function(s)
5188     return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5189         * parsers.sp * parsers.more
5190 end
5191
5192 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5193         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5194         * parsers.more
5195
5196 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
5197         * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5198         * parsers.more
5199
5200 parsers.displaytext = (parsers.any - parsers.less)^1
5201
5202 -- return content between two matched HTML tags
5203 parsers.in_matched = function(s)
5204     return { parsers.openelt_exact(s)
5205         * (V(1) + parsers.displaytext
5206             + (parsers.less - parsers.closeelt_exact(s)))^0
5207         * parsers.closeelt_exact(s) }
5208 end
5209
5210 local function parse_matched_tags(s,pos)
5211     local t = string.lower(peg.match(C(parsers.keyword),s,pos))
5212     return peg.match(parsers.in_matched(t),s,pos-1)
5213 end
5214
5215 parsers.in_matched_block_tags = parsers.less
5216         * Cmt(#parsers.openelt_block, parse_matched_tags)
5217

```

3.1.4.7 Parsers Used for HTML Entities

```

5218 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5219         * C(parsers.hexdigit^1) * parsers.semicolon
5220 parsers.decentity = parsers.ampersand * parsers.hash
5221         * C(parsers.digit^1) * parsers.semicolon
5222 parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
5223         * parsers.semicolon

```

3.1.4.8 Helpers for References

```

5224 -- parse a reference definition: [foo]: /bar "title"
5225 parsers.define_reference_parser = parsers.leader * parsers.tag * parsers.colon

```

```

5226                                     * parsers.spacechar^0 * parsers.url
5227                                     * parsers.optionaltitle * parsers.blankline^1

```

3.1.4.9 Inline Elements

```

5228 parsers.Inline = V("Inline")
5229 parsers.IndentedInline = V("IndentedInline")
5230
5231 -- parse many p between starter and ender
5232 parsers.between = function(p, starter, ender)
5233   local ender2 = B(parsers.nonspacechar) * ender
5234   return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
5235 end
5236
5237 parsers.urlchar = parsers.anyescaped - parsers.newline - parsers.more

```

3.1.4.10 Block Elements

```

5238 parsers.lineof = function(c)
5239   return (parsers.leader * (P(c) * parsers.optionalspace)^3
5240         * (parsers.newline * parsers.blankline^1
5241         + parsers.newline^-1 * parsers.eof))
5242 end

```

3.1.4.11 Headings

```

5243 parsers.heading_attribute = (parsers.dash * Cc(".unnumbered"))
5244                             + C((parsers.hash + parsers.period)
5245                             * parsers.attribute_key)
5246                             + Cs( parsers.attribute_key
5247                             * parsers.optionalspace * parsers.equal * parsers.optional
5248                             * parsers.attribute_value)
5249 parsers.HeadingAttributes = parsers.lbrace
5250                             * parsers.optionalspace
5251                             * parsers.heading_attribute
5252                             * (parsers.spacechar^1
5253                             * parsers.heading_attribute)^0
5254                             * parsers.optionalspace
5255                             * parsers.rbrace
5256
5257 -- parse Atx heading start and return level
5258 parsers.HeadingStart = #parsers.hash * C(parsers.hash^-6)
5259                       * -parsers.hash / length
5260
5261 -- parse setext header ending and return level
5262 parsers.HeadingLevel = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)
5263
5264 local function strip_atx_end(s)

```



```

5265   return s:gsub("#%s]*\n$", "")
5266 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`.

```

5267 M.reader = {}
5268 function M.reader.new(writer, options)
5269   local self = {}

```

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

```

5270   self.writer = writer
5271   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.

```

5272   self.parsers = {}
5273   (function(parsers)
5274     setmetatable(self.parsers, {
5275       __index = function (_, key)
5276         return parsers[key]
5277       end
5278     })
5279   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.

```

5280   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.

```

5281   function self.normalize_tag(tag)
5282     return string.lower(

```

```

5283     gsub(util.ropetostring(tag), "[\n\r\t]+", " ")
5284 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.

```

5285 local function iterlines(s, f)
5286     local rope = lpeg.match(Ct((parsers.line / f)^1), s)
5287     return util.ropetostring(rope)
5288 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.

```

5289 if options.preserveTabs then
5290     self.expandtabs = function(s) return s end
5291 else
5292     self.expandtabs = function(s)
5293         if s:find("\t") then
5294             return iterlines(s, util.expand_tabs_in_line)
5295         else
5296             return s
5297         end
5298     end
5299 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.

```

5300 self.parser_functions = {}
5301 self.create_parser = function(name, grammar, topLevel)
5302     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.

```

5303         if topLevel and options.stripIndent then
5304             local min_prefix_length, min_prefix = nil, ''
5305             str = iterlines(str, function(line)
5306                 if lpeg.match(parsers.nonemptyline, line) == nil then
5307                     return line
5308                 end
5309                 line = util.expand_tabs_in_line(line)

```

```

5310         local prefix = lpeg.match(C(parsers.optionalspace), line)
5311         local prefix_length = #prefix
5312         local is_shorter = min_prefix_length == nil
5313         is_shorter = is_shorter or prefix_length < min_prefix_length
5314         if is_shorter then
5315             min_prefix_length, min_prefix = prefix_length, prefix
5316         end
5317         return line
5318     end)
5319     str = str:gsub('^' .. min_prefix, '')
5320 end

```

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

```

5321     if toplevel and (options.texComments or options.hybrid) then
5322         str = lpeg.match(Ct(parsers.commented_line^1), str)
5323         str = util.rope_to_string(str)
5324     end
5325     local res = lpeg.match(grammar(), str)
5326     if res == nil then
5327         error(format("%s failed on:\n%s", name, str:sub(1,20)))
5328     else
5329         return res
5330     end
5331 end
5332 end
5333
5334 self.create_parser("parse_blocks",
5335     function()
5336         return parsers.blocks
5337     end, true)
5338
5339 self.create_parser("parse_blocks_nested",
5340     function()
5341         return parsers.blocks_nested
5342     end, false)
5343
5344 self.create_parser("parse_inlines",
5345     function()
5346         return parsers.inlines
5347     end, false)
5348
5349 self.create_parser("parse_inlines_no_link",
5350     function()
5351         return parsers.inlines_no_link
5352     end, false)

```

```

5353
5354 self.create_parser("parse_inlines_no_inline_note",
5355                     function()
5356                         return parsers.inlines_no_inline_note
5357                     end, false)
5358
5359 self.create_parser("parse_inlines_no_html",
5360                     function()
5361                         return parsers.inlines_no_html
5362                     end, false)
5363
5364 self.create_parser("parse_inlines_nbsp",
5365                     function()
5366                         return parsers.inlines_nbsp
5367                     end, false)

```

3.1.5.3 Parsers Used for Markdown Lists (local)

```

5368 if options.hashEnumerators then
5369     parsers.dig = parsers.digit + parsers.hash
5370 else
5371     parsers.dig = parsers.digit
5372 end
5373
5374 parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
5375                     + C(parsers.dig^2 * parsers.period) * #parsers.spacing
5376                     * (parsers.tab + parsers.space^1)
5377                     + C(parsers.dig * parsers.period) * #parsers.spacing
5378                     * (parsers.tab + parsers.space^-2)
5379                     + parsers.space * C(parsers.dig^2 * parsers.period)
5380                     * #parsers.spacing
5381                     + parsers.space * C(parsers.dig * parsers.period)
5382                     * #parsers.spacing
5383                     * (parsers.tab + parsers.space^-1)
5384                     + parsers.space * parsers.space * C(parsers.dig^1
5385                     * parsers.period) * #parsers.spacing

```

3.1.5.4 Parsers Used for Blockquotes (local)

```

5386 -- strip off leading > and indents, and run through blocks
5387 parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
5388                             1)/""
5389                             * parsers.linechar^0 * parsers.newline)^1
5389                             * (-(parsers.leader * parsers.more
5390                             + parsers.blankline) * parsers.linechar^1
5391                             * parsers.newline)^0
5392
5393 if not options.breakableBlockquotes then

```

```

5394     parsers.blockquote_body = parsers.blockquote_body
5395                               * (parsers.blankline~0 / "")
5396 end

```

3.1.5.5 Parsers Used for Footnotes (local)

3.1.5.6 Helpers for Links and References (local)

```

5397 -- List of references defined in the document
5398 local references
5399
5400 -- add a reference to the list
5401 local function register_link(tag,url,title)
5402     references[self.normalize_tag(tag)] = { url = url, title = title }
5403     return ""
5404 end
5405
5406 -- lookup link reference and return either
5407 -- the link or nil and fallback text.
5408 local function lookup_reference(label,sps,tag)
5409     local tagpart
5410     if not tag then
5411         tag = label
5412         tagpart = ""
5413     elseif tag == "" then
5414         tag = label
5415         tagpart = "[]"
5416     else
5417         tagpart = {"[",
5418             self.parser_functions.parse_inlines(tag),
5419             "]" }
5420     end
5421     if sps then
5422         tagpart = {sps, tagpart}
5423     end
5424     local r = references[self.normalize_tag(tag)]
5425     if r then
5426         return r
5427     else
5428         return nil, {"[",
5429             self.parser_functions.parse_inlines(label),
5430             "]", tagpart}
5431     end
5432 end
5433
5434 -- lookup link reference and return a link, if the reference is found,
5435 -- or a bracketed label otherwise.

```

```

5436 local function indirect_link(label,sps,tag)
5437     return writer.defer_call(function()
5438         local r,fallback = lookup_reference(label,sps,tag)
5439         if r then
5440             return writer.link(
5441                 self.parser_functions.parse_inlines_no_link(label),
5442                 r.url, r.title)
5443         else
5444             return fallback
5445         end
5446     end)
5447 end
5448
5449 -- lookup image reference and return an image, if the reference is found,
5450 -- or a bracketed label otherwise.
5451 local function indirect_image(label,sps,tag)
5452     return writer.defer_call(function()
5453         local r,fallback = lookup_reference(label,sps,tag)
5454         if r then
5455             return writer.image(writer.string(label), r.url, r.title)
5456         else
5457             return {"!", fallback}
5458         end
5459     end)
5460 end

```

3.1.5.7 Inline Elements (local)

```

5461 parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
5462                  / writer.string
5463
5464 parsers.Symbol   = (V("SpecialChar") - parsers.tightblocksep)
5465                  / writer.string
5466
5467 parsers.Ellipsis = P("...") / writer.ellipsis
5468
5469 parsers.Smart    = parsers.Ellipsis
5470
5471 parsers.Code     = parsers.inticks / writer.code
5472
5473 if options.blankBeforeBlockquote then
5474     parsers.bqstart = parsers.fail
5475 else
5476     parsers.bqstart = parsers.more
5477 end
5478
5479 if options.blankBeforeHeading then

```

```

5480     parsers.headerstart = parsers.fail
5481 else
5482     parsers.headerstart = parsers.hash
5483                         + (parsers.line * (parsers.equal^1 + parsers.dash^1)
5484                         * parsers.optionalspace * parsers.newline)
5485 end
5486
5487 parsers.EndlineExceptions
5488     = parsers.blankline -- paragraph break
5489     + parsers.tightblocksep -- nested list
5490     + parsers.eof        -- end of document
5491     + parsers.bqstart
5492     + parsers.headerstart
5493
5494 parsers.Endline = parsers.newline
5495                 * -V("EndlineExceptions")
5496                 * parsers.spacechar^0
5497                 / (options.hardLineBreaks and writer.linebreak
5498                   or writer.space)
5499
5500 parsers.OptionalIndent
5501     = parsers.spacechar^1 / writer.space
5502
5503 parsers.Space = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5504               + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5505               + parsers.spacechar^1 * parsers.Endline
5506                               * parsers.optionalspace
5507                               / (options.hardLineBreaks
5508                                 and writer.linebreak
5509                                 or writer.space)
5510               + parsers.spacechar^1 * parsers.optionalspace
5511                               / writer.space
5512
5513 parsers.NonbreakingEndline
5514     = parsers.newline
5515     * -V("EndlineExceptions")
5516     * parsers.spacechar^0
5517     / (options.hardLineBreaks and writer.linebreak
5518       or writer.nbsp)
5519
5520 parsers.NonbreakingSpace
5521     = parsers.spacechar^2 * parsers.Endline / writer.linebreak
5522     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
5523     + parsers.spacechar^1 * parsers.Endline
5524                               * parsers.optionalspace
5525                               / (options.hardLineBreaks
5526                                 and writer.linebreak

```

```

5527                                     or writer.nbsp)
5528         + parsers.spacechar^1 * parsers.optionalspace
5529                                     / writer.nbsp
5530
5531 if options.underscores then
5532     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5533                                     parsers.doubleasterisks)
5534         + parsers.between(parsers.Inline, parsers.doubleunderscores,
5535                                     parsers.doubleunderscores)
5536     ) / writer.strong
5537
5538     parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5539                                     parsers.asterisk)
5540         + parsers.between(parsers.Inline, parsers.underscore,
5541                                     parsers.underscore)
5542     ) / writer.emphasis
5543 else
5544     parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
5545                                     parsers.doubleasterisks)
5546     ) / writer.strong
5547
5548     parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
5549                                     parsers.asterisk)
5550     ) / writer.emphasis
5551 end
5552
5553 parsers.AutoLinkUrl   = parsers.less
5554                       * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
5555                       * parsers.more
5556                       / function(url)
5557                           return writer.link(writer.escape(url), url)
5558                       end
5559
5560 parsers.AutoLinkEmail = parsers.less
5561                       * C((parsers.alphanumeric + S("-._+"))^1
5562                       * P("@") * parsers.urlchar^1)
5563                       * parsers.more
5564                       / function(email)
5565                           return writer.link(writer.escape(email),
5566                                           "mailto: "..email)
5567                       end
5568
5569 parsers.AutoLinkRelativeReference
5570     = parsers.less
5571     * C(parsers.urlchar^1)
5572     * parsers.more
5573     / function(url)

```



```

5574         return writer.link(writer.escape(url), url)
5575     end
5576
5577     parsers.DirectLink = (parsers.tag / self.parser_functions.parse_inlines_no_link)
5578     * parsers.spnl
5579     * parsers.lparent
5580     * (parsers.url + Cc("")) -- link can be empty [foo]()
5581     * parsers.optionaltitle
5582     * parsers.rparent
5583     / writer.link
5584
5585     parsers.IndirectLink = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
1
5586     / indirect_link
5587
5588     -- parse a link or image (direct or indirect)
5589     parsers.Link = parsers.DirectLink + parsers.IndirectLink
5590
5591     parsers.DirectImage = parsers.exclamation
5592     * (parsers.tag / self.parser_functions.parse_inlines)
5593     * parsers.spnl
5594     * parsers.lparent
5595     * (parsers.url + Cc("")) -- link can be empty [foo]()
5596     * parsers.optionaltitle
5597     * parsers.rparent
5598     / writer.image
5599
5600     parsers.IndirectImage = parsers.exclamation * parsers.tag
5601     * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5602
5603     parsers.Image = parsers.DirectImage + parsers.IndirectImage
5604
5605     -- avoid parsing long strings of * or _ as emph/strong
5606     parsers.UlOrStarLine = parsers.asterisk^4 + parsers.underscore^4
5607     / writer.string
5608
5609     parsers.EscapedChar = parsers.backslash * C(parsers.escapable) / writer.string
5610
5611     parsers.InlineHtml = parsers.emptyelt_any / writer.inline_html_tag
5612     + (parsers.htmlcomment / self.parser_functions.parse_inlines_no
5613     / writer.inline_html_comment
5614     + parsers.htmlinstruction
5615     + parsers.openelt_any / writer.inline_html_tag
5616     + parsers.closeelt_any / writer.inline_html_tag
5617
5618     parsers.HtmlEntity = parsers.hexentity / entities.hex_entity / writer.string
5619     + parsers.decentity / entities.dec_entity / writer.string

```

```
5620                                     + parsers.tagentity / entities.char_entity / writer.string
```

3.1.5.8 Block Elements (local)

```
5621 parsers.DisplayHtml = (parsers.htmlcomment / self.parser_functions.parse_blocks_nest
5622                        / writer.block_html_comment
5623                        + parsers.emptyelt_block / writer.block_html_element
5624                        + parsers.openelt_exact("hr") / writer.block_html_element
5625                        + parsers.in_matched_block_tags / writer.block_html_element
5626                        + parsers.htmlinstruction
5627
5628 parsers.Verbatim      = Cs( (parsers.blanklines
5629                            * ((parsers.indentedline - parsers.blankline))^1)^1
5630                            ) / self.expandtabs / writer.verbatim
5631
5632 parsers.Blockquote    = Cs(parsers.blockquote_body^1)
5633                        / self.parser_functions.parse_blocks_nested
5634                        / writer.blockquote
5635
5636 parsers.HorizontalRule = ( parsers.lineof(parsers.asterisk)
5637                            + parsers.lineof(parsers.dash)
5638                            + parsers.lineof(parsers.underscore)
5639                            ) / writer.hrule
5640
5641 parsers.Reference     = parsers.define_reference_parser / register_link
5642
5643 parsers.Paragraph     = parsers.nonindentspace * Ct(parsers.Inline^1)
5644                        * ( parsers.newline
5645                        * ( parsers.blankline^1
5646                        + #parsers.hash
5647                        + #(parsers.leader * parsers.more * parsers.space^-
5648                        1)
5649                        + parsers.eof
5650                        )
5651                        + parsers.eof )
5652                        / writer.paragraph
5653
5653 parsers.Plain         = parsers.nonindentspace * Ct(parsers.Inline^1)
5654                        / writer.plain
```

3.1.5.9 Lists (local)

```
5655 parsers.starter = parsers.bullet + parsers.enumerator
5656
5657 if options.taskLists then
5658     parsers.tickbox = ( parsers.ticked_box
5659                       + parsers.halfticked_box
5660                       + parsers.unticked_box
```

```

5661             ) / writer.tickbox
5662     else
5663         parsers.tickbox = parsers.fail
5664     end
5665
5666     -- we use \001 as a separator between a tight list item and a
5667     -- nested list under it.
5668     parsers.NestedList = Cs((parsers.optionallyindentedline
5669                             - parsers.starter)^1)
5670                             / function(a) return "\001"..a end
5671
5672     parsers.ListBlockLine = parsers.optionallyindentedline
5673                             - parsers.blankline - (parsers.indent~-1
5674                                                     * parsers.starter)
5675
5676     parsers.ListBlock = parsers.line * parsers.ListBlockLine^0
5677
5678     parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
5679                                     * parsers.ListBlock
5680
5681     parsers.TightListItem = function(starter)
5682         return -parsers.HorizontalRule
5683             * (Cs(starter / "" * parsers.tickbox~-1 * parsers.ListBlock * parsers.Nested
5684 1)
5685                 / self.parser_functions.parse_blocks_nested)
5686                 * -(parsers.blanklines * parsers.indent)
5687     end
5688
5689     parsers.LooseListItem = function(starter)
5690         return -parsers.HorizontalRule
5691             * Cs( starter / "" * parsers.tickbox~-1 * parsers.ListBlock * Cc("\n")
5692                 * (parsers.NestedList + parsers.ListContinuationBlock^0)
5693                 * (parsers.blanklines / "\n\n")
5694                 ) / self.parser_functions.parse_blocks_nested
5695     end
5696
5697     parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
5698                             * parsers.skipblanklines * -parsers.bullet
5699                             + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
5700                             * parsers.skipblanklines )
5701     / writer.bulletlist
5702
5703     local function ordered_list(items,tight,startnum)
5704         if options.startNumber then
5705             startnum = tonumber(startnum) or 1 -- fallback for '#'
5706             if startnum ~= nil then
5707                 startnum = math.floor(startnum)

```

```

5707     end
5708   else
5709     startnum = nil
5710   end
5711   return writer.orderedlist(items,tight,startnum)
5712 end
5713
5714 parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
5715   ( Ct(parsers.TightListItem(Cb("listtype")))
5716     * parsers.TightListItem(parsers.enumerator)^0)
5717   * Cc(true) * parsers.skipblanklines * -parsers.enumerator
5718   + Ct(parsers.LooseListItem(Cb("listtype")))
5719     * parsers.LooseListItem(parsers.enumerator)^0)
5720   * Cc(false) * parsers.skipblanklines
5721   ) * Cb("listtype") / ordered_list

```

3.1.5.10 Blank (local)

```

5722 parsers.Blank      = parsers.blankline / ""
5723                   + parsers.Reference
5724                   + (parsers.tightblocksep / "\n")

```

3.1.5.11 Headings (local)

```

5725 -- parse atx header
5726 parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
5727                   * parsers.optionalspace
5728                   * (C(parsers.line)
5729                     / strip_atx_end
5730                     / self.parser_functions.parse_inlines)
5731                   * Cb("level")
5732                   / writer.heading
5733
5734 parsers.SetextHeading = #(parsers.line * S("--"))
5735                   * Ct(parsers.linechar^1
5736                     / self.parser_functions.parse_inlines)
5737                   * parsers.newline
5738                   * parsers.HeadingLevel
5739                   * parsers.optionalspace
5740                   * parsers.newline
5741                   / writer.heading
5742
5743 parsers.Heading = parsers.AtxHeading + parsers.SetextHeading

```

3.1.5.12 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 TeX output.

```
5744 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.

```
5745     local walkable_syntax = (function(global_walkable_syntax)
5746         local local_walkable_syntax = {}
5747         for lhs, rule in pairs(global_walkable_syntax) do
5748             local_walkable_syntax[lhs] = util.table_copy(rule)
5749         end
5750         return local_walkable_syntax
5751     end)(walkable_syntax)
```

Define `reader->insert_pattern` as a function that receives two 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. The function adds the pattern to `walkable_syntax[left-hand side terminal symbol]` before or after *right-hand side terminal symbol*.

```
5752     self.insert_pattern = function(selector, pattern)
5753         local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
5754         assert(lhs ~= nil,
5755             [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
5756             .. selector .. [[]])
5757         assert(walkable_syntax[lhs] ~= nil,
5758             [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
5759         assert(pos == "before" or pos == "after" or pos == "instead of",
5760             [[Expected positional specifier "before", "after", or "instead of", not "]]
5761             .. pos .. [[]])
5762         local rule = walkable_syntax[lhs]
5763         local index = nil
5764         for current_index, current_rhs in ipairs(rule) do
5765             if type(current_rhs) == "string" and current_rhs == rhs then
5766                 index = current_index
5767                 if pos == "after" then
5768                     index = index + 1
5769                 end
5770                 break
5771             end
5772         end
5773         assert(index ~= nil,
5774             [[Rule ]] .. lhs .. [[ -> ]] .. rhs
5775             .. [[ does not exist in markdown grammar]])
5776         if pos == "instead of" then
```

```

5777     rule[index] = pattern
5778   else
5779     table.insert(rule, index, pattern)
5780   end
5781 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.

```

5782   local syntax =
5783     { "Blocks",
5784
5785       Blocks = ( V("ExpectedJekyllData")
5786                 * (V("Blank")^0 / writer.interblocksep)
5787                 )^-1
5788                 * V("Blank")^0
5789                 * V("Block")^-1
5790                 * (V("Blank")^0 / writer.interblocksep
5791                   * V("Block"))^0
5792                 * V("Blank")^0 * parsers.eof,
5793
5794       ExpectedJekyllData = parsers.fail,
5795
5796       Blank = parsers.Blank,
5797
5798       Blockquote = parsers.Blockquote,
5799       Verbatim = parsers.Verbatim,
5800       HorizontalRule = parsers.HorizontalRule,
5801       BulletList = parsers.BulletList,
5802       OrderedList = parsers.OrderedList,
5803       Heading = parsers.Heading,
5804       DisplayHtml = parsers.DisplayHtml,
5805       Paragraph = parsers.Paragraph,
5806       Plain = parsers.Plain,
5807       EndlineExceptions = parsers.EndlineExceptions,
5808
5809       Str = parsers.Str,
5810       Space = parsers.Space,
5811       OptionalIndent = parsers.OptionalIndent,
5812       Endline = parsers.Endline,
5813       U1OrStarLine = parsers.U1OrStarLine,
5814       Strong = parsers.Strong,
5815       Emph = parsers.Emph,
5816       Link = parsers.Link,
5817       Image = parsers.Image,
5818       Code = parsers.Code,
5819       AutoLinkUrl = parsers.AutoLinkUrl,
5820       AutoLinkEmail = parsers.AutoLinkEmail,

```

```

5821     AutoLinkRelativeReference
5822         = parsers.AutoLinkRelativeReference,
5823     InlineHtml
5824         = parsers.InlineHtml,
5825     HtmlEntity
5826         = parsers.HtmlEntity,
5827     EscapedChar
5828         = parsers.EscapedChar,
5829     Smart
5830         = parsers.Smart,
5831     Symbol
5832         = parsers.Symbol,
5833     SpecialChar
5834         = parsers.fail,
5835 }

```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a PEG pattern. The function (re)defines `walkable_syntax[left-hand side terminal symbol]` to be equal to pattern.

```

5830     self.update_rule = function(rule_name, pattern)
5831     assert(syntax[rule_name] ~= nil,
5832         [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
5833     walkable_syntax[rule_name] = { pattern }
5834     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.

```

5835     local special_characters = {}
5836     self.add_special_character = function(c)
5837         table.insert(special_characters, c)
5838         syntax.SpecialChar = S(table.concat(special_characters, ""))
5839     end
5840
5841     self.add_special_character("*")
5842     self.add_special_character("`")
5843     self.add_special_character("[")
5844     self.add_special_character("]")
5845     self.add_special_character("<")
5846     self.add_special_character("!")
5847     self.add_special_character("\\")

```

Apply syntax extensions.

```

5848     for _, extension in ipairs(extensions) do
5849         extension.extend_writer(writer)
5850         extension.extend_reader(self)
5851     end

```

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

```

5852     walkable_syntax["IndentedInline"] = util.table_copy(
5853         walkable_syntax["Inline"])
5854     self.insert_pattern(
5855         "IndentedInline instead of Space",

```

```
5856     "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`.

```
5857     for lhs, rule in pairs(walkable_syntax) do
5858         syntax[lhs] = parsers.fail
5859         for _, rhs in ipairs(rule) do
5860             local pattern
5861             if type(rhs) == "string" then
5862                 pattern = V(rhs)
5863             else
5864                 pattern = rhs
5865             end
5866             syntax[lhs] = syntax[lhs] + pattern
5867         end
5868     end
```

Finalize the parser by enabling built-in syntax extensions and producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```
5869     if options.underscores then
5870         self.add_special_character("_")
5871     end
5872
5873     if not options.codeSpans then
5874         syntax.Code = parsers.fail
5875     end
5876
5877     if not options.html then
5878         syntax.DisplayHtml = parsers.fail
5879         syntax.InlineHtml = parsers.fail
5880         syntax.HtmlEntity = parsers.fail
5881     else
5882         self.add_special_character("&")
5883     end
5884
5885     if options.preserveTabs then
5886         options.stripIndent = false
5887     end
5888
5889     if not options.smartEllipses then
5890         syntax.Smart = parsers.fail
5891     else
5892         self.add_special_character(".")
5893     end
5894
```



```

5895     if not options.relativeReferences then
5896         syntax.AutoLinkRelativeReference = parsers.fail
5897     end
5898
5899     local blocks_nested_t = util.table_copy(syntax)
5900     blocks_nested_t.ExpectedJekyllData = parsers.fail
5901     parsers.blocks_nested = Ct(blocks_nested_t)
5902
5903     parsers.blocks = Ct(syntax)
5904
5905     local inlines_t = util.table_copy(syntax)
5906     inlines_t[1] = "Inlines"
5907     inlines_t.Inlines = parsers.Inline^0 * (parsers.spacing^0 * parsers.eof / "")
5908     parsers.inlines = Ct(inlines_t)
5909
5910     local inlines_no_link_t = util.table_copy(inlines_t)
5911     inlines_no_link_t.Link = parsers.fail
5912     parsers.inlines_no_link = Ct(inlines_no_link_t)
5913
5914     local inlines_no_inline_note_t = util.table_copy(inlines_t)
5915     inlines_no_inline_note_t.InlineNote = parsers.fail
5916     parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
5917
5918     local inlines_no_html_t = util.table_copy(inlines_t)
5919     inlines_no_html_t.DisplayHtml = parsers.fail
5920     inlines_no_html_t.InlineHtml = parsers.fail
5921     inlines_no_html_t.HtmlEntity = parsers.fail
5922     parsers.inlines_no_html = Ct(inlines_no_html_t)
5923
5924     local inlines_nbsp_t = util.table_copy(inlines_t)
5925     inlines_nbsp_t.Endline = parsers.NonbreakingEndline
5926     inlines_nbsp_t.Space = parsers.NonbreakingSpace
5927     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.

```

5928     return function(input)
5929         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.

```

5930     local opt_string = {}
5931     for k, _ in pairs(defaultOptions) do
5932         local v = options[k]
5933         if type(v) == "table" then
5934             for _, i in ipairs(v) do
5935                 opt_string[#opt_string+1] = k .. "=" .. tostring(i)

```

```

5936         end
5937     elseif k ~= "cacheDir" then
5938         opt_string[#opt_string+1] = k .. "=" .. tostring(v)
5939     end
5940 end
5941 table.sort(opt_string)
5942 local salt = table.concat(opt_string, ",") .. "," .. metadata.version
5943 local output

```

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

```

5944     local function convert(input)
5945         local document = self.parser_functions.parse_blocks(input)
5946         return util.rope_to_string(writer.document(document))
5947     end
5948     if options.eagerCache or options.finalizeCache then
5949         local name = util.cache(options.cacheDir, input, salt, convert,
5950                                ".md" .. writer.suffix)
5951         output = writer.pack(name)

```

Otherwise, return the result of the conversion directly.

```

5952     else
5953         output = convert(input)
5954     end

```

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

```

5955     if options.finalizeCache then
5956         local file, mode
5957         if options.frozenCacheCounter > 0 then
5958             mode = "a"
5959         else
5960             mode = "w"
5961         end
5962         file = assert(io.open(options.frozenCacheFileName, mode),
5963                        [[Could not open file ]] .. options.frozenCacheFileName
5964                        .. [[ for writing]])
5965         assert(file:write([[\\expandafter\\global\\expandafter\\def\\csname ]]
5966                           .. [[markdownFrozenCache]] .. options.frozenCacheCounter
5967                           .. [[\\endcsname{]] .. output .. [[]]] .. "\\n"))
5968         assert(file:close())
5969     end
5970     return output
5971 end
5972 end
5973 return self
5974 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.

```
5975 M.extensions = {}
```

3.1.6.1 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.

```
5976 M.extensions.citations = function(citation_nbsps)
```

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

```
5977   local escaped_citation_chars = {
5978     ["{"] = "\\markdownRendererLeftBrace{}",
5979     ["}"] = "\\markdownRendererRightBrace{}",
5980     ["%"] = "\\markdownRendererPercentSign{}",
5981     ["\\"] = "\\markdownRendererBackslash{}",
5982     ["#"] = "\\markdownRendererHash{}",
5983   }
5984   return {
5985     extend_writer = function(self)
5986       local options = self.options
5987
```

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

```
5988     local escape_citation = util.escaper(
5989       escaped_citation_chars,
5990       self.escaped_minimal_strings)
```

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

```
5991     if options.hybrid then
5992       self.citation = self.escape_minimal
5993     else
5994       self.citation = escape_citation
5995     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.

```

5996     function self.citations(text_cites, cites)
5997         local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
5998             "{", #cites, "}"}
5999         for _,cite in ipairs(cites) do
6000             buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6001                 cite.prenote or "", "}{" , cite.postnote or "", "}{" , cite.name, "}"}
6002         end
6003         return buffer
6004     end
6005 end, extend_reader = function(self)
6006     local parsers = self.parsers
6007     local writer = self.writer
6008
6009     local citation_chars
6010         = parsers.alphanumeric
6011         + S("#$%&~+<>~/_")
6012
6013     local citation_name
6014         = Cs(parsers.dash^-1) * parsers.at
6015         * Cs(citation_chars
6016             * (((citation_chars + parsers.internal_punctuation
6017                 - parsers.comma - parsers.semicolon)
6018                 * -#((parsers.internal_punctuation - parsers.comma
6019                     - parsers.semicolon)^0
6020                     * -(citation_chars + parsers.internal_punctuation
6021                         - parsers.comma - parsers.semicolon)))^0
6022                 * citation_chars)^-1)
6023
6024     local citation_body_prenote
6025         = Cs((parsers.alphanumeric^1
6026             + parsers.bracketed
6027             + parsers.inticks
6028             + (parsers.anyescaped
6029                 - (parsers.rbracket + parsers.blankline^2))

```

```

6030         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6031
6032     local citation_body_postnote
6033         = Cs((parsers.alphanumeric^1
6034             + parsers.bracketed
6035             + parsers.inticks
6036             + (parsers.anyescaped
6037               - (parsers.rbracket + parsers.semicolon
6038                 + parsers.blankline^2))
6039             - (parsers.spnl * parsers.rbracket))^0)
6040
6041     local citation_body_chunk
6042         = citation_body_prenote
6043         * parsers.spnl * citation_name
6044         * (parsers.internal_punctuation - parsers.semicolon)^-
1
6045         * parsers.spnl * citation_body_postnote
6046
6047     local citation_body
6048         = citation_body_chunk
6049         * (parsers.semicolon * parsers.spnl
6050           * citation_body_chunk)^0
6051
6052     local citation_headless_body_postnote
6053         = Cs((parsers.alphanumeric^1
6054             + parsers.bracketed
6055             + parsers.inticks
6056             + (parsers.anyescaped
6057               - (parsers.rbracket + parsers.at
6058                 + parsers.semicolon + parsers.blankline^2))
6059             - (parsers.spnl * parsers.rbracket))^0)
6060
6061     local citation_headless_body
6062         = citation_headless_body_postnote
6063         * (parsers.sp * parsers.semicolon * parsers.spnl
6064           * citation_body_chunk)^0
6065
6066     local citations
6067         = function(text_cites, raw_cites)
6068         local function normalize(str)
6069             if str == "" then
6070                 str = nil
6071             else
6072                 str = (citation_nbsps and
6073                       self.parser_functions.parse_inlines_nbsp or
6074                       self.parser_functions.parse_inlines)(str)
6075             end

```

```

6076         return str
6077     end
6078
6079     local cites = {}
6080     for i = 1,#raw_cites,4 do
6081         cites[#cites+1] = {
6082             prenote = normalize(raw_cites[i]),
6083             suppress_author = raw_cites[i+1] == "-",
6084             name = writer.citation(raw_cites[i+2]),
6085             postnote = normalize(raw_cites[i+3]),
6086         }
6087     end
6088     return writer.citations(text_cites, cites)
6089 end
6090
6091 local TextCitations
6092     = Ct((parsers.spnl
6093         * Cc("")
6094         * citation_name
6095         * ((parsers.spnl
6096             * parsers.lbracket
6097             * citation_headless_body
6098             * parsers.rbracket) + Cc("")))^1)
6099 / function(raw_cites)
6100     return citations(true, raw_cites)
6101 end
6102
6103 local ParenthesizedCitations
6104     = Ct((parsers.spnl
6105         * parsers.lbracket
6106         * citation_body
6107         * parsers.rbracket)^1)
6108 / function(raw_cites)
6109     return citations(false, raw_cites)
6110 end
6111
6112 local Citations = TextCitations + ParenthesizedCitations
6113
6114 self.insert_pattern("Inline after Emph", Citations)
6115
6116 self.add_special_character("@")
6117 self.add_special_character("-")
6118 end
6119 }
6120 end

```

3.1.6.2 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.

```
6121 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.

```
6122   local languages_json = (function()
6123     local base, prev, curr
6124     for _, pathname in ipairs(util.lookup_files(language_map, { all=true })) do
6125       local file = io.open(pathname, "r")
6126       if not file then goto continue end
6127       local input = assert(file:read("*a"))
6128       assert(file:close())
6129       local json = input:gsub('("[^\\n]-"):', '[%1]=')
6130       curr = load("_ENV = {}; return \"..json()\")
6131       if type(curr) == "table" then
6132         if base == nil then
6133           base = curr
6134         else
6135           setmetatable(prev, { __index = curr })
6136         end
6137         prev = curr
6138       end
6139       ::continue::
6140     end
6141     return base or {}
6142   end)()
6143
6144   return {
6145     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.

```
6146     function self.contentblock(src,suf,type,tit)
6147       if not self.is_writing then return "" end
6148       src = src..".."..suf
6149       suf = suf:lower()
6150       if type == "onlineimage" then
6151         return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
6152               {"{",self.string(src),"}",
6153               {"{",self.uri(src),"}",
6154               {"{",self.string(tit or ""),"}}
```

```

6155     elseif languages_json[suf] then
6156         return {"\\markdownRendererContentBlockCode{" , suf, "}" ,
6157               "{" , self.string(languages_json[suf]) , "}" ,
6158               "{" , self.string(src) , "}" ,
6159               "{" , self.uri(src) , "}" ,
6160               "{" , self.string(tit or "") , "}" }
6161     else
6162         return {"\\markdownRendererContentBlock{" , suf, "}" ,
6163               "{" , self.string(src) , "}" ,
6164               "{" , self.uri(src) , "}" ,
6165               "{" , self.string(tit or "") , "}" }
6166     end
6167 end
6168 end, extend_reader = function(self)
6169     local parsers = self.parsers
6170     local writer = self.writer
6171
6172     local contentblock_tail
6173         = parsers.optionaltitle
6174         * (parsers.newline + parsers.eof)
6175
6176     -- case insensitive online image suffix:
6177     local onlineimagesuffix
6178         = (function(...)
6179             local parser = nil
6180             for _, suffix in ipairs({...}) do
6181                 local pattern=nil
6182                 for i=1,#suffix do
6183                     local char=suffix:sub(i,i)
6184                     char = S(char:lower()..char:upper())
6185                     if pattern == nil then
6186                         pattern = char
6187                     else
6188                         pattern = pattern * char
6189                     end
6190                 end
6191                 if parser == nil then
6192                     parser = pattern
6193                 else
6194                     parser = parser + pattern
6195                 end
6196             end
6197             return parser
6198         end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6199
6200     -- online image url for iA Writer content blocks with mandatory suffix,
6201     -- allowing nested brackets:

```



```

6202     local onlineimageurl
6203         = (parsers.less
6204             * Cs((parsers.anyescaped
6205                 - parsers.more
6206                 - #(parsers.period
6207                     * onlineimagesuffix
6208                     * parsers.more
6209                     * contentblock_tail))^0)
6210             * parsers.period
6211             * Cs(onlineimagesuffix)
6212             * parsers.more
6213             + (Cs((parsers.inparens
6214                 + (parsers.anyescaped
6215                     - parsers.spacing
6216                     - parsers.rparent
6217                     - #(parsers.period
6218                         * onlineimagesuffix
6219                         * contentblock_tail))))^0)
6220             * parsers.period
6221             * Cs(onlineimagesuffix))
6222             ) * Cc("onlineimage")
6223
6224     -- filename for iA Writer content blocks with mandatory suffix:
6225     local localfilepath
6226         = parsers.slash
6227         * Cs((parsers.anyescaped
6228             - parsers.tab
6229             - parsers.newline
6230             - #(parsers.period
6231                 * parsers.alphanumeric^1
6232                 * contentblock_tail))^1)
6233         * parsers.period
6234         * Cs(parsers.alphanumeric^1)
6235         * Cc("localfile")
6236
6237     local ContentBlock
6238         = parsers.leader
6239         * (localfilepath + onlineimageurl)
6240         * contentblock_tail
6241         / writer.contentblock
6242
6243     self.insert_pattern("Block before Blockquote", ContentBlock)
6244 end
6245 }
6246 end

```

3.1.6.3 Definition Lists The `extensions.definition_lists` function implements

the definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```

6247 M.extensions.definition_lists = function(tight_lists)
6248   return {
6249     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.
6250       local function dliitem(term, defs)
6251         local retVal = {"\\markdownRenderDlItem{" ,term,""}
6252         for _, def in ipairs(defs) do
6253           retVal[#retVal+1] = {"\\markdownRenderDlDefinitionBegin " ,def,
6254                                "\\markdownRenderDlDefinitionEnd "}
6255         end
6256         retVal[#retVal+1] = "\\markdownRenderDlItemEnd "
6257         return retVal
6258       end
6259
6260       function self.definitionlist(items,tight)
6261         if not self.is_writing then return "" end
6262         local buffer = {}
6263         for _,item in ipairs(items) do
6264           buffer[#buffer + 1] = dliitem(item.term, item.definitions)
6265         end
6266         if tight and tight_lists then
6267           return {"\\markdownRenderDlBeginTight\\n", buffer,
6268                  "\\n\\markdownRenderDlEndTight"}
6269         else
6270           return {"\\markdownRenderDlBegin\\n", buffer,
6271                  "\\n\\markdownRenderDlEnd"}
6272         end
6273       end
6274     end, extend_reader = function(self)
6275       local parsers = self.parsers
6276       local writer = self.writer
6277
6278       local defstartchar = S("~:")
6279
6280       local defstart = ( defstartchar * #parsers.spacing
6281                          * (parsers.tab + parsers.space^-3)
6282                          + parsers.space * defstartchar * #parsers.spacing
6283                          * (parsers.tab + parsers.space^-2)
6284                          + parsers.space * parsers.space * defstartchar
6285                          * #parsers.spacing
6286                          * (parsers.tab + parsers.space^-1)

```

```

6287             + parsers.space * parsers.space * parsers.space
6288                 * defstartchar * #parsers.spacing
6289         )
6290
6291     local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)
6292
6293     local function definition_list_item(term, defs, _)
6294         return { term = self.parser_functions.parse_inlines(term),
6295                 definitions = defs }
6296     end
6297
6298     local DefinitionListItemLoose
6299         = C(parsers.line) * parsers.skipblanklines
6300         * Ct((defstart
6301             * parsers.indented_blocks(dlchunk)
6302             / self.parser_functions.parse_blocks_nested)^1)
6303         * Cc(false) / definition_list_item
6304
6305     local DefinitionListItemTight
6306         = C(parsers.line)
6307         * Ct((defstart * dlchunk
6308             / self.parser_functions.parse_blocks_nested)^1)
6309         * Cc(true) / definition_list_item
6310
6311     local DefinitionList
6312         = ( Ct(DefinitionListItemLoose^1) * Cc(false)
6313           + Ct(DefinitionListItemTight^1)
6314           * (parsers.skipblanklines
6315             * -DefinitionListItemLoose * Cc(true))
6316           ) / writer.definitionlist
6317
6318     self.insert_pattern("Block after Heading", DefinitionList)
6319 end
6320 }
6321 end

```

3.1.6.4 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 between a paragraph and the following fenced code block.

```

6322 M.extensions.fenced_code = function(blank_before_code_fence)
6323     return {
6324         extend_writer = function(self)
6325             local options = self.options
6326

```

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

```

6327     function self.fencedCode(i, s)
6328         if not self.is_writing then return "" end
6329         s = string.gsub(s, '[\r\n%s]*$', '')
6330         local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
6331         return {"\\markdownRendererInputFencedCode{"..name.."}{"..i.."}"}
6332     end
6333 end, extend_reader = function(self)
6334     local parsers = self.parsers
6335     local writer = self.writer
6336
6337     local function captures_geq_length(_, i, a, b)
6338         return #a >= #b and i
6339     end
6340
6341     local infostring = (parsers.linechar - (parsers.backtick
6342         + parsers.space^1 * (parsers.newline + parsers.eof)))^0
6343
6344     local fenceindent
6345     local fencehead = function(char)
6346         return
6347             C(parsers.nonindentspace) / function(s) fenceindent = #s end
6348             * Cg(char^3, "fencelength")
6349             * parsers.optionalspace * C(infostring)
6350             * parsers.optionalspace * (parsers.newline + parsers.eof)
6351     end
6352
6353     local fencetail = function(char)
6354         return
6355             parsers.nonindentspace
6356             * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
6357             * parsers.optionalspace * (parsers.newline + parsers.eof)
6358             + parsers.eof
6359     end
6360
6361     local fencedline = function(char)
6362         return
6363             C(parsers.line - fencetail(char))
6364             / function(s)
6365                 local i = 1
6366                 local remaining = fenceindent
6367                 while true do
6368                     local c = s:sub(i, i)
6369                     if c == " " and remaining > 0 then
6370                         remaining = remaining - 1
6371                         i = i + 1
6372                     elseif c == "\t" and remaining > 3 then
6373                         remaining = remaining - 4
6374                         i = i + 1

```

```

6372             else
6373                 break
6374             end
6375         end
6376         return s:sub(i)
6377     end
6378 end
6379
6380 local TildeFencedCode
6381     = fencehead(parsers.tilde)
6382     * Cs(fencedline(parsers.tilde)^0)
6383     * fencetail(parsers.tilde)
6384
6385 local BacktickFencedCode
6386     = fencehead(parsers.backtick)
6387     * Cs(fencedline(parsers.backtick)^0)
6388     * fencetail(parsers.backtick)
6389
6390 local FencedCode = (TildeFencedCode
6391                     + BacktickFencedCode)
6392 / function(infostring, code)
6393     return writer.fencedCode(writer.string(infostring),
6394                             self.expandtabs(code))
6395 end
6396
6397 self.insert_pattern("Block after Verbatim",
6398                    FencedCode)
6399
6400 local fencestart
6401 if blank_before_code_fence then
6402     fencestart = parsers.fail
6403 else
6404     fencestart = fencehead(parsers.backtick)
6405                 + fencehead(parsers.tilde)
6406 end
6407
6408 local EndlineExceptions = parsers.EndlineExceptions + fencestart
6409 self.update_rule("EndlineExceptions", EndlineExceptions)
6410
6411 self.add_special_character("~")
6412 end
6413 }
6414 end

```

3.1.6.5 Footnotes The `extensions.footnotes` function implements the Pandoc footnote and inline footnote syntax extensions. When the `footnote` parame-

ter is `true`, the Pandoc footnote syntax extension will be enabled. When the `inline_footnotes` parameter is `true`, the Pandoc inline footnote syntax extension will be enabled.

```
6415 M.extensions.footnotes = function(footnotes, inline_footnotes)
6416   assert(footnotes or inline_footnotes)
6417   return {
6418     extend_writer = function(self)
```

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

```
6419     function self.note(s)
6420       return {"\\markdownRendererFootnote{",s,""}
6421     end
6422   end, extend_reader = function(self)
6423     local parsers = self.parsers
6424     local writer = self.writer
6425
6426     if inline_footnotes then
6427       local InlineNote
6428         = parsers.circumflex
6429         * (parsers.tag / self.parser_functions.parse_inlines_no_inline_note)
6430         / writer.note
6431
6432       self.insert_pattern("Inline after Emph", InlineNote)
6433     end
6434     if footnotes then
6435       local function strip_first_char(s)
6436         return s:sub(2)
6437       end
6438
6439       local RawNoteRef
6440         = #(parsers.lbracket * parsers.circumflex)
6441         * parsers.tag / strip_first_char
6442
6443       local rawnotes = {}
6444
6445       -- like indirect_link
6446       local function lookup_note(ref)
6447         return writer.defer_call(function()
6448           local found = rawnotes[self.normalize_tag(ref)]
6449           if found then
6450             return writer.note(
6451               self.parser_functions.parse_blocks_nested(found))
6452           else
6453             return {"[",
6454               self.parser_functions.parse_inlines("^" .. ref), ""]}
6455           end
```

```

6456         end)
6457     end
6458
6459     local function register_note(ref,rawnote)
6460         rawnotes[self.normalize_tag(ref)] = rawnote
6461         return ""
6462     end
6463
6464     local NoteRef = RawNoteRef / lookup_note
6465
6466     local NoteBlock
6467         = parsers.leader * RawNoteRef * parsers.colon
6468         * parsers.spnl * parsers.indented_blocks(parsers.chunk)
6469         / register_note
6470
6471     local Blank = NoteBlock + parsers.Blank
6472     self.update_rule("Blank", Blank)
6473
6474     self.insert_pattern("Inline after Emph", NoteRef)
6475 end
6476
6477 self.add_special_character("^")
6478 end
6479 }
6480 end

```

3.1.6.6 Header Attributes The `extensions.header_attributes` function implements a syntax extension that enables the assignment of HTML attributes to headings.

```

6481 M.extensions.header_attributes = function()
6482     return {
6483         extend_writer = function()
6484             end, extend_reader = function(self)
6485                 local parsers = self.parsers
6486                 local writer = self.writer
6487
6488                 parsers.AtxHeading = Cg(parsers.HeadingStart,"level")
6489                     * parsers.optionalspace
6490                     * (C(((parsers.linechar
6491                         - ((parsers.hash^1
6492                             * parsers.optionalspace
6493                             * parsers.HeadingAttributes^-1
6494                             + parsers.HeadingAttributes)
6495                             * parsers.optionalspace
6496                             * parsers.newline))
6497                         * (parsers.linechar
6498                             - parsers.hash

```

```

6499         - parsers.lbrace)^0)^1)
6500         / self.parser_functions.parse_inlines)
6501     * Cg(Ct(parsers.newline
6502         + (parsers.hash^1
6503           * parsers.optionalspace
6504           * parsers.HeadingAttributes~-1
6505           + parsers.HeadingAttributes)
6506           * parsers.optionalspace
6507           * parsers.newline), "attributes")
6508     * Cb("level")
6509     * Cb("attributes")
6510     / writer.heading
6511
6512     parsers.SettextHeading = #(parsers.line * S("="))
6513     * (C(((parsers.linechar
6514         - (parsers.HeadingAttributes
6515           * parsers.optionalspace
6516           * parsers.newline))
6517       * (parsers.linechar
6518         - parsers.lbrace)^0)^1)
6519     / self.parser_functions.parse_inlines)
6520     * Cg(Ct(parsers.newline
6521         + (parsers.HeadingAttributes
6522           * parsers.optionalspace
6523           * parsers.newline)), "attributes")
6524     * parsers.HeadingLevel
6525     * Cb("attributes")
6526     * parsers.optionalspace
6527     * parsers.newline
6528     / writer.heading
6529
6530     local Heading = parsers.AtHeading + parsers.SettextHeading
6531     self.update_rule("Heading", Heading)
6532 end
6533 }
6534 end

```

3.1.6.7 YAML Metadata The `extensions.jekyll_data` function implements the Pandoc `yml_metadata_block` syntax extension for entering metadata in YAML. 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

```

6535 M.extensions.jekyll_data = function(expect_jekyll_data)
6536   return {
6537     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`.

```
6538     function self.jekyllData(d, t, p)
6539         if not self.is_writing then return "" end
6540
6541         local buf = {}
6542
6543         local keys = {}
6544         for k, _ in pairs(d) do
6545             table.insert(keys, k)
6546         end
6547         table.sort(keys)
6548
6549         if not p then
6550             table.insert(buf, "\\markdownRendererJekyllDataBegin")
6551         end
6552
6553         if #d > 0 then
6554             table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
6555             table.insert(buf, self.uri(p or "null"))
6556             table.insert(buf, "{")
6557             table.insert(buf, #keys)
6558             table.insert(buf, "}")
6559         else
6560             table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
6561             table.insert(buf, self.uri(p or "null"))
6562             table.insert(buf, "{")
6563             table.insert(buf, #keys)
6564             table.insert(buf, "}")
6565         end
6566
6567         for _, k in ipairs(keys) do
6568             local v = d[k]
6569             local typ = type(v)
6570             k = tostring(k or "null")
6571             if typ == "table" and next(v) ~= nil then
6572                 table.insert(
6573                     buf,
6574                     self.jekyllData(v, t, k)
6575                 )
6576             else
6577                 k = self.uri(k)
6578                 v = tostring(v)
6579                 if typ == "boolean" then
6580                     table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
6581                     table.insert(buf, k)
```

```

6582         table.insert(buf, "{")
6583         table.insert(buf, v)
6584         table.insert(buf, "}")
6585     elseif typ == "number" then
6586         table.insert(buf, "\\markdownRendererJekyllDataNumber{")
6587         table.insert(buf, k)
6588         table.insert(buf, "{")
6589         table.insert(buf, v)
6590         table.insert(buf, "}")
6591     elseif typ == "string" then
6592         table.insert(buf, "\\markdownRendererJekyllDataString{")
6593         table.insert(buf, k)
6594         table.insert(buf, "{")
6595         table.insert(buf, t(v))
6596         table.insert(buf, "}")
6597     elseif typ == "table" then
6598         table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
6599         table.insert(buf, k)
6600         table.insert(buf, "}")
6601     else
6602         error(format("Unexpected type%s for value of " ..
6603                     "YAML key %s", typ, k))
6604     end
6605 end
6606 end
6607
6608 if #d > 0 then
6609     table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
6610 else
6611     table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
6612 end
6613
6614 if not p then
6615     table.insert(buf, "\\markdownRendererJekyllDataEnd")
6616 end
6617
6618 return buf
6619 end
6620 end, extend_reader = function(self)
6621     local parsers = self.parsers
6622     local writer = self.writer
6623
6624     local JekyllData
6625         = Cmt( C((parsers.line - P("---") - P("..."))^0)
6626             , function(s, i, text) -- luacheck: ignore s i
6627                 local data
6628                 local ran_ok, _ = pcall(function()

```

```

6629         local tinyyaml = require("markdown-tinyyaml")
6630         data = tinyyaml.parse(text, {timestamps=false})
6631     end)
6632     if ran_ok and data ~= nil then
6633         return true, writer.jekyllData(data, function(s)
6634             return self.parser_functions.parse_blocks_nested(s)
6635         end, nil)
6636     else
6637         return false
6638     end
6639 end
6640 )
6641
6642 local UnexpectedJekyllData
6643     = P("---")
6644     * parsers.blankline / 0
6645     * #(-parsers.blankline) -- if followed by blank, it's an hrule
6646     * JekyllData
6647     * (P("---") + P("..."))
6648
6649 local ExpectedJekyllData
6650     = ( P("---")
6651         * parsers.blankline / 0
6652         * #(-parsers.blankline) -- if followed by blank, it's an hrule
6653         )~-1
6654     * JekyllData
6655     * (P("---") + P("..."))~-1
6656
6657 self.insert_pattern("Block before Blockquote", UnexpectedJekyllData)
6658 if expect_jekyll_data then
6659     self.update_rule("ExpectedJekyllData", ExpectedJekyllData)
6660 end
6661 end
6662 }
6663 end

```

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

```

6664 M.extensions.pipe_tables = function(table_captions)
6665
6666     local function make_pipe_table_rectangular(rows)
6667         local num_columns = #rows[2]
6668         local rectangular_rows = {}
6669         for i = 1, #rows do

```

```

6670     local row = rows[i]
6671     local rectangular_row = {}
6672     for j = 1, num_columns do
6673         rectangular_row[j] = row[j] or ""
6674     end
6675     table.insert(rectangular_rows, rectangular_row)
6676 end
6677 return rectangular_rows
6678 end
6679
6680 local function pipe_table_row(allow_empty_first_column
6681                             , nonempty_column
6682                             , column_separator
6683                             , column)
6684     local row_beginning
6685     if allow_empty_first_column then
6686         row_beginning = -- empty first column
6687                         #(parsers.spacechar^4
6688                         * column_separator)
6689                         * parsers.optionalspace
6690                         * column
6691                         * parsers.optionalspace
6692         -- non-empty first column
6693         + parsers.nonindentspace
6694         * nonempty_column^-1
6695         * parsers.optionalspace
6696     else
6697         row_beginning = parsers.nonindentspace
6698                         * nonempty_column^-1
6699                         * parsers.optionalspace
6700     end
6701
6702     return Ct(row_beginning
6703              * (-- single column with no leading pipes
6704                #(column_separator
6705                  * parsers.optionalspace
6706                  * parsers.newline)
6707                * column_separator
6708                * parsers.optionalspace
6709                -- single column with leading pipes or
6710                -- more than a single column
6711                + (column_separator
6712                  * parsers.optionalspace
6713                  * column
6714                  * parsers.optionalspace)^1
6715                * (column_separator
6716                  * parsers.optionalspace)^-1))

```

```

6717     end
6718
6719     return {
6720         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.

```

6721         function self.table(rows, caption)
6722             if not self.is_writing then return "" end
6723             local buffer = {"\\markdownRendererTable{",
6724                 caption or "", "}{" , #rows - 1, "}{" , #rows[1], "}" }
6725             local temp = rows[2] -- put alignments on the first row
6726             rows[2] = rows[1]
6727             rows[1] = temp
6728             for i, row in ipairs(rows) do
6729                 table.insert(buffer, "{")
6730                 for _, column in ipairs(row) do
6731                     if i > 1 then -- do not use braces for alignments
6732                         table.insert(buffer, "{")
6733                     end
6734                     table.insert(buffer, column)
6735                     if i > 1 then
6736                         table.insert(buffer, "}")
6737                     end
6738                 end
6739                 table.insert(buffer, "}")
6740             end
6741             return buffer
6742         end
6743     end, extend_reader = function(self)
6744         local parsers = self.parsers
6745         local writer = self.writer
6746
6747         local table_hline_separator = parsers.pipe + parsers.plus
6748
6749         local table_hline_column = (parsers.dash
6750             - #(parsers.dash
6751                 * (parsers.spacechar
6752                     + table_hline_separator
6753                     + parsers.newline)))^1
6754             * (parsers.colon * Cc("r")
6755                 + parsers.dash * Cc("d"))
6756             + parsers.colon
6757             * (parsers.dash
6758                 - #(parsers.dash
6759                     * (parsers.spacechar

```

```

6760             + table_hline_separator
6761             + parsers.newline)))^1
6762         * (parsers.colon * Cc("c")
6763         + parsers.dash * Cc("l"))
6764
6765     local table_hline = pipe_table_row(false
6766                                     , table_hline_column
6767                                     , table_hline_separator
6768                                     , table_hline_column)
6769
6770     local table_caption_beginning = parsers.skipblanklines
6771                                   * parsers.nonindentspace
6772                                   * (P("Table")^-1 * parsers.colon)
6773                                   * parsers.optionalspace
6774
6775     local table_row = pipe_table_row(true
6776                                     , (C((parsers.linechar - parsers.pipe)^1)
6777                                     / self.parser_functions.parse_inlines)
6778                                     , parsers.pipe
6779                                     , (C((parsers.linechar - parsers.pipe)^0)
6780                                     / self.parser_functions.parse_inlines))
6781
6782     local table_caption
6783     if table_captions then
6784         table_caption = #table_caption_beginning
6785                       * table_caption_beginning
6786                       * Ct(parsers.IndentedInline^1)
6787                       * parsers.newline
6788     else
6789         table_caption = parsers.fail
6790     end
6791
6792     local PipeTable = Ct(table_row * parsers.newline
6793                       * table_hline
6794                       * (parsers.newline * table_row)^0)
6795                       / make_pipe_table_rectangular
6796                       * table_caption^-1
6797                       / writer.table
6798
6799     self.insert_pattern("Block after Blockquote", PipeTable)
6800 end
6801 }
6802 end

```

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

```

6803 M.extensions.strike_through = function()
6804   return {
6805     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.

```

6806       function self.strike_through(s)
6807         return {"\\markdownRendererStrikeThrough{" ,s,"}"}
6808       end
6809     end, extend_reader = function(self)
6810       local parsers = self.parsers
6811       local writer = self.writer
6812
6813       local StrikeThrough = (
6814         parsers.between(parsers.Inline, parsers.doubletildes,
6815                       parsers.doubletildes)
6816       ) / writer.strike_through
6817
6818       self.insert_pattern("Inline after Emph", StrikeThrough)
6819
6820       self.add_special_character("~")
6821     end
6822   }
6823 end

```

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

```

6824 M.extensions.superscripts = function()
6825   return {
6826     extend_writer = function(self)

```

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

```

6827       function self.superscript(s)
6828         return {"\\markdownRendererSuperscript{" ,s,"}"}
6829       end
6830     end, extend_reader = function(self)
6831       local parsers = self.parsers
6832       local writer = self.writer
6833
6834       local Superscript = (
6835         parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
6836       ) / writer.superscript
6837
6838       self.insert_pattern("Inline after Emph", Superscript)
6839
6840       self.add_special_character("^")

```

```

6841     end
6842   }
6843 end

```

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

```

6844 M.extensions.subscripts = function()
6845   return {
6846     extend_writer = function(self)

```

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

```

6847     function self.subscript(s)
6848       return {"\\markdownRendererSubscript{" ,s,""}"}
6849     end
6850   end, extend_reader = function(self)
6851     local parsers = self.parsers
6852     local writer = self.writer
6853
6854     local Subscript = (
6855       parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
6856     ) / writer.subscript
6857
6858     self.insert_pattern("Inline after Emph", Subscript)
6859
6860     self.add_special_character("~")
6861   end
6862 }
6863 end

```

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

```

6864 M.extensions.fancy_lists = function()
6865   return {
6866     extend_writer = function(self)
6867       local options = self.options
6868

```

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.

```

6869     function self.fancylist(items,tight,startnum,numstyle,numdelim)
6870     if not self.is_writing then return "" end
6871     local buffer = {}
6872     local num = startnum
6873     for _,item in ipairs(items) do
6874         buffer[#buffer + 1] = self.fancyitem(item,num)
6875         if num ~= nil then
6876             num = num + 1
6877         end
6878     end
6879     local contents = util.intersperse(buffer,"\n")
6880     if tight and options.tightLists then
6881         return {"\\markdownRendererFancyOlBeginTight{",
6882             numstyle,"}{",numdelim,"}",contents,
6883             "\\n\\markdownRendererFancyOlEndTight "}
6884     else
6885         return {"\\markdownRendererFancyOlBegin{",
6886             numstyle,"}{",numdelim,"}",contents,
6887             "\\n\\markdownRendererFancyOlEnd "}
6888     end
6889 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.

```

6890     function self.fancyitem(s,num)
6891     if num ~= nil then
6892         return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
6893             "\\n\\markdownRendererFancyOlItemEnd "}
6894     else
6895         return {"\\markdownRendererFancyOlItem ",s,"\\n\\markdownRendererFancyOlItemEnd "}
6896     end
6897 end
6898 end, extend_reader = function(self)
6899     local parsers = self.parsers
6900     local options = self.options
6901     local writer = self.writer
6902

```

```

6903 local label = parsers.dig + parsers.letter
6904 local numdelim = parsers.period + parsers.rparent
6905 local enumerator = C(label^3 * numdelim) * #parsers.spacing
6906                   + C(label^2 * numdelim) * #parsers.spacing
6907                   * (parsers.tab + parsers.space^1)
6908                   + C(label * numdelim) * #parsers.spacing
6909                   * (parsers.tab + parsers.space^-2)
6910                   + parsers.space * C(label^2 * numdelim)
6911                   * #parsers.spacing
6912                   + parsers.space * C(label * numdelim)
6913                   * #parsers.spacing
6914                   * (parsers.tab + parsers.space^-1)
6915                   + parsers.space * parsers.space * C(label^1
6916                   * numdelim) * #parsers.spacing
6917 local starter = parsers.bullet + enumerator
6918
6919 local NestedList = Cs((parsers.optionallyindentedline
6920                      - starter)^1)
6921                      / function(a) return "\001"..a end
6922
6923 local ListBlockLine = parsers.optionallyindentedline
6924                      - parsers.blankline - (parsers.indent^-1
6925                      * starter)
6926
6927 local ListBlock = parsers.line * ListBlockLine^0
6928
6929 local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6930                          * ListBlock
6931
6932 local TightListItem = function(starter)
6933     return -parsers.HorizontalRule
6934           * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
1)
6935           / self.parser_functions.parse_blocks_nested)
6936           * -(parsers.blanklines * parsers.indent)
6937 end
6938
6939 local LooseListItem = function(starter)
6940     return -parsers.HorizontalRule
6941           * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
6942           * (NestedList + ListContinuationBlock^0)
6943           * (parsers.blanklines / "\n\n")
6944           ) / self.parser_functions.parse_blocks_nested
6945 end
6946
6947 local function roman2number(roman)
6948     local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }

```

```

6949     local numeral = 0
6950
6951     local i = 1
6952     local len = string.len(roman)
6953     while i < len do
6954         local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman, i+1, i+1) ]
6955         if z1 < z2 then
6956             numeral = numeral + (z2 - z1)
6957             i = i + 2
6958         else
6959             numeral = numeral + z1
6960             i = i + 1
6961         end
6962     end
6963     if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
6964     return numeral
6965 end
6966
6967 local function sniffstyle(itemprefix)
6968     local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.])")
6969     local numdelim
6970     if delimend == ")" then
6971         numdelim = "OneParen"
6972     elseif delimend == "." then
6973         numdelim = "Period"
6974     else
6975         numdelim = "Default"
6976     end
6977     numstr = numstr or itemprefix
6978
6979     local num
6980     num = numstr:match("^([IVXL]+)")
6981     if num then
6982         return roman2number(num), "UpperRoman", numdelim
6983     end
6984     num = numstr:match("^([ivxl]+)")
6985     if num then
6986         return roman2number(string.upper(num)), "LowerRoman", numdelim
6987     end
6988     num = numstr:match("^([A-Z])")
6989     if num then
6990         return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
6991     end
6992     num = numstr:match("^([a-z])")
6993     if num then
6994         return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
6995     end

```

```

6996         return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
6997     end
6998
6999     local function fancylist(items,tight,start)
7000         local startnum, numstyle, numdelim = sniffstyle(start)
7001         return writer.fancylist(items,tight,
7002                                 options.startNumber and startnum,
7003                                 numstyle or "Decimal",
7004                                 numdelim or "Default")
7005     end
7006
7007     local FancyList = Cg(enumerator, "listtype") *
7008         ( Ct(TightListItem(Cb("listtype")))
7009           * TightListItem(enumerator)^0)
7010       * Cc(true) * parsers.skipblanklines * -enumerator
7011       + Ct(LooseListItem(Cb("listtype")))
7012         * LooseListItem(enumerator)^0)
7013       * Cc(false) * parsers.skipblanklines
7014       ) * Cb("listtype") / fancylist
7015
7016     self.update_rule("OrderedList", FancyList)
7017 end
7018 }
7019 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 ??.

```

7020 function M.new(options)

```

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

```

7021     options = options or {}
7022     setmetatable(options, { __index = function (_, key)
7023         return defaultOptions[key] end })

```

Apply built-in syntax extensions based on `options`.

```

7024     local extensions = {}
7025
7026     if options.contentBlocks then
7027         local content_blocks_extension = M.extensions.content_blocks(
7028             options.contentBlocksLanguageMap)
7029         table.insert(extensions, content_blocks_extension)
7030     end
7031
7032     if options.definitionLists then
7033         local definition_lists_extension = M.extensions.definition_lists(

```

```

7034     options.tightLists)
7035     table.insert(extensions, definition_lists_extension)
7036 end
7037
7038 if options.fencedCode then
7039     local fenced_code_extension = M.extensions.fenced_code(
7040         options.blankBeforeCodeFence)
7041     table.insert(extensions, fenced_code_extension)
7042 end
7043
7044 if options.headerAttributes then
7045     local header_attributes_extension = M.extensions.header_attributes()
7046     table.insert(extensions, header_attributes_extension)
7047 end
7048
7049 if options.jekyllData then
7050     local jekyll_data_extension = M.extensions.jekyll_data(
7051         options.expectJekyllData)
7052     table.insert(extensions, jekyll_data_extension)
7053 end
7054
7055 if options.pipeTables then
7056     local pipe_tables_extension = M.extensions.pipe_tables(
7057         options.tableCaptions)
7058     table.insert(extensions, pipe_tables_extension)
7059 end
7060
7061 if options.strikeThrough then
7062     local strike_through_extension = M.extensions.strike_through()
7063     table.insert(extensions, strike_through_extension)
7064 end
7065
7066 if options.subscripts then
7067     local subscript_extension = M.extensions.subscripts()
7068     table.insert(extensions, subscript_extension)
7069 end
7070
7071 if options.superscripts then
7072     local superscript_extension = M.extensions.superscripts()
7073     table.insert(extensions, superscript_extension)
7074 end
7075
7076 if options.footnotes or options.inlineFootnotes then
7077     local footnotes_extension = M.extensions.footnotes(
7078         options.footnotes, options.inlineFootnotes)
7079     table.insert(extensions, footnotes_extension)
7080 end

```

```

7081
7082 if options.citations then
7083     local citations_extension = M.extensions.citations(options.citationNbsps)
7084     table.insert(extensions, citations_extension)
7085 end
7086
7087 if options.fancyLists then
7088     local fancy_lists_extension = M.extensions.fancy_lists()
7089     table.insert(extensions, fancy_lists_extension)
7090 end

```

Apply user-defined syntax extensions based on `options.extensions`.

```

7091 for _, user_extension_filename in ipairs(options.extensions) do
7092     local user_extension = (function(filename)

```

First, load and compile the contents of the user-defined syntax extension.

```

7093         local pathname = util.lookup_files(filename)
7094         local input_file = assert(io.open(pathname, "r"),
7095             [[Could not open user-defined syntax extension "]]
7096             .. pathname .. [[ for reading]])
7097         local input = assert(input_file:read("*a"))
7098         assert(input_file:close())
7099         local user_extension, err = load([[
7100             local sandbox = {}
7101             setmetatable(sandbox, {__index = _G})
7102             _ENV = sandbox
7103         ]] .. input)()
7104         assert(user_extension,
7105             [[Failed to compile user-defined syntax extension "]]
7106             .. pathname .. [[ ": ]] .. (err or [[]]))

```

Then, validate the user-defined syntax extension.

```

7107         assert(user_extension.api_version ~= nil,
7108             [[User-defined syntax extension "]] .. pathname
7109             .. [[ does not specify mandatory field "api_version"]])
7110         assert(type(user_extension.api_version) == "number",
7111             [[User-defined syntax extension "]] .. pathname
7112             .. [[ specifies field "api_version" of type "]]
7113             .. type(user_extension.api_version)
7114             .. [[ but "number" was expected]])
7115         assert(user_extension.api_version == metadata.user_extension_api_version,
7116             [[User-defined syntax extension "]] .. pathname
7117             .. [[ uses syntax extension API version "]]
7118             .. user_extension.api_version .. [[ but markdown.lua ]]
7119             .. metadata.version .. [[ uses API version ]]
7120             .. metadata.user_extension_api_version
7121             .. [[, which is incompatible]])
7122

```

```

7123     assert(user_extension.grammar_version ~= nil,
7124            [[User-defined syntax extension "]] .. pathname
7125            .. [" does not specify mandatory field "grammar_version"]])
7126     assert(type(user_extension.grammar_version) == "number",
7127            [[User-defined syntax extension "]] .. pathname
7128            .. [" specifies field "grammar_version" of type "]]
7129            .. type(user_extension.grammar_version)
7130            .. [" but "number" was expected]])
7131     assert(user_extension.grammar_version == metadata.grammar_version,
7132            [[User-defined syntax extension "]] .. pathname
7133            .. [" uses grammar version "]] .. user_extension.grammar_version
7134            .. [" but markdown.lua ]] .. metadata.version
7135            .. [" uses grammar version ]] .. metadata.grammar_version
7136            .. [[, which is incompatible]])
7137
7138     assert(user_extension.finalize_grammar ~= nil,
7139            [[User-defined syntax extension "]] .. pathname
7140            .. [" does not specify mandatory "finalize_grammar" field]])
7141     assert(type(user_extension.finalize_grammar) == "function",
7142            [[User-defined syntax extension "]] .. pathname
7143            .. [" specifies field "finalize_grammar" of type "]]
7144            .. type(user_extension.finalize_grammar)
7145            .. [" 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 ??.)

```

7146     local extension = {
7147         extend_reader = user_extension.finalize_grammar,
7148         extend_writer = function() end,
7149     }
7150     return extension
7151 end)(user_extension_filename)
7152 table.insert(extensions, user_extension)
7153 end

```

Produce and return a conversion function from markdown to plain TeX.

```

7154     local writer = M.writer.new(options)
7155     local reader = M.reader.new(writer, options)
7156     local convert = reader.finalize_grammar(extensions)
7157
7158     return convert
7159 end
7160
7161 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.

```
7162
7163 local input
7164 if input_filename then
7165     local input_file = assert(io.open(input_filename, "r"),
7166         [[Could not open file ]] .. input_filename .. [[ for reading]])
7167     input = assert(input_file:read("*a"))
7168     assert(input_file:close())
7169 else
7170     input = assert(io.read("*a"))
7171 end
7172
```

First, ensure that the `options.cacheDir` directory exists.

```
7173 local lfs = require("lfs")
7174 if options.cacheDir and not lfs.isdir(options.cacheDir) then
7175     assert(lfs.mkdir(options["cacheDir"]))
7176 end
7177
7178 local ran_ok, kpse = pcall(require, "kpse")
7179 if ran_ok then kpse.set_program_name("luatex") end
7180 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.

```
7181 if metadata.version ~= md.metadata.version then
7182     warn("markdown-cli.lua " .. metadata.version .. " used with " ..
7183         "markdown.lua " .. md.metadata.version .. ".")
7184 end
7185 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.

```
7186 local output = convert(input:gsub("\r\n?", "\n") .. "\n")
7187
7188 if output_filename then
7189     local output_file = assert(io.open(output_filename, "w"),
7190         [[Could not open file ]] .. output_filename .. [[ for writing]])
7191     assert(output_file:write(output))
7192     assert(output_file:close())
7193 else
7194     assert(io.write(output))
7195 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

```
7196 \ifx\markdownInfo\undefined
7197   \def\markdownInfo#1{%
7198     \immediate\write-1{(1.\the\inputlineno) markdown.tex info: #1.}}%
7199 \fi
7200 \ifx\markdownWarning\undefined
7201   \def\markdownWarning#1{%
7202     \immediate\write16{(1.\the\inputlineno) markdown.tex warning: #1}}%
7203 \fi
7204 \ifx\markdownError\undefined
7205   \def\markdownError#1#2{%
7206     \errhelp{#2.}}%
7207     \errmessage{(1.\the\inputlineno) markdown.tex error: #1}}%
7208 \fi
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
7209 \def\markdownRendererInterblockSeparatorPrototype{\par}%
7210 \def\markdownRendererLineBreakPrototype{\hfil\break}%
7211 \let\markdownRendererEllipsisPrototype\dots
7212 \def\markdownRendererNbspPrototype{~}%
7213 \def\markdownRendererLeftBracePrototype{\char`\{}%
7214 \def\markdownRendererRightBracePrototype{\char`\}%
7215 \def\markdownRendererDollarSignPrototype{\char`$}%
7216 \def\markdownRendererPercentSignPrototype{\char`\}%
7217 \def\markdownRendererAmpersandPrototype{\&}%
7218 \def\markdownRendererUnderscorePrototype{\char`\_%}
7219 \def\markdownRendererHashPrototype{\char`\#}%
7220 \def\markdownRendererCircumflexPrototype{\char`\^}%
7221 \def\markdownRendererBackslashPrototype{\char`\}%
7222 \def\markdownRendererTildePrototype{\char`\~}%
7223 \def\markdownRendererPipePrototype{|}%
7224 \def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
7225 \def\markdownRendererLinkPrototype#1#2#3#4{#2}%
7226 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
7227   \markdownInput{#3}}%
7228 \def\markdownRendererContentBlockOnlineImagePrototype{%
7229   \markdownRendererImage}%
```

```

7230 \def\markdownRenderContentBlockCodePrototype#1#2#3#4#5{%
7231   \markdownRenderInputFencedCode{#3}{#2}}%
7232 \def\markdownRenderImagePrototype#1#2#3#4{#2}%
7233 \def\markdownRenderU1BeginPrototype{}%
7234 \def\markdownRenderU1BeginTightPrototype{}%
7235 \def\markdownRenderU1ItemPrototype{}%
7236 \def\markdownRenderU1ItemEndPrototype{}%
7237 \def\markdownRenderU1EndPrototype{}%
7238 \def\markdownRenderU1EndTightPrototype{}%
7239 \def\markdownRenderO1BeginPrototype{}%
7240 \def\markdownRenderO1BeginTightPrototype{}%
7241 \def\markdownRenderFancyO1BeginPrototype#1#2{\markdownRenderO1Begin}%
7242 \def\markdownRenderFancyO1BeginTightPrototype#1#2{\markdownRenderO1BeginTight}%
7243 \def\markdownRenderO1ItemPrototype{}%
7244 \def\markdownRenderO1ItemWithNumberPrototype#1{}%
7245 \def\markdownRenderO1ItemEndPrototype{}%
7246 \def\markdownRenderFancyO1ItemPrototype{\markdownRenderO1Item}%
7247 \def\markdownRenderFancyO1ItemWithNumberPrototype{\markdownRenderO1ItemWithNumber}%
7248 \def\markdownRenderFancyO1ItemEndPrototype{}%
7249 \def\markdownRenderO1EndPrototype{}%
7250 \def\markdownRenderO1EndTightPrototype{}%
7251 \def\markdownRenderFancyO1EndPrototype{\markdownRenderO1End}%
7252 \def\markdownRenderFancyO1EndTightPrototype{\markdownRenderO1EndTight}%
7253 \def\markdownRenderD1BeginPrototype{}%
7254 \def\markdownRenderD1BeginTightPrototype{}%
7255 \def\markdownRenderD1ItemPrototype#1{#1}%
7256 \def\markdownRenderD1ItemEndPrototype{}%
7257 \def\markdownRenderD1DefinitionBeginPrototype{}%
7258 \def\markdownRenderD1DefinitionEndPrototype{\par}%
7259 \def\markdownRenderD1EndPrototype{}%
7260 \def\markdownRenderD1EndTightPrototype{}%
7261 \def\markdownRenderEmphasisPrototype#1{{\it#1}}%
7262 \def\markdownRenderStrongEmphasisPrototype#1{{\bf#1}}%
7263 \def\markdownRenderBlockQuoteBeginPrototype{\par\begingroup\it}%
7264 \def\markdownRenderBlockQuoteEndPrototype{\endgroup\par}%
7265 \def\markdownRenderInputVerbatimPrototype#1{%
7266   \par{\tt\input#1\relax{}}\par}%
7267 \def\markdownRenderInputFencedCodePrototype#1#2{%
7268   \markdownRenderInputVerbatimPrototype{#1}}%
7269 \def\markdownRenderHeadingOnePrototype#1{#1}%
7270 \def\markdownRenderHeadingTwoPrototype#1{#1}%
7271 \def\markdownRenderHeadingThreePrototype#1{#1}%
7272 \def\markdownRenderHeadingFourPrototype#1{#1}%
7273 \def\markdownRenderHeadingFivePrototype#1{#1}%
7274 \def\markdownRenderHeadingSixPrototype#1{#1}%
7275 \def\markdownRenderHorizontalRulePrototype{}%
7276 \def\markdownRenderFootnotePrototype#1{#1}%

```

```

7277 \def\markdownRendererCitePrototype#1{%
7278 \def\markdownRendererTextCitePrototype#1{%
7279 \def\markdownRendererTickedBoxPrototype{[X]}%
7280 \def\markdownRendererHalfTickedBoxPrototype{[/]}%
7281 \def\markdownRendererUntickedBoxPrototype{[ ]}%
7282 \def\markdownRendererStrikeThroughPrototype#1{#1}%
7283 \def\markdownRendererSuperscriptPrototype#1{#1}%
7284 \def\markdownRendererSubscriptPrototype#1{#1}%

```

3.2.2.1 YAML Metadata Renderer Prototypes To keep track of the current type of structure we inhabit when we are traversing a YAML document, we will maintain the `\g_@@_jekyll_data_datatypes_seq` stack. At every step of the traversal, the stack will contain one of the following constants at any position p :

`\c_@@_jekyll_data_sequence_tl` The currently traversed branch of the YAML document contains a sequence at depth p .

`\c_@@_jekyll_data_mapping_tl` The currently traversed branch of the YAML document contains a mapping at depth p .

`\c_@@_jekyll_data_scalar_tl` The currently traversed branch of the YAML document contains a scalar value at depth p .

```

7285 \ExplSyntaxOn
7286 \seq_new:N \g_@@_jekyll_data_datatypes_seq
7287 \tl_const:Nn \c_@@_jekyll_data_sequence_tl { sequence }
7288 \tl_const:Nn \c_@@_jekyll_data_mapping_tl { mapping }
7289 \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.

```

7290 \seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
7291 \cs_new:Nn \markdown_jekyll_data_push_address_segment:n
7292 {
7293   \seq_if_empty:NF
7294     \g_@@_jekyll_data_datatypes_seq
7295     {
7296       \seq_get_right:NN
7297       \g_@@_jekyll_data_datatypes_seq
7298       \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.

```

7299     \str_if_eq:NNTF
7300     \l_tmpa_tl
7301     \c_@@_jekyll_data_sequence_tl
7302     {
7303         \seq_put_right:Nn
7304         \g_@@_jekyll_data_wildcard_absolute_address_seq
7305         { * }
7306     }
7307     {
7308         \seq_put_right:Nn
7309         \g_@@_jekyll_data_wildcard_absolute_address_seq
7310         { #1 }
7311     }
7312 }
7313 }

```

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.

```

7314 \tl_new:N \g_@@_jekyll_data_wildcard_absolute_address_tl
7315 \tl_new:N \g_@@_jekyll_data_wildcard_relative_address_tl
7316 \cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
7317 {

```

```

7318     \seq_pop_left:NN #1 \l_tmpa_tl
7319     \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
7320     \seq_put_left:NV #1 \l_tmpa_tl
7321   }
7322   \cs_new:Nn \markdown_jekyll_data_update_address_tls:
7323   {
7324     \markdown_jekyll_data_concatenate_address:NN
7325     \g_@@_jekyll_data_wildcard_absolute_address_seq
7326     \g_@@_jekyll_data_wildcard_absolute_address_tl
7327     \seq_get_right:NN
7328     \g_@@_jekyll_data_wildcard_absolute_address_seq
7329     \g_@@_jekyll_data_wildcard_relative_address_tl
7330   }

```

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.

```

7331   \cs_new:Nn \markdown_jekyll_data_push:nN
7332   {
7333     \markdown_jekyll_data_push_address_segment:n
7334     { #1 }
7335     \seq_put_right:NV
7336     \g_@@_jekyll_data_datatypes_seq
7337     #2
7338     \markdown_jekyll_data_update_address_tls:
7339   }
7340   \cs_new:Nn \markdown_jekyll_data_pop:
7341   {
7342     \seq_pop_right:NN
7343     \g_@@_jekyll_data_wildcard_absolute_address_seq
7344     \l_tmpa_tl
7345     \seq_pop_right:NN
7346     \g_@@_jekyll_data_datatypes_seq
7347     \l_tmpa_tl
7348     \markdown_jekyll_data_update_address_tls:
7349   }

```

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.

```

7350   \cs_new:Nn \markdown_jekyll_data_set_keyval:nn
7351   {
7352     \keys_set_known:nn
7353     { markdown/jekyllData }
7354     { { #1 } = { #2 } }
7355   }
7356   \cs_generate_variant:Nn
7357   \markdown_jekyll_data_set_keyval:nn
7358   { Vn }

```

```

7359 \cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
7360 {
7361     \markdown_jekyll_data_push:nN
7362     { #1 }
7363     \c_@@_jekyll_data_scalar_tl
7364     \markdown_jekyll_data_set_keyval:Vn
7365     \g_@@_jekyll_data_wildcard_absolute_address_tl
7366     { #2 }
7367     \markdown_jekyll_data_set_keyval:Vn
7368     \g_@@_jekyll_data_wildcard_relative_address_tl
7369     { #2 }
7370     \markdown_jekyll_data_pop:
7371 }

```

Finally, we will register our macros as token renderer prototypes to be able to react to the traversal of a YAML document.

```

7372 \def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
7373     \markdown_jekyll_data_push:nN
7374     { #1 }
7375     \c_@@_jekyll_data_sequence_tl
7376 }
7377 \def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
7378     \markdown_jekyll_data_push:nN
7379     { #1 }
7380     \c_@@_jekyll_data_mapping_tl
7381 }
7382 \def\markdownRendererJekyllDataSequenceEndPrototype{
7383     \markdown_jekyll_data_pop:
7384 }
7385 \def\markdownRendererJekyllDataMappingEndPrototype{
7386     \markdown_jekyll_data_pop:
7387 }
7388 \def\markdownRendererJekyllDataBooleanPrototype#1#2{
7389     \markdown_jekyll_data_set_keyvals:nn
7390     { #1 }
7391     { #2 }
7392 }
7393 \def\markdownRendererJekyllDataEmptyPrototype#1{}
7394 \def\markdownRendererJekyllDataNumberPrototype#1#2{
7395     \markdown_jekyll_data_set_keyvals:nn
7396     { #1 }
7397     { #2 }
7398 }
7399 \def\markdownRendererJekyllDataStringPrototype#1#2{
7400     \markdown_jekyll_data_set_keyvals:nn
7401     { #1 }
7402     { #2 }

```

```

7403 }
7404 \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).

```

7405 \ExplSyntaxOn
7406 \tl_new:N \g_@@_formatted_lua_options_tl
7407 \cs_new:Nn \@@_format_lua_options:
7408 {
7409   \tl_gclear:N
7410   \g_@@_formatted_lua_options_tl
7411   \seq_map_function:NN
7412     \g_@@_lua_options_seq
7413     \@@_format_lua_option:n
7414 }
7415 \cs_new:Nn \@@_format_lua_option:n
7416 {
7417   \@@_typecheck_option:n
7418     { #1 }
7419   \@@_get_option_type:nN
7420     { #1 }
7421   \l_tmpa_tl
7422   \bool_case_true:nF
7423     {
7424       {
7425         \str_if_eq_p:VV
7426           \l_tmpa_tl
7427           \c_@@_option_type_boolean_tl ||
7428         \str_if_eq_p:VV
7429           \l_tmpa_tl
7430           \c_@@_option_type_number_tl ||
7431         \str_if_eq_p:VV
7432           \l_tmpa_tl
7433           \c_@@_option_type_counter_tl
7434       }
7435       {
7436         \@@_get_option_value:nN
7437           { #1 }
7438         \l_tmpa_tl
7439         \tl_gput_right:Nx
7440           \g_@@_formatted_lua_options_tl
7441           { #1~== \l_tmpa_tl ,~ }
7442       }

```

```

7443     {
7444         \str_if_eq_p:VV
7445         \l_tmpa_tl
7446         \c_@@_option_type_clist_tl
7447     }
7448     {
7449         \@@_get_option_value:nN
7450         { #1 }
7451         \l_tmpa_tl
7452         \tl_gput_right:Nx
7453         \g_@@_formatted_lua_options_tl
7454         { #1~::~\c_left_brace_str }
7455         \clist_map_inline:Nn
7456         \l_tmpa_tl
7457         {
7458             \tl_gput_right:Nx
7459             \g_@@_formatted_lua_options_tl
7460             { "##1" ,~ }
7461         }
7462         \tl_gput_right:Nx
7463         \g_@@_formatted_lua_options_tl
7464         { \c_right_brace_str ,~ }
7465     }
7466 }
7467 {
7468     \@@_get_option_value:nN
7469     { #1 }
7470     \l_tmpa_tl
7471     \tl_gput_right:Nx
7472     \g_@@_formatted_lua_options_tl
7473     { #1~::~ " \l_tmpa_tl " ,~ }
7474 }
7475 }
7476 \cs_generate_variant:Nn
7477 \clist_map_inline:nn
7478 { Vn }
7479 \let\markdownPrepareLuaOptions=\@@_format_lua_options:
7480 \def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
7481 \ExplSyntaxOff

```

The `\markdownPrepare` macro contains the Lua code that is executed prior to any conversion from markdown to plain \TeX . It exposes the `convert` function for the use by any further Lua code.

```

7482 \def\markdownPrepare{%

```

First, ensure that the `\markdownOptionCacheDir` directory exists.

```

7483     local lfs = require("lfs")
7484     local cacheDir = "\markdownOptionCacheDir"

```



```

7485 if not lfs.isdir(cacheDir) then
7486     assert(lfs.mkdir(cacheDir))
7487 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.

```

7488 local md = require("markdown")
7489 local convert = md.new(\markdownLuaOptions)
7490 }%

```

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.

```

7491 \ExplSyntaxOn
7492 \cs_new:Nn
7493   \@@_if_option:nTF
7494   {
7495     \@@_get_option_type:nN
7496     { #1 }
7497     \l_tmpa_tl
7498     \str_if_eq:NNF
7499     \l_tmpa_tl
7500     \c_@@_option_type_boolean_tl
7501     {
7502       \msg_error:nnxx
7503       { @@ }
7504       { expected-boolean-option }
7505       { #1 }
7506       { \l_tmpa_tl }
7507     }
7508     \@@_get_option_value:nN
7509     { #1 }
7510     \l_tmpa_tl
7511     \str_if_eq:NNTF
7512     \l_tmpa_tl
7513     \c_@@_option_value_true_tl
7514     { #2 }
7515     { #3 }
7516   }
7517 \msg_new:nnn
7518   { @@ }
7519   { expected-boolean-option }
7520   {
7521     Option~#1~has~type~#2,~
7522     but~a~boolean~was~expected.

```

```

7523 }
7524 \let\markdownIfOption=\@@_if_option:nTF
7525 \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.

```

7526 \csname newread\endcsname\markdownInputFileStream
7527 \csname newwrite\endcsname\markdownOutputFileStream

```

The `\markdownReadAndConvertTab` macro contains the tab character literal.

```

7528 \begingroup
7529   \catcode`\^^I=12%
7530   \gdef\markdownReadAndConvertTab{^^I}%
7531 \endgroup

```

The `\markdownReadAndConvert` macro is largely a rewrite of the $\text{\LaTeX 2}\epsilon$ `\filecontents` macro to plain \TeX .

```

7532 \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 `\markdownOptionStripPercentSigns` is enabled.

```

7533   \catcode`\^^M=13%
7534   \catcode`\^^I=13%
7535   \catcode`\|=0%
7536   \catcode`\|=12%
7537   |catcode`@=14%
7538   |catcode`|=12@
7539   |gdef|markdownReadAndConvert#1#2{@
7540     |begingroup@

```

If we are not reading markdown documents from the frozen cache, open the `\markdownOptionInputTempFileName` file for writing.

```

7541     |markdownIfOption{frozenCache}{@}{@
7542       |immediate|openout|markdownOutputFileStream@
7543       |markdownOptionInputTempFileName|relax@
7544       |markdownInfo{Buffering markdown input into the temporary @
7545         input file "|markdownOptionInputTempFileName" and scanning @
7546         for the closing token sequence "#1"}@
7547     }@

```

Locally change the category of the special plain \TeX characters to *other* in order to prevent unwanted interpretation of the input. Change also the category of the space character, so that we can retrieve it unaltered.

```

7548     |def|do##1{|catcode`##1=12}|dospecials@

```

```

7549 |catcode`| =12@
7550 |markdownMakeOther@

```

The `\markdownReadAndConvertStripPercentSigns` macro will process the individual lines of output, stripping away leading percent signs (%) when `\markdownOptionStripPercentSigns` is enabled. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (\sim) are produced.

```

7551 |def|markdownReadAndConvertStripPercentSign##1{@
7552 |markdownIfOption{stripPercentSigns}{@
7553 |if##1%@
7554 |expandafter|expandafter|expandafter@
7555 |markdownReadAndConvertProcessLine@
7556 |else@
7557 |expandafter|expandafter|expandafter@
7558 |markdownReadAndConvertProcessLine@
7559 |expandafter|expandafter|expandafter##1@
7560 |fi@
7561 }{@
7562 |expandafter@
7563 |markdownReadAndConvertProcessLine@
7564 |expandafter##1@
7565 }@
7566 }@

```

The `\markdownReadAndConvertProcessLine` macro will process the individual lines of output. Notice the use of the comments (@) to ensure that the entire macro is at a single line and therefore no (active) newline symbols (\sim) are produced.

```

7567 |def|markdownReadAndConvertProcessLine##1##2##3|relax{@

```

If we are not reading markdown documents from the frozen cache and the ending token sequence does not appear in the line, store the line in the `\markdownOptionInputTempFileName` 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.

```

7568 |ifx|relax##3|relax@
7569 |markdownIfOption{frozenCache}{@
7570 |immediate|write|markdownOutputFileStream{##1}@
7571 }@
7572 |else@

```

When the ending token sequence appears in the line, make the next newline character close the `\markdownOptionInputTempFileName` file, return the character categories back to the former state, convert the `\markdownOptionInputTempFileName` file from markdown to plain T_EX, `\input` the result of the conversion, and expand the ending control sequence.

```

7573 |def $\sim$ M{@

```

```

7574         |markdownInfo{The ending token sequence was found}@
7575         |markdownIfOption{frozenCache}{-}{@
7576             |immediate|closeout|markdownOutputFileStream@
7577         }@
7578         |endgroup@
7579         |markdownInput{@
7580             |markdownOptionOutputDir@
7581             /|markdownOptionInputTempFileName@
7582         }@
7583         #2}@
7584         |fi@

```

Repeat with the next line.

```

7585         ^^M}@

```

Make the tab character active at expansion time and make it expand to a literal tab character.

```

7586         |catcode`|^I=13@
7587         |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.

```

7588         |catcode`|^M=13@
7589         |def^^M##1^^M{@
7590             |def^^M####1^^M{@
7591                 |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
7592             ^^M}@
7593         ^^M}@

```

Reset the character categories back to the former state.

```

7594 |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.

```

7595 \ExplSyntaxOn
7596 \int_compare:nT
7597 { \markdownMode = 3 }
7598 {
7599     \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
7600     \file_input:n { lt3luabridge.tex }
7601     \cs_new:Npn
7602         \markdownLuaExecute
7603         { \luabridgeExecute }
7604     }
7605 \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].

```
7606 \ifnum\markdownMode<2\relax
7607 \ifnum\markdownMode=0\relax
7608   \markdownWarning{Using mode 0: Shell escape via write18
7609                   (deprecated, to be removed in Markdown 3.0.0)}%
7610 \else
7611   \markdownWarning{Using mode 1: Shell escape via os.execute
7612                   (deprecated, to be removed in Markdown 3.0.0)}%
7613 \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.

```
7614 \ifx\pdfshellescape\undefined
7615   \ifx\shellescape\undefined
7616     \ifnum\markdownMode=0\relax
7617       \def\markdownExecuteShellEscape{1}%
7618     \else
7619       \def\markdownExecuteShellEscape{%
7620         \directlua{tex.sprint(status.shell_escape or "1")}}%
7621     \fi
7622   \else
7623     \let\markdownExecuteShellEscape\shellescape
7624   \fi
7625 \else
7626   \let\markdownExecuteShellEscape\pdfshellescape
7627 \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.

```
7628 \ifnum\markdownMode=0\relax
7629   \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
7630 \else
```

```

7631 \def\markdownExecuteDirect#1{%
7632     \directlua{os.execute("\luaescapestring{#1}")}}%
7633 \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.

```

7634 \def\markdownExecute#1{%
7635     \ifnum\markdownExecuteShellEscape=1\relax
7636         \markdownExecuteDirect{#1}%
7637     \else
7638         \markdownError{I can not access the shell}{Either run the TeX
7639             compiler with the --shell-escape or the --enable-write18 flag,
7640             or set shell_escape=t in the texmf.cnf file}%
7641     \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.

```

7642 \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.

```

7643 \catcode`\|=0%
7644 \catcode`\|=12%
7645 \gdef\markdownLuaExecute#1{%

```

Create the file `\markdownOptionHelperScriptFileName` and fill it with the input Lua code prepended with `kpathsea` initialization, so that Lua modules from the \TeX distribution are available.

```

7646     |immediate|openout|markdownOutputFileStream=%
7647     |markdownOptionHelperScriptFileName
7648     |markdownInfo{Writing a helper Lua script to the file
7649     "|markdownOptionHelperScriptFileName"}%
7650     |immediate|write|markdownOutputFileStream{%
7651         local ran_ok, error = pcall(function()
7652             local ran_ok, kpse = pcall(require, "kpse")
7653             if ran_ok then kpse.set_program_name("luatex") end
7654             #1
7655         end)

```

If there was an error, use the file `\markdownOptionErrorTempFileName` to store the error message.

```

7656         if not ran_ok then
7657             local file = io.open("%
7658                 |markdownOptionOutputDir
7659                 /|markdownOptionErrorTempFileName", "w")
7660             if file then

```

```

7661         file:write(error .. "\n")
7662         file:close()
7663     end
7664     print('\markdownError{An error was encountered while executing
7665         Lua code}{For further clues, examine the file
7666         "|markdownOptionOutputDir
7667         /|markdownOptionErrorTempFileName"}')
7668     end}%
7669     |immediate|closeout|markdownOutputFileStream

```

Execute the generated `\markdownOptionHelperScriptFileName` Lua script using the `TEX`Lua binary and store the output in the `\markdownOptionOutputTempFileName` file.

```

7670     |markdownInfo{Executing a helper Lua script from the file
7671         "|markdownOptionHelperScriptFileName" and storing the result in the
7672         file "|markdownOptionOutputTempFileName"}%
7673     |markdownExecute{texlua "|markdownOptionOutputDir
7674         /|markdownOptionHelperScriptFileName" > %
7675         "|markdownOptionOutputDir
7676         /|markdownOptionOutputTempFileName"}%

```

`\input` the generated `\markdownOptionOutputTempFileName` file.

```

7677     |input|markdownOptionOutputTempFileName|relax}%
7678 |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.

```

7679 \fi
7680 \ifnum\markdownMode=2\relax
7681     \markdownWarning{Using mode 2: Direct Lua access
7682         (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,

```

7683 \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.

```

7684     \catcode`\=0%
7685     \catcode`\=12%
7686     |gdef|markdownLuaExecute#1{%

```

```

7687 |directlua{%
7688   local function print(input)
7689     local output = {}
7690     for line in input:gmatch("[^\r\n]+") do
7691       table.insert(output, line)
7692     end
7693     tex.print(output)
7694   end
7695   #1
7696 }%
7697 }%
7698 |endgroup
7699 \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 TeX.

```

7700 \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.

```

7701 \catcode`\|=0%
7702 \catcode`\|=12%
7703 \catcode`\&=6%
7704 |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.

```

7705 |begingroup
7706 |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.

```

7707 |catcode`\#=12

```

If we are reading from the frozen cache, input it, expand the corresponding `\markdownFrozenCache`*<number>* macro, and increment `frozenCacheCounter`.

```

7708 |markdownIfOption{frozenCache}{%
7709   |ifnum|markdownOptionFrozenCacheCounter=0|relax
7710   |markdownInfo{Reading frozen cache from
7711     "|markdownOptionFrozenCacheFileName"}%
7712   |input|markdownOptionFrozenCacheFileName|relax

```



```

7713     |fi
7714     |markdownInfo{Including markdown document number
7715         "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
7716     |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
7717     |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7718 }{%
7719     |markdownInfo{Including markdown document "&1"}%

```

Attempt to open the markdown document to record it in the `.log` and `.fls` files. This allows external programs such as \LaTeX Mk to track changes to the markdown document.

```

7720     |openin|markdownInputFileStream&1
7721     |closein|markdownInputFileStream
7722     |markdownPrepareLuaOptions
7723     |markdownLuaExecute{%
7724         |markdownPrepare
7725         local file = assert(io.open("&1", "r"),
7726             [[Could not open file "&1" for reading]])
7727         local input = assert(file:read("*a"))
7728         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.

```

7729         print(convert(input:gsub("\r\n?", "\n") .. "\n"))}%

```

In case we were finalizing the frozen cache, increment `frozenCacheCounter`.

```

7730     |global|advance|markdownOptionFrozenCacheCounter by 1|relax
7731 }%
7732 |endgroup
7733 }%
7734 |endgroup

```

3.3 \LaTeX Implementation

The \LaTeX implementation makes use of the fact that, apart from some subtle differences, \LaTeX implements the majority of the plain \TeX format [10, Section 9]. As a consequence, we can directly reuse the existing plain \TeX implementation.

```

7735 \def\markdownVersionSpace{ }%
7736 \ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
7737 \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.

```

7738 \ExplSyntaxOn
7739 \@@_latex_define_renderers:
7740 \@@_latex_define_renderer_prototypes:
7741 \ExplSyntaxOff

```

3.3.1 Logging Facilities

The \LaTeX implementation redefines the plain \TeX logging macros (see Section 3.2.1) to use the \LaTeX `\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 \LaTeX interface (see Section 2.3.2).

```
7742 \let\markdownInputPlainTeX\markdownInput
7743 \renewcommand\markdownInput[2][]{%
7744   \begingroup
7745     \markdownSetup{#1}%
7746     \markdownInputPlainTeX{#2}%
7747   \endgroup}%
```

The `markdown`, and `markdown*` \LaTeX environments are implemented using the `\markdownReadAndConvert` macro.

```
7748 \renewenvironment{markdown}{%
7749   \markdownReadAndConvert@markdown{}}{%
7750   \markdownEnd}%
7751 \renewenvironment{markdown*}[1]{%
7752   \markdownSetup{#1}%
7753   \markdownReadAndConvert@markdown*}{%
7754   \markdownEnd}%
7755 \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*.

```
7756 \catcode`\|=0\catcode`\<=1\catcode`\>=2%
7757 \catcode`\|=12\catcode`\{=12\catcode`\}=12%
7758 |gdef|markdownReadAndConvert@markdown#1<%
7759   |markdownReadAndConvert<\end{markdown#1}>%
7760                               <|end<markdown#1>>>%
7761 |endgroup
```

3.3.2.1 \LaTeX Themes This section implements the theme-loading mechanism and the example themes provided with the Markdown package.

```
7762 \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.

```

7763 \newcommand\markdownLaTeXThemeName{}
7764 \seq_new:N \g_@@_latex_themes_seq
7765 \seq_put_right:NV
7766   \g_@@_latex_themes_seq
7767   \markdownLaTeXThemeName
7768 \newcommand\markdownLaTeXThemeLoad[2]{
7769   \def\@tempa{%
7770     \def\markdownLaTeXThemeName{#2}
7771     \seq_put_right:NV
7772       \g_@@_latex_themes_seq
7773       \markdownLaTeXThemeName
7774     \RequirePackage{#1}
7775     \seq_pop_right:NN
7776       \g_@@_latex_themes_seq
7777       \l_tmpa_tl
7778     \seq_get_right:NN
7779       \g_@@_latex_themes_seq
7780       \l_tmpa_tl
7781     \exp_args:NNV
7782       \def
7783         \markdownLaTeXThemeName
7784         \l_tmpa_tl}
7785   \ifmarkdownLaTeXLoaded
7786     \@tempa
7787   \else
7788     \exp_args:No
7789       \AtEndOfPackage
7790     { \@tempa }
7791   \fi}
7792 \ExplSyntaxOff

```

The `witiko/dot` theme enables the `fencedCode` Lua option:

```

7793 \markdownSetup{fencedCode}%

```

We load the `ifthen` and `grffile` packages, see also Section 1.1.3:

```

7794 \RequirePackage{ifthen,grffile}

```

We store the previous definition of the fenced code token renderer prototype:

```

7795 \let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
7796 \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 `\markdownOptionFrozenCache` plain \TeX option is disabled and the code block has not been previously typeset:

```

7797 \renewcommand\markdownRendererInputFencedCode[2]{%
7798   \def\next##1 ##2\relax{%
7799     \ifthenelse{\equal{##1}{dot}}{%

```

```

7800     \markdownIfOption{frozenCache}{}{}%
7801     \immediate\write18{%
7802         if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
7803         then
7804             dot -Tpdf -o #1.pdf #1;
7805             cp #1 #1.pdf.source;
7806         fi}}%

```

We include the typeset image using the image token renderer:

```

7807     \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:

```

7808     }{%
7809     \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
7810     }%
7811     }%
7812     \next#2 \relax}%

```

The `witiko/graphicx/http` theme stores the previous definition of the image token renderer prototype:

```

7813 \let\markdown@witiko@graphicx@http@oldRendererImagePrototype
7814 \markdownRendererImagePrototype

```

We load the catchfile and grffile packages, see also Section 1.1.3:

```

7815 \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.

```

7816 \newcount\markdown@witiko@graphicx@http@counter
7817 \markdown@witiko@graphicx@http@counter=0
7818 \newcommand\markdown@witiko@graphicx@http@filename{%
7819     \markdownOptionCacheDir/witiko_graphicx_http%
7820     .\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.

```

7821 \newcommand\markdown@witiko@graphicx@http@download[2]{%
7822     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:

```

7823 \begingroup
7824 \catcode`\%=12
7825 \catcode`\^^A=14

```

We redefine the image token renderer prototype, so that it tries to download an online image.

```
7826 \global\def\markdownRendererImagePrototype#1#2#3#4{^^A
7827 \begingroup
7828 \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 `\markdownOptionFrozenCache` plain TeX option is disabled:

```
7829 \markdownIfOption{frozenCache}{}{^^A
7830 \immediate\write18{^^A
7831 mkdir -p "\markdownOptionCacheDir";
7832 if printf '%s' "#3" | grep -q -E '^https?:';
7833 then
```

The image will be downloaded to the pathname `\markdownOptionCacheDir/⟨the MD5 digest of the image URL⟩.⟨the suffix of the image URL⟩`:

```
7834 OUTPUT_PREFIX="\markdownOptionCacheDir";
7835 OUTPUT_BODY="$(printf '%s' '#3' | md5sum | cut -d' ' -f1)";
7836 OUTPUT_SUFFIX="$(printf '%s' '#3' | sed 's/.*[.]/')";
7837 OUTPUT="$OUTPUT_PREFIX/$OUTPUT_BODY.$OUTPUT_SUFFIX";
```

The image will be downloaded only if it has not already been downloaded:

```
7838 if ! [ -e "$OUTPUT" ];
7839 then
7840 \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
7841 printf '%s' "$OUTPUT" > "\filename";
7842 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:

```
7843 else
7844 printf '%s' '#3' > "\filename";
7845 fi}}^^A
```

We load the pathname of the downloaded image and we typeset the image using the previous definition of the image renderer prototype:

```
7846 \CatchFileDef{\filename}{\filename}{\endlinechar=-1}^^A
7847 \markdown@witiko@graphicx@http@oldRendererImagePrototype^^A
7848 {#1}{#2}{\filename}{#4}^^A
7849 \endgroup
7850 \global\advance\markdown@witiko@graphicx@http@counter by 1\relax}^^A
7851 \endgroup
```

The `witiko/tilde` theme redefines the tilde token renderer prototype, so that it expands to a non-breaking space:

```
7852 \renewcommand\markdownRendererTildePrototype{~}%
```

3.3.3 Options

The supplied package options are processed using the `\markdownSetup` macro.

```
7853 \DeclareOption*{%
7854   \expandafter\markdownSetup\expandafter{\CurrentOption}}%
7855 \ProcessOptions\relax
```

After processing the options, activate the `jeekyllDataRenderers`, `renderers`, `rendererPrototypes`, and `code` keys.

```
7856 \ExplSyntaxOn
7857 \keys_define:nn
7858   { markdown/latex-options }
7859   {
7860     renderers .code:n = {
7861       \keys_set:nn
7862         { markdown/latex-options/renderers }
7863         { #1 }
7864     },
7865     rendererPrototypes .code:n = {
7866       \keys_set:nn
7867         { markdown/latex-options/renderer-prototypes }
7868         { #1 }
7869     },
```

The `code` key is used to immediately expand and execute code, which can be especially useful in \LaTeX setup snippets.

```
7870     code .code:n = { #1 },
```

The `jeekyllDataRenderers` key can be used as a syntactic sugar for setting the `markdown/jeekyllData` key-values (see Section 2.2.4.1) without using the `expl3` language.

```
7871     jeekyllDataRenderers .code:n = {
7872       \keys_set:nn
7873         { markdown/latex-options/jeekyll-data-renderers }
7874         { #1 }
7875     },
7876   }
7877 \keys_define:nn
7878   { markdown/latex-options/jeekyll-data-renderers }
7879   {
7880     unknown .code:n = {
7881       \tl_set_eq:NN
7882         \l_tmpa_tl
7883         \l_keys_key_str
7884       \tl_put_right:Nn
7885         \l_tmpa_tl
7886         {
7887           .code:n = { #1 }

```

```

7888     }
7889     \keys_define:nV
7890     { markdown/jekyllData }
7891     \l_tmpa_tl
7892   }
7893 }
7894 \cs_generate_variant:Nn
7895   \keys_define:nn
7896   { nV }
7897 \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.

```

7898 \markdownIfOption{plain}{\iffalse}{\iftrue}

```

If the `tightLists` Lua option is disabled or the current document class is beamer, do not load the `paralist` package.

```

7899 \markdownIfOption{tightLists}{
7900   \@ifclassloaded{beamer}{}{\RequirePackage{paralist}}}%
7901 }{}

```

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.

```

7902 \ExplSyntaxOn
7903 \@ifpackageloaded{paralist}{
7904   \tl_new:N
7905     \l_@@_latex_fancy_list_item_label_number_style_tl
7906   \tl_new:N
7907     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7908   \cs_new:Nn
7909     \@@_latex_fancy_list_item_label_number:nn
7910     {
7911       \str_case:nn
7912         { #1 }
7913         {
7914           { Decimal } { #2 }
7915           { LowerRoman } { \int_to_roman:n { #2 } }
7916           { UpperRoman } { \int_to_Roman:n { #2 } }
7917           { LowerAlpha } { \int_to_alph:n { #2 } }
7918           { UpperAlpha } { \int_to_alph:n { #2 } }
7919         }
7920     }
7921   \cs_new:Nn
7922     \@@_latex_fancy_list_item_label_delimiter:n

```

```

7923     {
7924         \str_case:nn
7925         { #1 }
7926         {
7927             { Default } { . }
7928             { OneParen } { ) }
7929             { Period } { . }
7930         }
7931     }
7932 \cs_new:Nn
7933 \@@_latex_fancy_list_item_label:nnn
7934 {
7935     \@@_latex_fancy_list_item_label_number:nn
7936     { #1 }
7937     { #3 }
7938     \@@_latex_fancy_list_item_label_delimiter:n
7939     { #2 }
7940 }
7941 \cs_new:Nn
7942 \@@_latex_paralist_style:nn
7943 {
7944     \str_case:nn
7945     { #1 }
7946     {
7947         { Decimal } { 1 }
7948         { LowerRoman } { i }
7949         { UpperRoman } { I }
7950         { LowerAlpha } { a }
7951         { UpperAlpha } { A }
7952     }
7953     \@@_latex_fancy_list_item_label_delimiter:n
7954     { #2 }
7955 }
7956 \markdownSetup{rendererPrototypes={
7957     ulBeginTight = {\begin{compactitem}},
7958     ulEndTight = {\end{compactitem}},
7959     fancyOlBegin = {
7960         \group_begin:
7961         \tl_set:Nn
7962         \l_@@_latex_fancy_list_item_label_number_style_tl
7963         { #1 }
7964         \tl_set:Nn
7965         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7966         { #2 }
7967         \tl_set:Nn
7968         \l_tmpa_tl
7969         { \begin{enumerate}[ ] }

```



```

7970     \tl_put_right:Nx
7971     \l_tmpa_tl
7972     { \@@_latex_paralist_style:nn { #1 } { #2 } }
7973     \tl_put_right:Nn
7974     \l_tmpa_tl
7975     { ] }
7976     \l_tmpa_tl
7977 },
7978 fancyOlEnd = {
7979     \end{enumerate}
7980     \group_end:
7981 },
7982 olBeginTight = {\begin{compactenum}},
7983 olEndTight = {\end{compactenum}},
7984 fancyOlBeginTight = {
7985     \group_begin:
7986     \tl_set:Nn
7987     \l_@@_latex_fancy_list_item_label_number_style_tl
7988     { #1 }
7989     \tl_set:Nn
7990     \l_@@_latex_fancy_list_item_label_delimiter_style_tl
7991     { #2 }
7992     \tl_set:Nn
7993     \l_tmpa_tl
7994     { \begin{compactenum}[ ] }
7995     \tl_put_right:Nx
7996     \l_tmpa_tl
7997     { \@@_latex_paralist_style:nn { #1 } { #2 } }
7998     \tl_put_right:Nn
7999     \l_tmpa_tl
8000     { ] }
8001     \l_tmpa_tl
8002 },
8003 fancyOlEndTight = {
8004     \end{compactenum}
8005     \group_end:
8006 },
8007 fancyOlItemWithNumber = {
8008     \item
8009     [
8010         \@@_latex_fancy_list_item_label:VVn
8011         \l_@@_latex_fancy_list_item_label_number_style_tl
8012         \l_@@_latex_fancy_list_item_label_delimiter_style_tl
8013         { #1 }
8014     ]
8015 },
8016 dlBeginTight = {\begin{compactdesc}},

```

```

8017     dlEndTight = {\end{compactdesc}}}}
8018   \cs_generate_variant:Nn
8019     \@@_latex_fancy_list_item_label:nnn
8020     { VVn }
8021   }{
8022     \markdownSetup{rendererPrototypes={
8023       ulBeginTight = {\markdownRendererUlBegin},
8024       ulEndTight = {\markdownRendererUlEnd},
8025       fancyOlBegin = {\markdownRendererOlBegin},
8026       fancyOlEnd = {\markdownRendererOlEnd},
8027       olBeginTight = {\markdownRendererOlBegin},
8028       olEndTight = {\markdownRendererOlEnd},
8029       fancyOlBeginTight = {\markdownRendererOlBegin},
8030       fancyOlEndTight = {\markdownRendererOlEnd},
8031       dlBeginTight = {\markdownRendererDlBegin},
8032       dlEndTight = {\markdownRendererDlEnd}}}
8033   }
8034   \ExplSyntaxOff
8035   \RequirePackage{amsmath,ifthen}

      Unless the unicode-math package has been loaded, load the amssymb package
      with symbols to be used for tickboxes.

8036   \ifpackageloaded{unicode-math}{
8037     \markdownSetup{rendererPrototypes={
8038       untickedBox = {\$ \mdlgwhtsquare$},
8039     }}
8040   }{
8041     \RequirePackage{amssymb}
8042     \markdownSetup{rendererPrototypes={
8043       untickedBox = {\$ \square$},
8044     }}
8045   }
8046   \RequirePackage{csvsimple}
8047   \RequirePackage{fancyvrb}
8048   \RequirePackage{graphicx}
8049   \markdownSetup{rendererPrototypes={
8050     lineBreak = {\},
8051     leftBrace = {\textbraceleft},
8052     rightBrace = {\textbraceright},
8053     dollarSign = {\textdollar},
8054     underscore = {\textunderscore},
8055     circumflex = {\textasciicircum},
8056     backslash = {\textbackslash},
8057     tilde = {\textasciitilde},
8058     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.

```

8059 codeSpan = {%
8060   \ifmmode
8061     \text{#1}%
8062   \else
8063     \texttt{#1}%
8064   \fi
8065 },
8066 contentBlock = {%
8067   \ifthenelse{\equal{#1}{csv}}{%
8068     \begin{table}%
8069       \begin{center}%
8070         \csvautotabular{#3}%
8071       \end{center}
8072     \ifx\empty#4\empty\else
8073       \caption{#4}%
8074     \fi
8075   \end{table}%
8076 }{%
8077   \ifthenelse{\equal{#1}{tex}}{%
8078     \catcode`\%=14\relax
8079     \catcode`\#=6\relax
8080     \input #3\relax
8081     \catcode`\%=12\relax
8082     \catcode`\#=12\relax
8083   }{%
8084     \markdownInput{#3}%
8085   }%
8086 }%
8087 },
8088 image = {%
8089   \begin{figure}%
8090     \begin{center}%
8091       \includegraphics{#3}%
8092     \end{center}%
8093     \ifx\empty#4\empty\else
8094       \caption{#4}%
8095     \fi
8096   \end{figure}},
8097 ulBegin = {\begin{itemize}},

```

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.

```

8098   ulEnd = {\end{itemize}},
8099   olBegin = {\begin{enumerate}},
8100   olItem = {\item{}},
8101   olItemWithNumber = {\item[#1.]},
8102   olEnd = {\end{enumerate}},
8103   dlBegin = {\begin{description}},
8104   dlItem = {\item[#1]},
8105   dlEnd = {\end{description}},
8106   emphasis = {\emph{#1}},
8107   tickedBox = {\$\boxtimes$},
8108   halfTickedBox = {\$\boxdot$},

```

If identifier attributes appear at the beginning of a section, we make the next heading produce the `\label` macro.

```

8109   headerAttributeContextBegin = {
8110     \markdownSetup{
8111       rendererPrototypes = {
8112         attributeIdentifier = {%
8113           \begingroup
8114           \def\next####1{%
8115             \def####1#####1{%
8116               \endgroup
8117               ####1{#####1}%
8118               \label{##1}%
8119             }%
8120           }%
8121           \next\markdownRendererHeadingOne
8122           \next\markdownRendererHeadingTwo
8123           \next\markdownRendererHeadingThree
8124           \next\markdownRendererHeadingFour
8125           \next\markdownRendererHeadingFive
8126           \next\markdownRendererHeadingSix
8127         },
8128       },
8129     }%
8130   },
8131   superscript = {\textsuperscript{#1}},
8132   subscript = {\textsubscript{#1}},
8133   blockQuoteBegin = {\begin{quotation}},
8134   blockQuoteEnd = {\end{quotation}},
8135   inputVerbatim = {\VerbatimInput{#1}},
8136   inputFencedCode = {%
8137     \ifx\relax#2\relax
8138       \VerbatimInput{#1}%
8139     \else
8140       \@ifundefined{minted@code}{%
8141         \@ifundefined{lst@version}{%

```

```

8142         \markdownRendererInputFencedCode{#1}{}%
      When the listings package is loaded, use it for syntax highlighting.
8143         }{%
8144         \lstinputlisting[language=#2]{#1}%
8145         }%

      When the minted package is loaded, use it for syntax highlighting. The minted
      package is preferred over listings.
8146         }{%
8147         \catcode`\#=6\relax
8148         \inputminted{#2}{#1}%
8149         \catcode`\#=12\relax
8150         }%
8151     \fi},
8152     horizontalRule = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
8153     footnote = {\footnote{#1}}}}

      Support the nesting of strong emphasis.
8154 \ExplSyntaxOn
8155 \def\markdownLATEXStrongEmphasis#1{%
8156   \str_if_in:NnTF
8157     \f@series
8158     { b }
8159     { \textnormal{#1} }
8160     { \textbf{#1} }
8161 }
8162 \ExplSyntaxOff
8163 \markdownSetup{rendererPrototypes={strongEmphasis={%
8164   \protect\markdownLATEXStrongEmphasis{#1}}}}

      Support LATEX document classes that do not provide chapters.
8165 \@ifundefined{chapter}{%
8166   \markdownSetup{rendererPrototypes = {
8167     headingOne = {\section{#1}},
8168     headingTwo = {\subsection{#1}},
8169     headingThree = {\subsubsection{#1}},
8170     headingFour = {\paragraph{#1}\leavevmode},
8171     headingFive = {\subparagraph{#1}\leavevmode}}}
8172 }{%
8173   \markdownSetup{rendererPrototypes = {
8174     headingOne = {\chapter{#1}},
8175     headingTwo = {\section{#1}},
8176     headingThree = {\subsection{#1}},
8177     headingFour = {\subsubsection{#1}},
8178     headingFive = {\paragraph{#1}\leavevmode},
8179     headingSix = {\subparagraph{#1}\leavevmode}}}
8180 }%

```

3.3.4.1 Tickboxes If the `taskLists` option is enabled, we will hide bullets in unordered list items with tickboxes.

```

8181 \markdownSetup{
8182   rendererPrototypes = {
8183     ulItem = {%
8184       \futurelet\markdownLaTeXCheckbox\markdownLaTeXUListItem
8185     },
8186   },
8187 }
8188 \def\markdownLaTeXUListItem{%
8189   \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
8190     \item[\markdownLaTeXCheckbox]%
8191     \expandafter\@gobble
8192   \else
8193     \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
8194       \item[\markdownLaTeXCheckbox]%
8195       \expandafter\expandafter\expandafter\@gobble
8196     \else
8197       \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
8198         \item[\markdownLaTeXCheckbox]%
8199         \expandafter\expandafter\expandafter\expandafter
8200         \expandafter\expandafter\expandafter\@gobble
8201       \else
8202         \item{}%
8203       \fi
8204     \fi
8205   \fi
8206 }

```

3.3.4.2 HTML elements If the `html` option is enabled and we are using $\text{\TeX}4\text{ht}$ ⁹, we will pass HTML elements to the output HTML document unchanged.

```

8207 \@ifundefined{HCode}{-}{
8208   \markdownSetup{
8209     rendererPrototypes = {
8210       inlineHtmlTag = {%
8211         \ifvmode
8212           \IgnorePar
8213         \EndP
8214         \fi
8215         \HCode{#1}%
8216       },
8217       inputBlockHtmlElement = {%
8218         \ifvmode
8219           \IgnorePar

```

⁹See <https://tug.org/tex4ht/>.

```

8220     \fi
8221     \EndP
8222     \special{t4ht*<#1}%
8223     \par
8224     \ShowPar
8225   },
8226 },
8227 }
8228 }

```

3.3.4.3 Citations Here is a basic implementation for citations that uses the \LaTeX `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the Bib \LaTeX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```

8229 \newcount\markdownLaTeXCitationsCounter
8230
8231 % Basic implementation
8232 \RequirePackage{gobble}
8233 \def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
8234   \advance\markdownLaTeXCitationsCounter by 1\relax
8235   \ifx\relax#4\relax
8236     \ifx\relax#5\relax
8237       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8238         \cite{#1#2#6}% Without prenotes and postnotes, just accumulate cites
8239         \expandafter\expandafter\expandafter
8240         \expandafter\expandafter\expandafter\expandafter
8241         \@gobblethree
8242       \fi
8243     \else% Before a postnote (#5), dump the accumulator
8244       \ifx\relax#1\relax\else
8245         \cite{#1}%
8246       \fi
8247       \cite[#5]{#6}%
8248       \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8249       \else
8250         \expandafter\expandafter\expandafter
8251         \expandafter\expandafter\expandafter\expandafter
8252         \expandafter\expandafter\expandafter
8253         \expandafter\expandafter\expandafter\expandafter
8254         \markdownLaTeXBasicCitations
8255       \fi
8256       \expandafter\expandafter\expandafter
8257       \expandafter\expandafter\expandafter\expandafter{%
8258       \expandafter\expandafter\expandafter\expandafter}%
8259       \expandafter\expandafter\expandafter\expandafter}%
8260       \expandafter\expandafter\expandafter

```

```

8261     \expandafter\expandafter\expandafter\expandafter{%
8262     \expandafter\expandafter\expandafter
8263     \expandafter\expandafter\expandafter\expandafter}%
8264     \expandafter\expandafter\expandafter
8265     \@gobblethree
8266     \fi
8267 \else% Before a prenote (#4), dump the accumulator
8268     \ifx\relax#1\relax\else
8269         \cite{#1}%
8270     \fi
8271     \ifnum\markdownLaTeXCitationsCounter>1\relax
8272         \space % Insert a space before the prenote in later citations
8273     \fi
8274     #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
8275     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8276     \else
8277         \expandafter\expandafter\expandafter
8278         \expandafter\expandafter\expandafter\expandafter
8279         \markdownLaTeXBasicCitations
8280     \fi
8281     \expandafter\expandafter\expandafter{%
8282     \expandafter\expandafter\expandafter}%
8283     \expandafter\expandafter\expandafter{%
8284     \expandafter\expandafter\expandafter}%
8285     \expandafter
8286     \@gobblethree
8287 \fi\markdownLaTeXBasicCitations{#1#2#6},}
8288 \let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations
8289
8290 % Natbib implementation
8291 \def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
8292     \advance\markdownLaTeXCitationsCounter by 1\relax
8293     \ifx\relax#3\relax
8294         \ifx\relax#4\relax
8295             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8296                 \citep{#1,#5}% Without prenotes and postnotes, just accumulate cites
8297                 \expandafter\expandafter\expandafter
8298                 \expandafter\expandafter\expandafter\expandafter
8299                 \@gobbletwo
8300             \fi
8301         \else% Before a postnote (#4), dump the accumulator
8302             \ifx\relax#1\relax\else
8303                 \citep{#1}%
8304             \fi
8305             \citep[] [#4]{#5}%
8306             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8307             \else

```



```

8308     \expandafter\expandafter\expandafter
8309     \expandafter\expandafter\expandafter\expandafter
8310     \expandafter\expandafter\expandafter
8311     \expandafter\expandafter\expandafter\expandafter
8312     \markdownLaTeXNatbibCitations
8313     \fi
8314     \expandafter\expandafter\expandafter
8315     \expandafter\expandafter\expandafter\expandafter{%
8316     \expandafter\expandafter\expandafter
8317     \expandafter\expandafter\expandafter\expandafter}%
8318     \expandafter\expandafter\expandafter
8319     \@gobbletwo
8320     \fi
8321 \else% Before a prenote (#3), dump the accumulator
8322     \ifx\relax#1\relax\relax\else
8323         \citep{#1}%
8324     \fi
8325     \citep[#3][#4]{#5}%
8326     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8327     \else
8328         \expandafter\expandafter\expandafter
8329         \expandafter\expandafter\expandafter\expandafter
8330         \markdownLaTeXNatbibCitations
8331     \fi
8332     \expandafter\expandafter\expandafter{%
8333     \expandafter\expandafter\expandafter}%
8334     \expandafter
8335     \@gobbletwo
8336     \fi\markdownLaTeXNatbibCitations{#1,#5}}
8337 \def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
8338     \advance\markdownLaTeXCitationsCounter by 1\relax
8339     \ifx\relax#3\relax
8340         \ifx\relax#4\relax
8341             \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8342                 \citet{#1,#5}% Without prenotes and postnotes, just accumulate cites
8343                 \expandafter\expandafter\expandafter
8344                 \expandafter\expandafter\expandafter\expandafter
8345                 \@gobbletwo
8346             \fi
8347         \else% After a prenote or a postnote, dump the accumulator
8348             \ifx\relax#1\relax\relax\else
8349                 \citet{#1}%
8350             \fi
8351             , \citet[#3][#4]{#5}%
8352             \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8353                 ,
8354             \else

```

```

8355     \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8356     ,
8357     \fi
8358     \fi
8359     \expandafter\expandafter\expandafter
8360     \expandafter\expandafter\expandafter\expandafter
8361     \markdownLaTeXNatbibTextCitations
8362     \expandafter\expandafter\expandafter
8363     \expandafter\expandafter\expandafter\expandafter{%
8364     \expandafter\expandafter\expandafter
8365     \expandafter\expandafter\expandafter\expandafter}%
8366     \expandafter\expandafter\expandafter
8367     \@gobbletwo
8368     \fi
8369     \else% After a prenote or a postnote, dump the accumulator
8370     \ifx\relax#1\relax\relax\else
8371     \citet{#1}%
8372     \fi
8373     , \citet{#3}[#4]{#5}%
8374     \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
8375     ,
8376     \else
8377     \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
8378     ,
8379     \fi
8380     \fi
8381     \expandafter\expandafter\expandafter
8382     \markdownLaTeXNatbibTextCitations
8383     \expandafter\expandafter\expandafter{%
8384     \expandafter\expandafter\expandafter}%
8385     \expandafter
8386     \@gobbletwo
8387     \fi\markdownLaTeXNatbibTextCitations{#1,#5}}
8388
8389 % BibLaTeX implementation
8390 \def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
8391     \advance\markdownLaTeXCitationsCounter by 1\relax
8392     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8393     \autocites#1[#3][#4]{#5}%
8394     \expandafter\@gobbletwo
8395     \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}
8396 \def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
8397     \advance\markdownLaTeXCitationsCounter by 1\relax
8398     \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
8399     \textcites#1[#3][#4]{#5}%
8400     \expandafter\@gobbletwo
8401     \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}

```

```

8402
8403 \markdownSetup{rendererPrototypes = {
8404   cite = {%
8405     \markdownLaTeXCitationsCounter=1%
8406     \def\markdownLaTeXCitationsTotal{#1}%
8407     \@ifundefined{autocites}{%
8408       \ifundefined{citep}{%
8409         \expandafter\expandafter\expandafter
8410         \markdownLaTeXBasicCitations
8411         \expandafter\expandafter\expandafter{%
8412         \expandafter\expandafter\expandafter}%
8413         \expandafter\expandafter\expandafter{%
8414         \expandafter\expandafter\expandafter}%
8415       }{%
8416         \expandafter\expandafter\expandafter
8417         \markdownLaTeXNatbibCitations
8418         \expandafter\expandafter\expandafter{%
8419         \expandafter\expandafter\expandafter}%
8420       }%
8421     }{%
8422       \expandafter\expandafter\expandafter
8423       \markdownLaTeXBibLaTeXCitations
8424       \expandafter{\expandafter}%
8425     }},
8426   textCite = {%
8427     \markdownLaTeXCitationsCounter=1%
8428     \def\markdownLaTeXCitationsTotal{#1}%
8429     \@ifundefined{autocites}{%
8430       \ifundefined{citep}{%
8431         \expandafter\expandafter\expandafter
8432         \markdownLaTeXBasicTextCitations
8433         \expandafter\expandafter\expandafter{%
8434         \expandafter\expandafter\expandafter}%
8435         \expandafter\expandafter\expandafter{%
8436         \expandafter\expandafter\expandafter}%
8437       }{%
8438         \expandafter\expandafter\expandafter
8439         \markdownLaTeXNatbibTextCitations
8440         \expandafter\expandafter\expandafter{%
8441         \expandafter\expandafter\expandafter}%
8442       }%
8443     }{%
8444       \expandafter\expandafter\expandafter
8445       \markdownLaTeXBibLaTeXTextCitations
8446       \expandafter{\expandafter}%
8447     }}}

```

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.

```

8448 \RequirePackage{url}
8449 \RequirePackage{expl3}
8450 \ExplSyntaxOn
8451 \def\markdownRendererLinkPrototype#1#2#3#4{
8452   \tl_set:Nn \l_tmpa_tl { #1 }
8453   \tl_set:Nn \l_tmpb_tl { #2 }
8454   \bool_set:Nn
8455     \l_tmpa_bool
8456     {
8457       \tl_if_eq_p:NN
8458         \l_tmpa_tl
8459         \l_tmpb_tl
8460     }
8461   \tl_set:Nn \l_tmpa_tl { #4 }
8462   \bool_set:Nn
8463     \l_tmpb_bool
8464     {
8465       \tl_if_empty_p:N
8466         \l_tmpa_tl
8467     }

```

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.

```

8468   \bool_if:nTF
8469     {
8470       \l_tmpa_bool && \l_tmpb_bool
8471     }
8472     {
8473       \markdownLaTeXRendererAutolink { #2 } { #3 }
8474     }{
8475       \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
8476     }
8477 }
8478 \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.

```

8479   \tl_set:Nn
8480     \l_tmpa_tl
8481     { #2 }
8482   \tl_trim_spaces:N
8483     \l_tmpa_tl
8484   \tl_set:Nx

```

```

8485     \l_tmpb_tl
8486     {
8487         \tl_range:Nnn
8488         \l_tmpa_tl
8489         { 1 }
8490         { 1 }
8491     }
8492     \str_if_eq:NNTF
8493     \l_tmpb_tl
8494     \c_hash_str
8495     {
8496         \tl_set:Nx
8497         \l_tmpb_tl
8498         {
8499             \tl_range:Nnn
8500             \l_tmpa_tl
8501             { 2 }
8502             { -1 }
8503         }
8504         \exp_args:NV
8505         \ref
8506         \l_tmpb_tl
8507     }{
8508         \url { #2 }
8509     }
8510 }
8511 \ExplSyntaxOff
8512 \def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
8513     #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.

```

8514 \newcount\markdownLaTeXRowCount
8515 \newcount\markdownLaTeXRowTotal
8516 \newcount\markdownLaTeXColumnCounter
8517 \newcount\markdownLaTeXColumnTotal
8518 \newtoks\markdownLaTeXTable
8519 \newtoks\markdownLaTeXTableAlignment
8520 \newtoks\markdownLaTeXTableEnd
8521 \AtBeginDocument{%
8522     \@ifpackageloaded{booktabs}{%
8523         \def\markdownLaTeXTopRule{\toprule}%
8524         \def\markdownLaTeXMidRule{\midrule}%
8525         \def\markdownLaTeXBottomRule{\bottomrule}%
8526     }{%
8527         \def\markdownLaTeXTopRule{\hline}%

```

```

8528     \def\markdownLaTeXMidRule{\hline}%
8529     \def\markdownLaTeXBottomRule{\hline}%
8530 }%
8531 }
8532 \markdownSetup{rendererPrototypes={
8533   table = {%
8534     \markdownLaTeXTable={}%
8535     \markdownLaTeXTableAlignment={}%
8536     \markdownLaTeXTableEnd={%
8537       \markdownLaTeXBottomRule
8538     \end{tabular}}}%
8539     \ifx\empty#1\empty\else
8540       \addto@hook\markdownLaTeXTable{%
8541         \begin{table}
8542         \centering}%
8543       \addto@hook\markdownLaTeXTableEnd{%
8544         \caption{#1}
8545         \end{table}}}%
8546     \fi
8547     \addto@hook\markdownLaTeXTable{\begin{tabular}}}%
8548     \markdownLaTeXRowCount=0%
8549     \markdownLaTeXRowTotal=#2%
8550     \markdownLaTeXColumnTotal=#3%
8551     \markdownLaTeXRenderTableRow
8552   }
8553 }}
8554 \def\markdownLaTeXRenderTableRow#1{%
8555   \markdownLaTeXColumnCounter=0%
8556   \ifnum\markdownLaTeXRowCount=0\relax
8557     \markdownLaTeXReadAlignments#1%
8558     \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
8559       \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
8560         \the\markdownLaTeXTableAlignment}}}%
8561     \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
8562   \else
8563     \markdownLaTeXRenderTableCell#1%
8564   \fi
8565   \ifnum\markdownLaTeXRowCount=1\relax
8566     \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
8567   \fi
8568   \advance\markdownLaTeXRowCount by 1\relax
8569   \ifnum\markdownLaTeXRowCount>\markdownLaTeXRowTotal\relax
8570     \the\markdownLaTeXTable
8571     \the\markdownLaTeXTableEnd
8572     \expandafter\@gobble
8573   \fi\markdownLaTeXRenderTableRow}
8574 \def\markdownLaTeXReadAlignments#1{%

```

```

8575 \advance\markdownLaTeXColumnCounter by 1\relax
8576 \if#1d%
8577   \addto@hook\markdownLaTeXTableAlignment{1}%
8578 \else
8579   \addto@hook\markdownLaTeXTableAlignment{#1}%
8580 \fi
8581 \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
8582   \expandafter\@gobble
8583 \fi\markdownLaTeXReadAlignments}
8584 \def\markdownLaTeXRenderTableCell#1{%
8585   \advance\markdownLaTeXColumnCounter by 1\relax
8586   \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
8587     \addto@hook\markdownLaTeXTable{#1&}%
8588   \else
8589     \addto@hook\markdownLaTeXTable{#1\\}%
8590     \expandafter\@gobble
8591   \fi\markdownLaTeXRenderTableCell}
8592 \fi

```

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.

```

8593 \ExplSyntaxOn
8594 \keys_define:nn
8595 { markdown/jekyllData }
8596 {
8597   author .code:n = { \author{#1} },
8598   date   .code:n = { \date{#1}   },
8599   title  .code:n = { \title{#1}  },
8600 }

```

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.

```

8601 % TODO: Remove the command definition in TeX Live 2021.
8602 \providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
8603 \markdownSetup{
8604   rendererPrototypes = {
8605     jekyllDataEnd = {
8606 %       TODO: Remove the else branch in TeX Live 2021.
8607       \IfFormatAtLeastTF
8608         { 2020-10-01 }
8609         { \AddToHook{begindocument/end}{\maketitle} }
8610       {
8611         \ifx\@onlypreamble\@notprerr

```

```

8612         % We are in the document
8613         \maketitle
8614     \else
8615         % We are in the preamble
8616         \RequirePackage{etoolbox}
8617         \AfterEndPreamble{\maketitle}
8618     \fi
8619 }
8620 },
8621 },
8622 }
8623 \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.

```

8624 \markdownIfOption{strikeThrough}{%
8625     \RequirePackage{soulutf8}%
8626     \markdownSetup{
8627         rendererPrototypes = {
8628             strikeThrough = {%
8629                 \st{#1}%
8630             },
8631         }
8632     }
8633 }{}

```

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.

```

8634 \newcommand\markdownMakeOther{%
8635     \count0=128\relax
8636     \loop
8637         \catcode\count0=11\relax
8638         \advance\count0 by 1\relax
8639     \ifnum\count0<256\repeat}%

```

3.4 ConT_EXt Implementation

The ConT_EXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConT_EXt formats *seem* to implement (the documentation is scarce) the majority of the plain T_EX format required by the plain

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.

```
8640 \def\markdownMakeOther{%
8641   \count0=128\relax
8642   \loop
8643     \catcode\count0=11\relax
8644     \advance\count0 by 1\relax
8645   \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.

```
8646   \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).

```
8647 \long\def\inputmarkdown{%
8648   \dosingleempty
8649   \doinputmarkdown}%
8650 \long\def\doinputmarkdown[#1]#2{%
8651   \begingroup
8652     \iffirstargument
8653     \setupmarkdown{#1}%
8654     \fi
8655     \markdownInput{#2}%
8656   \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. [11, sec. 31]. According to Eijkhout [12, 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.

```
8657 \ifx\startluacode\undefined % MkII
8658   \begingroup
8659     \catcode`\|=0%
8660     \catcode`\|=12%
8661     \gdef\startmarkdown{%
```

```

8662     |markdownReadAndConvert{\stopmarkdown}%
8663                               {|stopmarkdown}}%
8664     |gdef|stopmarkdown{%
8665       |markdownEnd}%
8666   |endgroup
8667 \else % MkIV
8668   \startluacode
8669     document.markdown_buffering = false
8670     local function preserve_trailing_spaces(line)
8671       if document.markdown_buffering then
8672         line = line:gsub("[ \t][ \t]$", "\t\t")
8673       end
8674       return line
8675     end
8676     resolvers.installinputlinehandler(preserve_trailing_spaces)
8677   \stopluacode
8678   \begingroup
8679     \catcode`\|=0%
8680     \catcode`\|=12%
8681     |gdef|startmarkdown{%
8682       |ctxlua{document.markdown_buffering = true}%
8683       |markdownReadAndConvert{\stopmarkdown}%
8684                               {|stopmarkdown}}%
8685     |gdef|stopmarkdown{%
8686       |ctxlua{document.markdown_buffering = false}%
8687       |markdownEnd}%
8688   |endgroup
8689 \fi

```

3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```

8690 \def\markdownRendererLineBreakPrototype{\blank}%
8691 \def\markdownRendererLeftBracePrototype{\textbraceleft}%
8692 \def\markdownRendererRightBracePrototype{\textbraceright}%
8693 \def\markdownRendererDollarSignPrototype{\textdollar}%
8694 \def\markdownRendererPercentSignPrototype{\percent}%
8695 \def\markdownRendererUnderscorePrototype{\textunderscore}%
8696 \def\markdownRendererCircumflexPrototype{\textcircumflex}%
8697 \def\markdownRendererBackslashPrototype{\textbackslash}%
8698 \def\markdownRendererTildePrototype{\textasciitilde}%
8699 \def\markdownRendererPipePrototype{\char`|}%
8700 \def\markdownRendererLinkPrototype#1#2#3#4{%
8701   \useURL[#1][#3][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
8702   \fi\tt<\hyphenatedurl{#3}>}}%
8703 \usemodule[database]
8704 \defineseparatedlist

```

```

8705 [MarkdownConTeXtCSV]
8706 [separator={,},
8707   before=\bTABLE,after=\eTABLE,
8708   first=\bTR,last=\eTR,
8709   left=\bTD,right=\eTD]
8710 \def\markdownConTeXtCSV{csv}
8711 \def\markdownRendererContentBlockPrototype#1#2#3#4{%
8712   \def\markdownConTeXtCSV@arg{#1}%
8713   \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
8714     \placetable[] [tab:#1]{#4}{%
8715       \processseparatedfile[MarkdownConTeXtCSV] [#3]}%
8716   \else
8717     \markdownInput{#3}%
8718   \fi}%
8719 \def\markdownRendererImagePrototype#1#2#3#4{%
8720   \placefigure[] []{#4}{\externalfigure[#3]}%
8721 \def\markdownRendererUlBeginPrototype{\startitemize}%
8722 \def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
8723 \def\markdownRendererUlItemPrototype{\item}%
8724 \def\markdownRendererUlEndPrototype{\stopitemize}%
8725 \def\markdownRendererUlEndTightPrototype{\stopitemize}%
8726 \def\markdownRendererOlBeginPrototype{\startitemize[n]}%
8727 \def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
8728 \def\markdownRendererOlItemPrototype{\item}%
8729 \def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
8730 \def\markdownRendererOlEndPrototype{\stopitemize}%
8731 \def\markdownRendererOlEndTightPrototype{\stopitemize}%
8732 \definedescription
8733 [MarkdownConTeXtDlItemPrototype]
8734 [location=hanging,
8735   margin=standard,
8736   headstyle=bold]%
8737 \definestartstop
8738 [MarkdownConTeXtDlPrototype]
8739 [before=\blank,
8740   after=\blank]%
8741 \definestartstop
8742 [MarkdownConTeXtDlTightPrototype]
8743 [before=\blank\startpacked,
8744   after=\stoppacked\blank]%
8745 \def\markdownRendererDlBeginPrototype{%
8746   \startMarkdownConTeXtDlPrototype}%
8747 \def\markdownRendererDlBeginTightPrototype{%
8748   \startMarkdownConTeXtDlTightPrototype}%
8749 \def\markdownRendererDlItemPrototype#1{%
8750   \startMarkdownConTeXtDlItemPrototype{#1}}%
8751 \def\markdownRendererDlItemEndPrototype{%

```

```

8752 \stopMarkdownConTeXtDlItemPrototype}%
8753 \def\markdownRendererDlEndPrototype{%
8754 \stopMarkdownConTeXtDlPrototype}%
8755 \def\markdownRendererDlEndTightPrototype{%
8756 \stopMarkdownConTeXtDlTightPrototype}%
8757 \def\markdownRendererEmphasisPrototype#1{\em#1}%
8758 \def\markdownRendererStrongEmphasisPrototype#1{\bf#1}%
8759 \def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
8760 \def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
8761 \def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
8762 \def\markdownRendererInputFencedCodePrototype#1#2{%
8763 \ifx\relax#2\relax
8764 \typefile{#1}%
8765 \else

```

The code fence infostring is used as a name from the ConT_EXt `\definetying` macro. This allows the user to set up code highlighting mapping as follows:

```

\definetying [latex]
\setuptying [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext

```

```

8766 \typefile[#2] [{#1}%
8767 \fi}%
8768 \def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
8769 \def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
8770 \def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
8771 \def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
8772 \def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
8773 \def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
8774 \def\markdownRendererHorizontalRulePrototype{%
8775 \blackrule[height=1pt, width=\hsize]}%
8776 \def\markdownRendererFootnotePrototype#1{\footnote{#1}}%
8777 \def\markdownRendererTickedBoxPrototype{\boxtimes$}
8778 \def\markdownRendererHalfTickedBoxPrototype{\boxdot$}
8779 \def\markdownRendererUntickedBoxPrototype{\square$}

```

```

8780 \def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
8781 \def\markdownRendererSuperscriptPrototype#1{\high{#1}}
8782 \def\markdownRendererSubscriptPrototype#1{\low{#1}}

```

3.4.2.1 Tables

There is a basic implementation of tables.

```

8783 \newcount\markdownConTeXtRowCounter
8784 \newcount\markdownConTeXtRowTotal
8785 \newcount\markdownConTeXtColumnCounter
8786 \newcount\markdownConTeXtColumnTotal
8787 \newtoks\markdownConTeXtTable
8788 \newtoks\markdownConTeXtTableFloat
8789 \def\markdownRendererTablePrototype#1#2#3{%
8790   \markdownConTeXtTable={}%
8791   \ifx\empty#1\empty
8792     \markdownConTeXtTableFloat={%
8793       \the\markdownConTeXtTable}%
8794   \else
8795     \markdownConTeXtTableFloat={%
8796       \placetable{#1}{\the\markdownConTeXtTable}}%
8797   \fi
8798   \begingroup
8799   \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
8800   \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
8801   \setupTABLE[r][1][topframe=on, bottomframe=on]
8802   \setupTABLE[r][#1][bottomframe=on]
8803   \markdownConTeXtRowCounter=0%
8804   \markdownConTeXtRowTotal=#2%
8805   \markdownConTeXtColumnTotal=#3%
8806   \markdownConTeXtRenderTableRow}
8807 \def\markdownConTeXtRenderTableRow#1{%
8808   \markdownConTeXtColumnCounter=0%
8809   \ifnum\markdownConTeXtRowCounter=0\relax
8810     \markdownConTeXtReadAlignments#1%
8811     \markdownConTeXtTable={\bTABLE}%
8812   \else
8813     \markdownConTeXtTable=\expandafter{%
8814       \the\markdownConTeXtTable\bTR}%
8815     \markdownConTeXtRenderTableRowCell#1%
8816     \markdownConTeXtTable=\expandafter{%
8817       \the\markdownConTeXtTable\eTR}%
8818   \fi
8819   \advance\markdownConTeXtRowCounter by 1\relax
8820   \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
8821     \markdownConTeXtTable=\expandafter{%
8822       \the\markdownConTeXtTable\eTABLE}%
8823     \the\markdownConTeXtTableFloat

```

```

8824 \endgroup
8825 \expandafter\gobbleoneargument
8826 \fi\markdownConTeXtRenderTableRow}
8827 \def\markdownConTeXtReadAlignments#1{%
8828 \advance\markdownConTeXtColumnCounter by 1\relax
8829 \if#1d%
8830 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8831 \fi\if#1l%
8832 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
8833 \fi\if#1c%
8834 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
8835 \fi\if#1r%
8836 \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
8837 \fi
8838 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8839 \expandafter\gobbleoneargument
8840 \fi\markdownConTeXtReadAlignments}
8841 \def\markdownConTeXtRenderTableCell#1{%
8842 \advance\markdownConTeXtColumnCounter by 1\relax
8843 \markdownConTeXtTable=\expandafter{%
8844 \the\markdownConTeXtTable\bTD#1\eTD}%
8845 \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
8846 \expandafter\gobbleoneargument
8847 \fi\markdownConTeXtRenderTableCell}
8848 \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] Donald Ervin Knuth. *The TeXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.
- [5] 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).

- [6] Till Tantau, Joseph Wright, and Vedran Miletic. *The Beamer class*. Feb. 10, 2021. URL: <https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf> (visited on 02/11/2021).
- [7] 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).
- [8] 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).
- [9] Roberto Ierusalimsky. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.
- [10] 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).
- [11] Donald Ervin Knuth. *T_EX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.
- [12] Victor Eijkhout. *T_EX by Topic. A T_EXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

Index

| | |
|---------------------------------------|-----------------|
| <code>\author</code> | 255 |
| <code>\autocites</code> | 247 |
| <code>blankBeforeBlockquote</code> | 15 |
| <code>blankBeforeCodeFence</code> | 15 |
| <code>blankBeforeHeading</code> | 15 |
| <code>breakableBlockquotes</code> | 16 |
| <code>cacheDir</code> | 14, 18, 40, 185 |
| <code>citationNbsps</code> | 16 |
| <code>citations</code> | 16, 70 |
| <code>\cite</code> | 247 |
| <code>\citep</code> | 247 |
| <code>\citett</code> | 247 |
| <code>codeSpans</code> | 17 |
| <code>compactdesc</code> | 4 |
| <code>compactenum</code> | 4 |
| <code>compactitem</code> | 4 |
| <code>contentBlocks</code> | 17 |
| <code>contentBlocksLanguageMap</code> | 18 |

| | |
|--|-----------------------------|
| convert | 224 |
| \csvautotabular | 4 |
| \date | 255 |
| defaultOptions | 8, 36, 212 |
| \definetyping | 260 |
| definitionLists | 18, 59 |
| \directlua | 4, 40, 231 |
| eagerCache | 18, 19 |
| \enableregime | 257 |
| \endmarkdown | 80 |
| entities.char_entity | 147 |
| entities.dec_entity | 147 |
| entities.hex_entity | 147 |
| escape_citation | 187, 187 |
| escaped_citation_chars | 187, 187 |
| expandtabs | 170 |
| expectJekyllData | 20 |
| extensions | 19, 98, 187 |
| extensions.citations | 187 |
| extensions.content_blocks | 190 |
| extensions.definition_lists | 193 |
| extensions.fancy_lists | 208 |
| extensions.fenced_code | 195 |
| extensions.footnotes | 197 |
| extensions.header_attributes | 199 |
| extensions.jekyll_data | 200 |
| extensions.pipe_table | 203 |
| extensions.strike_through | 206 |
| extensions.subscripts | 208 |
| extensions.superscripts | 207 |
| fancyLists | 22, 54–59 |
| fencedCode | 22, 63, 235 |
| \filecontents | 226 |
| finalizeCache | 14, 18, 19, 23, 24, 39, 186 |
| footnotes | 23, 70 |
| frozenCacheCounter | 24, 186, 232, 233 |
| frozenCacheFileName | 14, 23, 186 |
| \g_luaabridge_error_output_filename_str | 41 |
| \g_luaabridge_helper_script_filename_str | 40 |

| | |
|--|---------------------------------------|
| hardLineBreaks | 24 |
| hashEnumerators | 24 |
| headerAttributes | 25, 28, 73, 74 |
| html | 25, 71, 72, 246 |
| hybrid | 26, 32, 33, 42, 47, 87, 150, 171, 232 |
|
 | |
| \includegraphics | 4 |
| inlineFootnotes | 26 |
| \input | 37, 95, 227, 231 |
| \inputmarkdown | 95, 96, 257 |
| isdir | 3 |
| iterlines | 170 |
|
 | |
| jekyllData | 3, 20, 21, 26, 64-67 |
| \jobname | 40, 41 |
|
 | |
| \label | 244 |
| languages_json | 191, 191 |
|
 | |
| \maketitle | 255 |
| \markdown | 80 |
| markdown | 80, 80, 234 |
| markdown* | 80, 80, 81, 234 |
| \markdown_jekyll_data_concatenate_address:NN | 220 |
| \markdown_jekyll_data_pop: | 221 |
| \markdown_jekyll_data_push:nN | 221 |
| \markdown_jekyll_data_push_address_segment:n | 219 |
| \markdown_jekyll_data_set_keyval:Nn | 221 |
| \markdown_jekyll_data_set_keyvals:nn | 221 |
| \markdown_jekyll_data_update_address_tls: | 220 |
| \markdownBegin | 38, 38, 39, 77, 80, 95 |
| \markdownEnd | 38, 38, 39, 77, 80, 95 |
| \markdownError | 77, 77 |
| \markdownExecute | 230 |
| \markdownExecuteDirect | 229, 230 |
| \markdownExecuteShellEscape | 229, 230 |
| \markdownIfOption | 225 |
| \markdownIfSnippetExists | 81 |
| \markdownInfo | 77 |
| \markdownInput | 38, 39, 80, 81, 96, 232, 234 |
| \markdownInputFileStream | 226 |
| \markdownInputPlainTeX | 234 |
| \markdownLuaExecute | 229, 230, 231, 232 |

| | |
|---|----------------------------------|
| <code>\markdownLuaOptions</code> | 223, 225 |
| <code>\markdownLuaRegisterIBCallback</code> | 79 |
| <code>\markdownLuaUnregisterIBCallback</code> | 79 |
| <code>\markdownMakeOther</code> | 77, 256, 257 |
| <code>\markdownMode</code> | 4, 41, 78, 78, 228, 229, 231 |
| <code>\markdownOptionCacheDir</code> | 3, 92, 224, 237 |
| <code>\markdownOptionErrorTempFileName</code> | 41, 41, 230 |
| <code>\markdownOptionFinalizeCache</code> | 39, 40, 91 |
| <code>\markdownOptionFrozenCache</code> | 14, 23, 40, 40, 86, 91, 235, 237 |
| <code>\markdownOptionFrozenCacheFileName</code> | 39 |
| <code>\markdownOptionHelperScriptFileName</code> | 40, 40–42, 230, 231 |
| <code>\markdownOptionHybrid</code> | 42, 92 |
| <code>\markdownOptionInputTempFileName</code> | 40, 226, 227 |
| <code>\markdownOptionOutputDir</code> | 41 |
| <code>\markdownOptionOutputTempFileName</code> | 41, 41, 231 |
| <code>\markdownOptionSmartEllipses</code> | 92 |
| <code>\markdownOptionStripPercentSigns</code> | 44, 226, 227 |
| <code>\markdownOutputFileStream</code> | 226 |
| <code>\markdownPrepare</code> | 224 |
| <code>\markdownPrepareLuaOptions</code> | 223 |
| <code>\markdownReadAndConvert</code> | 77, 226, 234, 257 |
| <code>\markdownReadAndConvertProcessLine</code> | 227, 228 |
| <code>\markdownReadAndConvertStripPercentSigns</code> | 227 |
| <code>\markdownReadAndConvertTab</code> | 226 |
| <code>\markdownRendererAttributeClassName</code> | 73 |
| <code>\markdownRendererAttributeIdentifier</code> | 73 |
| <code>\markdownRendererAttributeKeyValue</code> | 73 |
| <code>\markdownRendererBlockHtmlCommentBegin</code> | 71 |
| <code>\markdownRendererBlockHtmlCommentEnd</code> | 71 |
| <code>\markdownRendererBlockQuoteBegin</code> | 63 |
| <code>\markdownRendererBlockQuoteEnd</code> | 63 |
| <code>\markdownRendererCite</code> | 70, 70 |
| <code>\markdownRendererCodeSpan</code> | 50 |
| <code>\markdownRendererCodeSpanPrototype</code> | 94 |
| <code>\markdownRendererContentBlock</code> | 51, 51 |
| <code>\markdownRendererContentBlockCode</code> | 52 |
| <code>\markdownRendererContentBlockOnlineImage</code> | 51 |
| <code>\markdownRendererDlBegin</code> | 59 |
| <code>\markdownRendererDlBeginTight</code> | 60 |
| <code>\markdownRendererDlDefinitionBegin</code> | 61 |
| <code>\markdownRendererDlDefinitionEnd</code> | 61 |
| <code>\markdownRendererDlEnd</code> | 61 |

| | |
|--|--------|
| \markdownRendererDlEndTight | 62 |
| \markdownRendererDlItem | 60 |
| \markdownRendererDlItemEnd | 60 |
| \markdownRendererDocumentBegin | 45 |
| \markdownRendererDocumentEnd | 45 |
| \markdownRendererEllipsis | 29, 47 |
| \markdownRendererEmphasis | 62, 93 |
| \markdownRendererFancyOlBegin | 55, 56 |
| \markdownRendererFancyOlBeginTight | 55 |
| \markdownRendererFancyOlEnd | 59 |
| \markdownRendererFancyOlEndTight | 59 |
| \markdownRendererFancyOlItem | 57 |
| \markdownRendererFancyOlItemEnd | 57 |
| \markdownRendererFancyOlItemWithNumber | 57 |
| \markdownRendererFootnote | 70 |
| \markdownRendererHalfTickedBox | 44 |
| \markdownRendererHeaderAttributeContextBegin | 74 |
| \markdownRendererHeaderAttributeContextEnd | 74 |
| \markdownRendererHeadingFive | 69 |
| \markdownRendererHeadingFour | 68 |
| \markdownRendererHeadingOne | 67 |
| \markdownRendererHeadingSix | 69 |
| \markdownRendererHeadingThree | 68 |
| \markdownRendererHeadingTwo | 68 |
| \markdownRendererHorizontalRule | 69 |
| \markdownRendererImage | 51 |
| \markdownRendererImagePrototype | 94 |
| \markdownRendererInlineHtmlComment | 71 |
| \markdownRendererInlineHtmlTag | 72 |
| \markdownRendererInputBlockHtmlElement | 72 |
| \markdownRendererInputFencedCode | 63 |
| \markdownRendererInputVerbatim | 63 |
| \markdownRendererInterblockSeparator | 46 |
| \markdownRendererJekyllDataBegin | 64 |
| \markdownRendererJekyllDataBoolean | 66 |
| \markdownRendererJekyllDataEmpty | 67 |
| \markdownRendererJekyllDataEnd | 64 |
| \markdownRendererJekyllDataMappingBegin | 64 |
| \markdownRendererJekyllDataMappingEnd | 65 |
| \markdownRendererJekyllDataNumber | 66 |
| \markdownRendererJekyllDataSequenceBegin | 65 |
| \markdownRendererJekyllDataSequenceEnd | 66 |

| | |
|--|------------------|
| <code>\markdownRendererJekyllDataString</code> | 67 |
| <code>\markdownRendererLineBreak</code> | 46 |
| <code>\markdownRendererLink</code> | 50, 93 |
| <code>\markdownRendererNbsp</code> | 47 |
| <code>\markdownRendererOlBegin</code> | 54 |
| <code>\markdownRendererOlBeginTight</code> | 55 |
| <code>\markdownRendererOlEnd</code> | 58 |
| <code>\markdownRendererOlEndTight</code> | 58 |
| <code>\markdownRendererOlItem</code> | 29, 56 |
| <code>\markdownRendererOlItemEnd</code> | 56 |
| <code>\markdownRendererOlItemWithNumber</code> | 29, 56 |
| <code>\markdownRendererStrikeThrough</code> | 75 |
| <code>\markdownRendererStrongEmphasis</code> | 62 |
| <code>\markdownRendererSubscript</code> | 75 |
| <code>\markdownRendererSuperscript</code> | 75 |
| <code>\markdownRendererTable</code> | 71 |
| <code>\markdownRendererTextCite</code> | 70 |
| <code>\markdownRendererTickedBox</code> | 44 |
| <code>\markdownRendererUlBegin</code> | 52 |
| <code>\markdownRendererUlBeginTight</code> | 53 |
| <code>\markdownRendererUlEnd</code> | 54 |
| <code>\markdownRendererUlEndTight</code> | 54 |
| <code>\markdownRendererUlItem</code> | 53 |
| <code>\markdownRendererUlItemEnd</code> | 53 |
| <code>\markdownRendererUntickedBox</code> | 44 |
| <code>\markdownSetup</code> | 81, 81, 233, 238 |
| <code>\markdownSetupSnippet</code> | 81, 81 |
| <code>\markdownWarning</code> | 77 |
| <code>new</code> | 6, 212 |
| <code>os.execute</code> | 78, 229 |
| <code>\PackageError</code> | 79, 234 |
| <code>\PackageInfo</code> | 79, 234 |
| <code>\PackageWarning</code> | 79, 234 |
| <code>parsers</code> | 157, 169 |
| <code>parsers.commented_line</code> | 159 |
| <code>\pdfshellescape</code> | 229 |
| <code>pipeTables</code> | 6, 27, 31, 71 |
| <code>preserveTabs</code> | 27, 30, 170 |
| <code>print</code> | 230, 231 |

| | |
|-------------------------------|---------------------------|
| reader | 7, 98, 157, 169, 187 |
| reader->add_special_character | 7, 8, 183 |
| reader->create_parser | 170 |
| reader->finalize_grammar | 180 |
| reader->insert_pattern | 7, 8, 181 |
| reader->normalize_tag | 169 |
| reader->options | 169 |
| reader->parser_functions | 170 |
| reader->parser_functions.name | 170 |
| reader->parsers | 169, 169 |
| reader->update_rule | 181, 183 |
| reader->writer | 169 |
| reader.new | 169, 169 |
| relativeReferences | 28 |
| | |
| \setupmarkdown | 96, 96 |
| \shellescape | 229 |
| shiftHeadings | 6, 28 |
| slice | 6, 25, 28, 148 |
| smartEllipses | 29, 47 |
| \startmarkdown | 95, 95, 257 |
| startNumber | 29, 56, 57 |
| status.shell_escape | 229 |
| \stopmarkdown | 95, 95, 257 |
| strikeThrough | 30, 75, 256 |
| stripIndent | 30, 170 |
| subscripts | 30, 75 |
| superscripts | 31, 75 |
| syntax | 182, 184 |
| | |
| tableCaptions | 6, 31 |
| taskLists | 32, 44, 246 |
| tex.print | 230, 231 |
| texComments | 32, 171 |
| \textcites | 247 |
| tightLists | 33, 53–55, 58–60, 62, 239 |
| \title | 255 |
| | |
| underscores | 33 |
| \url | 4 |
| \usepackage | 79, 83 |
| \usetheme | 83 |
| util.cache | 99 |

| | |
|---------------------------------|-----------------------|
| util.err | 99 |
| util.escaper | 102 |
| util.expand_tabs_in_line | 100 |
| util.flatten | 101 |
| util.intersperse | 102 |
| util.lookup_files | 99 |
| util.map | 102 |
| util.pathname | 103 |
| util.rope_last | 101 |
| util.rope_to_string | 101 |
| util.table_copy | 99 |
| util.walk | 100, 101 |
|
\VerbatimInput |
4 |
| walkable_syntax | 7, 181, 181, 183, 184 |
| writer | 98, 98, 147, 187 |
| writer->active_attributes | 154 |
| writer->active_headings | 154 |
| writer->block_html_comment | 152 |
| writer->block_html_element | 153 |
| writer->blockquote | 153 |
| writer->bulletitem | 151 |
| writer->bulletlist | 151 |
| writer->citation | 187 |
| writer->citations | 187 |
| writer->code | 150 |
| writer->codeFence | 196 |
| writer->contentblock | 191 |
| writer->defer_call | 157, 157 |
| writer->definitionlist | 194 |
| writer->document | 154 |
| writer->ellipsis | 149 |
| writer->emphasis | 153 |
| writer->escape | 150, 150 |
| writer->escape_minimal | 150, 150, 187 |
| writer->escape_uri | 150, 150 |
| writer->escaped_chars | 149, 150 |
| writer->escaped_minimal_strings | 149, 150 |
| writer->escaped_uri_chars | 149, 150 |
| writer->fancyitem | 209 |
| writer->fancylist | 208 |

| | |
|-----------------------------|----------|
| writer->get_state | 157 |
| writer->heading | 154 |
| writer->hrule | 149 |
| writer->hybrid | 187 |
| writer->image | 151 |
| writer->inline_html_comment | 152 |
| writer->inline_html_tag | 152 |
| writer->interblocksep | 149 |
| writer->is_writing | 148, 148 |
| writer->jekyllData | 200 |
| writer->linebreak | 149 |
| writer->link | 150 |
| writer->nbsp | 148 |
| writer->note | 198 |
| writer->options | 148 |
| writer->ordereditem | 152 |
| writer->orderedlist | 151 |
| writer->pack | 149, 186 |
| writer->paragraph | 149 |
| writer->plain | 148 |
| writer->set_state | 157 |
| writer->slice_begin | 148 |
| writer->slice_end | 148 |
| writer->space | 148 |
| writer->strike_through | 207 |
| writer->string | 150 |
| writer->strong | 153 |
| writer->subscript | 208 |
| writer->suffix | 148 |
| writer->superscript | 207 |
| writer->table | 205 |
| writer->checkbox | 153 |
| writer->uri | 150 |
| writer->verbatim | 153 |
| writer.new | 147, 147 |
| \writestatus | 95 |