A Markdown Interpreter for TEX

Vít Novotný witiko@mail.muni.cz

Version 2.20.0-0-gf64ade1
2023-02-01

Contents

1 Introduction **1**
1.1 Requirements 2
1.2 Feedback 5
1.3 Acknowledgements 6

2 Interfaces **6**
2.1 Lua Interface 6
2.2 Plain TEX Interface 43
2.3 LATEX Interface 90
2.4 ConTEXt Interface 106

3 Implementation **110**
3.1 Lua Implementation . . . 110
3.2 Plain TEX Implementation 238
3.3 LATEX Implementation . . 257
3.4 ConTEXt Implementation 283

References **290**

Index **291**

List of Figures

1 A block diagram of the Markdown package 7
2 A sequence diagram of typesetting a document using the TEX interface . . 39
3 A sequence diagram of typesetting a document using the Lua CLI 40
4 Various formats of mathemathical formulae 97
5 The banner of the Markdown package 98
6 A pushdown automaton that recognizes TEX comments 173

1 Introduction

The Markdown package[1] converts markdown[2] markup to TEX commands. The functionality is provided both as a Lua module and as plain TEX, LATEX, and ConTEXt macro packages that can be used to directly typeset TEX documents containing markdown markup. Unlike other convertors, the Markdown package does not require any external programs, and makes it easy to redefine how each and every markdown element is rendered. Creative abuse of the markdown syntax is encouraged. 😉

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

[1] See https://ctan.org/pkg/markdown.
[2] See 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]

```
local metadata = {
    version   = "(((VERSION)))",
    comment   = "A module for the conversion from markdown to plain TeX",
    author    = "John MacFarlane, Hans Hagen, Vít Novotný",
    copyright = {"2009-2016 John MacFarlane, Hans Hagen",
                 "2016-2022 Vít Novotný"},
    license   = "LPPL 1.3c"
}

if not modules then modules = { } end
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 $\geqslant$ 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 $\geqslant$ 0.10 is included in LuaTEX $\geqslant$ 0.72.0 (TEXLive $\geqslant$ 2013).

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

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

```
local unicode
(function()
  local ran_ok
  ran_ok, unicode = pcall(require, "unicode")
```

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

```
  if not ran_ok then
```

[3] See http://mirrors.ctan.org/macros/generic/markdown/markdown.html.

```
    unicode = {["utf8"]={char=utf8.char}}
  end
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 ⩾ 2008).

```
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 LaTeX3 kernel in TeX Live ⩽ 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.

```
⟨@@=markdown⟩
\ifx\ExplSyntaxOn\undefined
  \input expl3-generic\relax
\fi
```

lt3luabridge A package that allows us to execute Lua code with LuaTeX as well as with other TeX engines that provide the *shell escape* capability, which allows them to execute code with the system's shell.

The plain TeX part of the package also requires the following Lua module:

Lua File System A library that provides access to the filesystem via OS-specific syscalls. It is used by the plain TeX code to create the cache directory specified by the `cacheDir` option before interfacing with the Lunamark library. Lua File System is included in all releases of LuaTeX (TeXLive ⩾ 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 LaTeX Requirements

The LaTeX part of the package requires that the LaTeX 2$_\varepsilon$ 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 LaTeX 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 as well as the rendering of fancy lists.

ifthen A package that provides a concise syntax for the inspection of macro values. It is used in the `witiko/dot` LaTeX theme (see Section 2.3.2.2).

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

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

gobble A package that provides the `\@gobblethree` TeX command that is used in the default renderer prototype for citations. The package is included in TeXLive $\geqslant$ 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` LaTeX 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 ⩽ TeX Live ⩽ 2019. Since TeX Live ⩾ 2020, the functionality of the package has been integrated in the LaTeX 2_ε kernel. It is used in the `witiko/dot` and `witiko/graphicx/http` LaTeX 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 3.3.4.8, and also in the default renderer prototype for identifier attributes.

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

ltxcmds A package that is used to detect whether the minted and listings packages are loaded in the default renderer prototype for fenced code blocks.

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

```
\RequirePackage{expl3}
```

1.1.4 ConTeXt Prerequisites

The ConTeXt part of the package requires that either the Mark II or the Mark IV format is loaded, all the plain TeX prerequisites (see Section 1.1.2), and the following ConTeXt 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 GitHub[4] to report bugs and submit feature requests. If you do not want to report a bug or request a feature but are simply in need of assistance, you might want to consider posting your question to the TeX-LaTeX Stack Exchange.[5] community question answering web site under the `markdown` tag.

[4] See `https://github.com/witiko/markdown/issues`.
[5] See `https://tex.stackexchange.com`.

1.3 Acknowledgements

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

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

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

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

The TEX implementation of the package draws inspiration from several sources including the source code of LATEX 2_ε, 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 ConTEXt 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}
```

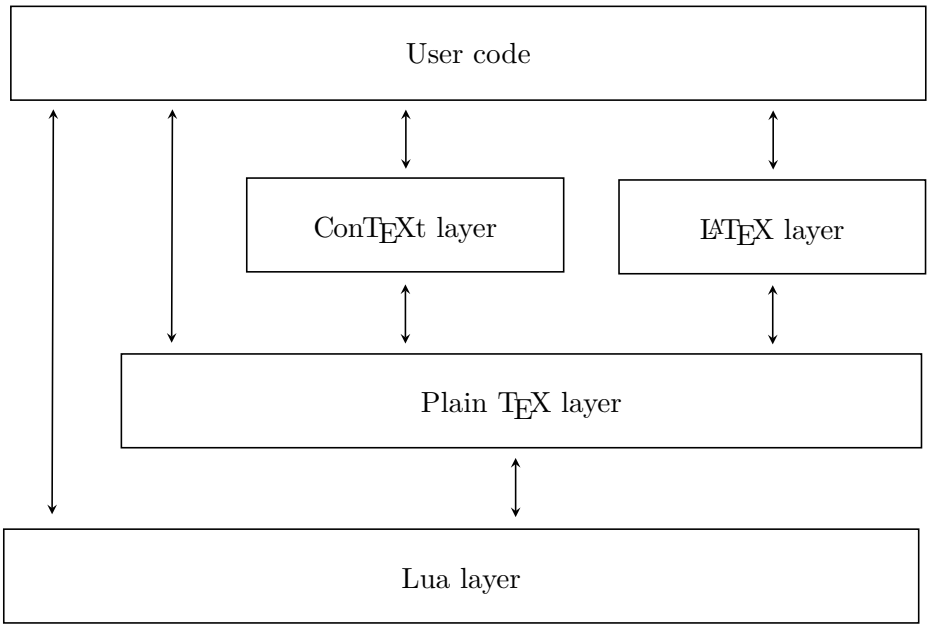

Figure 1: A block diagram of the Markdown package

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.

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

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

2.1.2 User-Defined Syntax Extensions

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

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

```
local walkable_syntax = {
  Block = {
    "Blockquote",
    "Verbatim",
    "ThematicBreak",
    "BulletList",
    "OrderedList",
    "Heading",
    "DisplayHtml",
    "Paragraph",
    "Plain",
  },
  Inline = {
    "Str",
    "Space",
    "Endline",
    "UlOrStarLine",
    "Strong",
    "Emph",
    "Link",
    "Image",
    "Code",
    "AutoLinkUrl",
    "AutoLinkEmail",
    "AutoLinkRelativeReference",
    "InlineHtml",
    "HtmlEntity",
    "EscapedChar",
    "Smart",
    "Symbol",
  },
}
```

The `reader->insert_pattern` method inserts a PEG pattern into the grammar of markdown. The method receives two mandatory arguments: a selector string in the form "⟨*left-hand side terminal symbol*⟩ ⟨*before*, *after*, *or* *instead of*⟩ ⟨*right-hand side terminal symbol*⟩" and a PEG pattern to insert, and an optional third argument with a name of the PEG pattern for debugging purposes (see the `debugExtensions` option). The name does not need to be unique and shall not be interpreted by the Markdown package; you can treat it as a comment.

For example. if we'd like to insert `pattern` into the grammar between the `Inline -> Emph` and `Inline -> Link` rules, we would call `reader->insert_pattern`

with `"Inline after Emph"` (or `"Inline before Link"`) and `pattern` as the arguments.

The `reader->add_special_character` method adds a new character with special meaning to the grammar of markdown. The method receives the character as its only argument.

2.1.3 Options

The Lua interface recognizes the following options. When unspecified, the value of a key is taken from the `defaultOptions` table.

```
local defaultOptions = {}
```

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

```
\ExplSyntaxOn
\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.

```
\prop_new:N \g_@@_lua_option_types_prop
\prop_new:N \g_@@_default_lua_options_prop
\seq_new:N \g_@@_option_layers_seq
\tl_const:Nn \c_@@_option_layer_lua_tl { lua }
\seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_lua_tl
\cs_new:Nn
  \@@_add_lua_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_lua_tl
      { #1 }
      { #2 }
      { #3 }
  }
\cs_new:Nn
  \@@_add_option:nnnn
  {
    \seq_gput_right:cn
      { g_@@_ #1 _options_seq }
      { #2 }
    \prop_gput:cnn
      { g_@@_ #1 _option_types_prop }
      { #2 }
      { #3 }
    \prop_gput:cnn
      { g_@@_default_ #1 _options_prop }
```

```
      { #2 }
      { #4 }
    \@@_typecheck_option:n
      { #2 }
  }
\cs_generate_variant:Nn
  \@@_add_option:nnnn
  { Vnnn }
\tl_const:Nn \c_@@_option_value_true_tl  { true  }
\tl_const:Nn \c_@@_option_value_false_tl { false }
\cs_new:Nn \@@_typecheck_option:n
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_case_e:Vn
      \l_tmpa_tl
      {
        { \c_@@_option_type_boolean_tl }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \bool_if:nF
              {
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_true_tl ||
                \str_if_eq_p:VV
                  \l_tmpa_tl
                  \c_@@_option_value_false_tl
              }
              {
                \msg_error:nnnV
                  { @@ }
                  { failed-typecheck-for-boolean-option }
                  { #1 }
                  \l_tmpa_tl
              }
          }
      }
  }
\msg_new:nnn
  { @@ }
  { failed-typecheck-for-boolean-option }
  {
    Option~#1~has~value~#2,~
```

```
    but~a~boolean~(true~or~false)~was~expected.
  }
\cs_generate_variant:Nn
  \str_case_e:nn
  { Vn }
\cs_generate_variant:Nn
  \msg_error:nnnn
  { nnnV }
\seq_new:N \g_@@_option_types_seq
\tl_const:Nn \c_@@_option_type_clist_tl { clist }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_clist_tl
\tl_const:Nn \c_@@_option_type_counter_tl { counter }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_counter_tl
\tl_const:Nn \c_@@_option_type_boolean_tl { boolean }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_boolean_tl
\tl_const:Nn \c_@@_option_type_number_tl  { number  }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_number_tl
\tl_const:Nn \c_@@_option_type_path_tl    { path    }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_path_tl
\tl_const:Nn \c_@@_option_type_slice_tl   { slice   }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_slice_tl
\tl_const:Nn \c_@@_option_type_string_tl  { string  }
\seq_gput_right:NV \g_@@_option_types_seq \c_@@_option_type_string_tl
\cs_new:Nn
  \@@_get_option_type:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_ ##1 _option_types_prop }
          { #1 }
          \l_tmpa_tl
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
      \l_tmpa_bool
      {
        \msg_error:nnn
          { @@ }
          { undefined-option }
```

```
          { #1 }
      }
    \seq_if_in:NVF
      \g_@@_option_types_seq
      \l_tmpa_tl
      {
        \msg_error:nnnV
          { @@ }
          { unknown-option-type }
          { #1 }
          \l_tmpa_tl
      }
    \tl_set_eq:NN
      #2
      \l_tmpa_tl
  }
\msg_new:nnn
  { @@ }
  { unknown-option-type }
  {
    Option~#1~has~unknown~type~#2.
  }
\msg_new:nnn
  { @@ }
  { undefined-option }
  {
    Option~#1~is~undefined.
  }
\cs_new:Nn
  \@@_get_default_option_value:nN
  {
    \bool_set_false:N
      \l_tmpa_bool
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \prop_get:cnNT
          { g_@@_default_ ##1 _options_prop }
          { #1 }
          #2
          {
            \bool_set_true:N
              \l_tmpa_bool
            \seq_map_break:
          }
      }
    \bool_if:nF
```

```
      \l_tmpa_bool
      {
        \msg_error:nnn
          { @@ }
          { undefined-option }
          { #1 }
      }
  }
\cs_new:Nn
  \@@_get_option_value:nN
  {
    \@@_option_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \cs_if_free:cTF
      { \l_tmpa_tl }
      {
        \@@_get_default_option_value:nN
          { #1 }
          #2
      }
      {
        \@@_get_option_type:nN
          { #1 }
          \l_tmpa_tl
        \str_if_eq:NNTF
          \c_@@_option_type_counter_tl
          \l_tmpa_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nx
              #2
              { \the \cs:w \l_tmpa_tl \cs_end: }
          }
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \tl_set:Nv
              #2
              { \l_tmpa_tl }
          }
      }
  }
\cs_new:Nn \@@_option_tl_to_csname:nN
```

```
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownOption
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
\seq_new:N \g_@@_cases_seq
\cs_new:Nn \@@_with_various_cases:nn
  {
    \seq_clear:N
      \l_tmpa_seq
    \seq_map_inline:Nn
      \g_@@_cases_seq
      {
        \tl_set:Nn
          \l_tmpa_tl
          { #1 }
        \use:c { ##1 }
          \l_tmpa_tl
        \seq_put_right:NV
          \l_tmpa_seq
          \l_tmpa_tl
      }
    \seq_map_inline:Nn
      \l_tmpa_seq
      { #2 }
  }
\cs_new:Nn \@@_camel_case:N
  {
    \regex_replace_all:nnN
      { _ ([a-z]) }
      { \c { str_uppercase:n } \cB\{ \1 \cE\} }
      #1
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_camel_case:N }
\cs_new:Nn \@@_snake_case:N
  {
    \regex_replace_all:nnN
```

```
      { ([a-z])([A-Z]) }
      { \1 _ \c { str_lowercase:n } \cB\{ \2 \cE\} }
      #1
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_snake_case:N }
```

2.1.4 File and Directory Names

`cacheDir`=⟨*path*⟩ default: `.`

A path to the directory containing auxiliary cache files. If the last segment of the path does not exist, it will be created by the Lua command-line and plain TEX 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.

```
\@@_add_lua_option:nnn
  { cacheDir }
  { path }
  { \markdownOptionOutputDir / _markdown_\jobname }
defaultOptions.cacheDir = "."
```

`contentBlocksLanguageMap`=⟨*filename*⟩
default: `markdown-languages.json`

The filename of the JSON file that maps filename extensions to programming language names in the iA,Writer content blocks when the `contentBlocks` option is enabled. See Section 2.2.3.7 for more information.

```
\@@_add_lua_option:nnn
  { contentBlocksLanguageMap }
  { path }
  { markdown-languages.json }
defaultOptions.contentBlocksLanguageMap = "markdown-languages.json"
```

`debugExtensionsFileName`=⟨*filename*⟩ default: `debug-extensions.json`

The filename of the JSON file that will be produced when the `debugExtensions` option is enabled. This file will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied.

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

`frozenCacheFileName`=⟨*path*⟩ default: `frozenCache.tex`

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

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

```
\@@_add_lua_option:nnn
  { frozenCacheFileName }
  { path }
  { \markdownOptionCacheDir / frozenCache.tex }
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.

```
\@@_add_lua_option:nnn
  { blankBeforeBlockquote }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { blankBeforeCodeFence }
  { boolean }
  { false }
defaultOptions.blankBeforeCodeFence = false
```

`blankBeforeDivFence`=`true`, `false` default: `false`

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

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

```
\@@_add_lua_option:nnn
  { blankBeforeDivFence }
  { boolean }
  { false }
defaultOptions.blankBeforeDivFence = false
```

`blankBeforeHeading`=`true`, `false` default: `false`

`true` Require a blank line between a paragraph and the following header.

`false` Do not require a blank line between a paragraph and the following header.

```
\@@_add_lua_option:nnn
  { blankBeforeHeading }
  { boolean }
  { false }
defaultOptions.blankBeforeHeading = false
```

`bracketedSpans`=`true`, `false` default: `false`

`true` Enable the Pandoc bracketed spans extension:

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

`false` Disable the Pandoc bracketed spans extension:

```
\@@_add_lua_option:nnn
  { bracketedSpans }
  { boolean }
  { false }
defaultOptions.bracketedSpans = false
```

`breakableBlockquotes`=`true`, `false` default: `false`

`true` A blank line separates block quotes.

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

```
\@@_add_lua_option:nnn
  { breakableBlockquotes }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { citationNbsps }
  { boolean }
  { true }
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.

```
\@@_add_lua_option:nnn
  { citations }
  { boolean }
  { false }
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.''
```

```
\@@_add_lua_option:nnn
  { codeSpans }
  { boolean }
  { true }
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.

```
\@@_add_lua_option:nnn
  { contentBlocks }
  { boolean }
  { false }
defaultOptions.contentBlocks = false
```

`debugExtensions`=true, false default: `false`

`true` Produce a JSON file that will contain the extensible subset of the PEG grammar of markdown (see the `walkable_syntax` hash table) after built-in syntax extensions (see Section 3.1.6) and user-defined syntax extensions (see Section 2.1.2) have been applied. This helps you to see how the different extensions interact. The name of the produced JSON file is controlled by the `debugExtensionsFileName` option.

`false` Do not produce a JSON file with the PEG grammar of markdown.

```
\@@_add_lua_option:nnn
  { debugExtensions }
  { boolean }
  { false }
defaultOptions.debugExtensions = false
```

`definitionLists`=true, false default: `false`

`true` Enable the pandoc definition list syntax extension:

```
Term 1

:   Definition 1
```

```
Term 2 with *inline markup*

:   Definition 2

        { some code, part of Definition 2 }

    Third paragraph of definition 2.
```

`false` Disable the pandoc definition list syntax extension.

```
\@@_add_lua_option:nnn
  { definitionLists }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { eagerCache }
  { boolean }
  { true }
defaultOptions.eagerCache = true
```

`expectJekyllData=true, false` default: `false`

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

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

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

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

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

```
\@@_add_lua_option:nnn
  { expectJekyllData }
  { boolean }
  { false }
defaultOptions.expectJekyllData = false
```

`extensions`=⟨*filenames*⟩

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

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

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

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

    reader.insert_pattern("Inline after Emph", read_strike_through,
                          "StrikeThrough")
    reader.add_special_character("/")
  end
}
```

```
return strike_through
```

The `api_version` and `grammar_version` fields specify the version of the user-defined syntax extension API and the markdown grammar for which the extension was written. See the current API and grammar versions below:

```
metadata.user_extension_api_version = 2
metadata.grammar_version = 2
```

```
Any changes to the syntax extension API or grammar will cause the
corresponding current version to be incremented.  After Markdown 3.0.0,
any changes to the API and the grammar will be either backwards-compatible
or constitute a breaking change that will cause the major version of the
Markdown package to increment (to 4.0.0).

The `finalize_grammar` field is a function that finalizes the grammar of
markdown using the interface of a Lua \luamref{reader} object, such as
the \luamref{reader->insert_pattern} and
\luamref{reader->add_special_character} methods,
see Section <#luauserextensions>.
```

```
\cs_generate_variant:Nn
  \@@_add_lua_option:nnn
  { nnV }
\@@_add_lua_option:nnV
  { extensions }
  { clist }
  \c_empty_clist
defaultOptions.extensions = {}
```

`fancyLists`=`true`, `false` default: `false`

`true` Enable the Pandoc fancy list extension:

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

`false` Disable the Pandoc fancy list extension.

```
\@@_add_lua_option:nnn
  { fancyLists }
  { boolean }
  { false }
```

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

`fencedCode`=true, false default: `false`

`true` Enable the commonmark fenced code block extension:

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

  ``` html
  <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.

```
438 \@@_add_lua_option:nnn
439   { fencedCode }
440   { boolean }
441   { false }
442 defaultOptions.fencedCode = false
```

`fencedCodeAttributes`=true, false default: `false`

`true` Enable the Pandoc fenced code attribute extension:

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

`false` Disable the Pandoc fenced code attribute extension.

```
\@@_add_lua_option:nnn
  { fencedCodeAttributes }
  { boolean }
  { false }
defaultOptions.fencedCodeAttributes = false
```

`fencedDivs`=`true`, `false` default: `false`

`true` Enable the Pandoc fenced divs extension:

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

And another.
:::::
```

`false` Disable the Pandoc fenced divs extension:

```
\@@_add_lua_option:nnn
  { fencedDivs }
  { boolean }
  { false }
defaultOptions.fencedDivs = false
```

`finalizeCache`=`true`, `false` default: `false`

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

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

```
\@@_add_lua_option:nnn
  { finalizeCache }
  { boolean }
  { false }
defaultOptions.finalizeCache = false
```

`frozenCacheCounter`=⟨*number*⟩ default: `0`

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

Each frozen cache entry will define a TEX macro `\markdownFrozenCache`⟨*number*⟩ that will typeset markdown document number ⟨*number*⟩.

```
\@@_add_lua_option:nnn
  { frozenCacheCounter }
  { counter }
  { 0 }
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.

```
\@@_add_lua_option:nnn
  { hardLineBreaks }
  { boolean }
  { false }
defaultOptions.hardLineBreaks = false
```

`hashEnumerators`=`true`, `false` default: `false`

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

```
#. Bird
#. McHale
#. Parish
```

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

```
\@@_add_lua_option:nnn
  { hashEnumerators }
  { boolean }
  { false }
defaultOptions.hashEnumerators = false
```

`headerAttributes`=true, false default: `false`

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

```
# My first heading {#foo}

## My second heading ##    {#bar .baz}

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

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

```
\@@_add_lua_option:nnn
  { headerAttributes }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { html }
  { boolean }
  { false }
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 interpretted by TEX. This is encouraged when typesetting automatically generated content or markdown documents that were not prepared with this package in mind.

```
\@@_add_lua_option:nnn
  { hybrid }
  { boolean }
  { false }
defaultOptions.hybrid = false
```

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

`true` Enable the Pandoc inline note syntax extension:

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

`false` Disable the Pandoc inline note syntax extension.

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

```
\@@_add_lua_option:nnn
  { inlineFootnotes }
  { boolean }
  { false }
\@@_add_lua_option:nnn
  { inlineNotes }
  { boolean }
  { false }
defaultOptions.inlineFootnotes = false
defaultOptions.inlineNotes = false
```

`jekyllData`=`true`, `false` default: `false`

`true` Enable the Pandoc `yaml_metadata_block` syntax extension for entering metadata in YAML:

```
---
title:  'This is the title: it contains a colon'
author:
- Author One
```

```
- Author Two
keywords: [nothing, nothingness]
abstract: |
  This is the abstract.

  It consists of two paragraphs.
---
```

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

```
\@@_add_lua_option:nnn
  { jekyllData }
  { boolean }
  { false }
defaultOptions.jekyllData = false
```

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

`true` Enable the Pandoc line block syntax extension.

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

`false` Disable the Pandoc line block syntax extension.

```
\@@_add_lua_option:nnn
  { lineBlocks }
  { boolean }
  { false }
defaultOptions.lineBlocks = false
```

`notes`=`true`, `false` default: `false`

`true` Enable the Pandoc note syntax extension:

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

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

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

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

        { some.code }

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

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

`false` Disable the Pandoc note syntax extension.

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

```
\@@_add_lua_option:nnn
  { footnotes }
  { boolean }
  { false }
\@@_add_lua_option:nnn
  { notes }
  { boolean }
  { false }
defaultOptions.footnotes = false
defaultOptions.notes = false
```

`pipeTables`=`true`, `false` default: `false`

`true` Enable the PHP Markdown pipe table syntax extension:

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

`false` Disable the PHP Markdown pipe table syntax extension.

```
\@@_add_lua_option:nnn
  { pipeTables }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { preserveTabs }
  { boolean }
  { false }
defaultOptions.preserveTabs = false
```

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

`true` Enable the Pandoc raw attribute syntax extension:

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

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

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

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

`false` Disable the Pandoc raw attribute syntax extension.

```
\@@_add_lua_option:nnn
  { rawAttribute }
  { boolean }
  { false }
defaultOptions.rawAttribute = true
```

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

`true` Enable relative references[6] 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.

```
\@@_add_lua_option:nnn
  { relativeReferences }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { shiftHeadings }
  { number }
  { 0 }
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).

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

- `$`⟨*identifier*⟩ selects the end of a section with the HTML attribute `#`⟨*identifier*⟩.
- ⟨*identifier*⟩ corresponds to `^`⟨*identifier*⟩ for the first selector and to `$`⟨*identifier*⟩ for the second selector.

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

```
\@@_add_lua_option:nnn
  { slice }
  { slice }
  { ^~$ }
defaultOptions.slice = "^ $"
```

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

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

`false` Preserve all ellipses in the input.

```
\@@_add_lua_option:nnn
  { smartEllipses }
  { boolean }
  { false }
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 `\markdownRendererOlItemWithNumber` TEX macro.

`false` Ignore the numbers in the ordered list items. Each item will only produce a `\markdownRendererOlItem` TEX macro.

```
\@@_add_lua_option:nnn
  { startNumber }
  { boolean }
  { true }
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.

```
\@@_add_lua_option:nnn
  { strikeThrough }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { stripIndent }
  { boolean }
  { false }
defaultOptions.stripIndent = false
```

`subscripts`=true, false default: `false`

`true` Enable the Pandoc subscript syntax extension:

```
H~2~O is a liquid.
```

`false` Disable the Pandoc subscript syntax extension.

```
\@@_add_lua_option:nnn
  { subscripts }
  { boolean }
  { false }
defaultOptions.subscripts = false
```

`superscripts`=true, false default: `false`

`true` Enable the Pandoc superscript syntax extension:

```
2^10^ is 1024.
```

`false` Disable the Pandoc superscript syntax extension.

```
\@@_add_lua_option:nnn
  { superscripts }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { tableCaptions }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { taskLists }
  { boolean }
  { false }
defaultOptions.taskLists = false
```

`texComments`=`true`, `false` default: `false`

`true` Strip TeX-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 TeX-style comments.

```
\@@_add_lua_option:nnn
  { texComments }
  { boolean }
  { false }
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.

```
\@@_add_lua_option:nnn
  { tightLists }
  { boolean }
  { true }
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.

```
\@@_add_lua_option:nnn
  { underscores }
  { boolean }
  { true }
\ExplSyntaxOff
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.

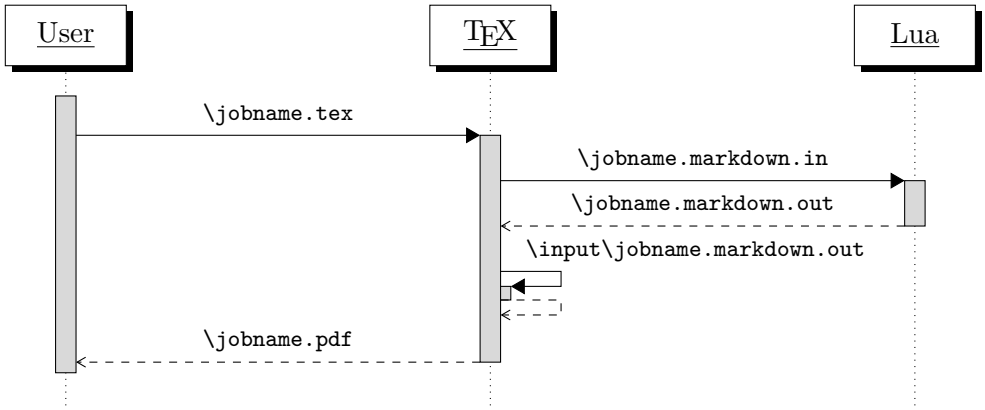

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

```
604
605 local HELP_STRING = [[
606 Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
607 where OPTIONS are documented in the Lua interface section of the
608 technical Markdown package documentation.
609
610 When OUTPUT_FILE is unspecified, the result of the conversion will be
611 written to the standard output. When INPUT_FILE is also unspecified, the
612 result of the conversion will be read from the standard input.
613
614 Report bugs to: witiko@mail.muni.cz
615 Markdown package home page: <https://github.com/witiko/markdown>]]
616
```

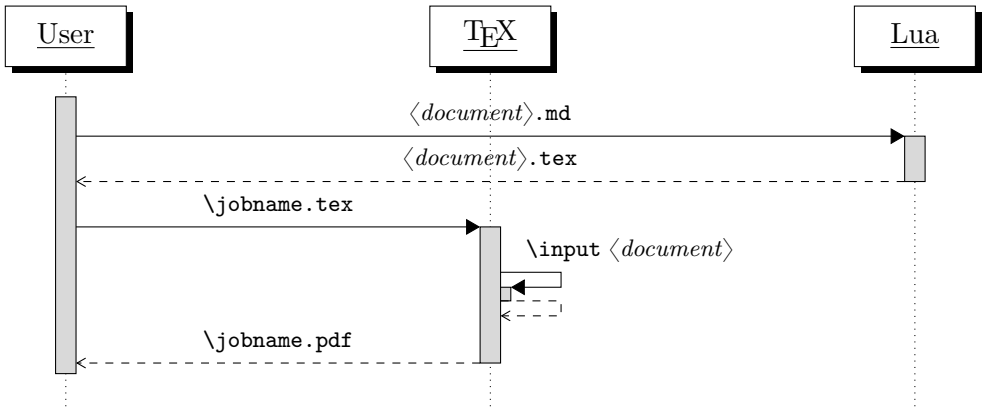

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```
local VERSION_STRING = [[
markdown-cli.lua (Markdown) ]] .. metadata.version .. [[

Copyright (C) ]] .. table.concat(metadata.copyright,
                                 "\nCopyright (C) ") .. [[

License: ]] .. metadata.license

local function warn(s)
  io.stderr:write("Warning: " .. s .. "\n") end

local function error(s)
  io.stderr:write("Error: " .. s .. "\n")
  os.exit(1)
end
```

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

```
local function camel_case(option_name)
  local cased_option_name = option_name:gsub("_(%l)", function(match)
    return match:sub(2, 2):upper()
  end)
  return cased_option_name
end

local function snake_case(option_name)
  local cased_option_name = option_name:gsub("%l%u", function(match)
    return match:sub(1, 1) .. "_" .. match:sub(2, 2):lower()
```

```
  end)
  return cased_option_name
end

local cases = {camel_case, snake_case}
local various_case_options = {}
for option_name, _ in pairs(defaultOptions) do
  for _, case in ipairs(cases) do
    various_case_options[case(option_name)] = option_name
  end
end

local process_options = true
local options = {}
local input_filename
local output_filename
for i = 1, #arg do
  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.

```
    if arg[i] == "--" then
      process_options = false
      goto continue
```

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

```
    elseif arg[i]:match("=") then
      local key, value = arg[i]:match("(.-)=(.*)")
      if defaultOptions[key] == nil then
        key = various_case_options[key]
      end
```

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

```
      local default_type = type(defaultOptions[key])
      if default_type == "boolean" then
        options[key] = (value == "true")
      elseif default_type == "number" then
        options[key] = tonumber(value)
      elseif default_type == "table" then
        options[key] = {}
        for item in value:gmatch("[^ ,]+") do
          table.insert(options[key], item)
        end
```

```
      else
        if default_type ~= "string" then
          if default_type == "nil" then
            warn('Option "' .. key .. '" not recognized.')
          else
            warn('Option "' .. key .. '" type not recognized, please file ' ..
                 'a report to the package maintainer.')
          end
          warn('Parsing the ' .. 'value "' .. value ..'" of option "' ..
               key .. '" as a string.')
        end
        options[key] = value
      end
      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.

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

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

The first argument that matches none of the above patters is assumed to be the input filename. The input filename should correspond to the Markdown document that is going to be converted to a TEX document.

```
  if input_filename == nil then
    input_filename = arg[i]
```

The first argument that matches none of the above patters is assumed to be the output filename. The output filename should correspond to the TEX document that will result from the conversion.

```
  elseif output_filename == nil then
    output_filename = arg[i]
  else
    error('Unexpected argument: "' .. arg[i] .. '".')
  end
  ::continue::
end
```

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

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

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

```
\input hello
```

2.2 Plain TEX Interface

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

```
709 \def\markdownLastModified{(((LASTMODIFIED)))}%
710 \def\markdownVersion{(((VERSION)))}%
```

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

```
\input markdown
```

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

2.2.1 Typesetting Markdown

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

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

```
711 \let\markdownBegin\relax
712 \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 corrolary, the `\markdownEnd` string may not appear anywhere inside the markdown input.

Another limitation concerns spaces at the right end of an input line. In markdown, these are used to produce a forced line break. However, any such spaces are removed before the lines enter the input buffer of TEX [6, p. 46]. As a corrolary, the `\markdownBegin` macro also ignores them.

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

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

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

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

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

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

```
713 \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
```

The `\markdownEscape` macro accepts a single parameter with the filename of a TEX document and executes the TEX document in the middle of a markdown document

fragment. Unlike the `\input` built-in of TEX, `\markdownEscape` guarantees that the standard catcode regime of your TEX format will be used.

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

2.2.2 Options

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

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

```
\ExplSyntaxOn
\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.

```
\prop_new:N \g_@@_plain_tex_option_types_prop
\prop_new:N \g_@@_default_plain_tex_options_prop
\tl_const:Nn \c_@@_option_layer_plain_tex_tl { plain_tex }
\seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_plain_tex_tl
\cs_new:Nn
  \@@_add_plain_tex_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_plain_tex_tl
      { #1 }
      { #2 }
      { #3 }
  }
```

2.2.2.1 Finalizing and Freezing the Cache

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

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

```
\@@_add_plain_tex_option:nnn
  { frozenCache }
  { boolean }
```

```
733   { false }
```

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

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

2.2.2.2 File and Directory Names

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

```
734 \@@_add_plain_tex_option:nnn
735   { helperScriptFileName }
736   { path }
737   { \jobname.markdown.lua }
```

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

```
738 \str_new:N
739   \g_luabridge_helper_script_filename_str
740 \tl_gset:Nn
741   \g_luabridge_helper_script_filename_str
742   { \markdownOptionHelperScriptFileName }
```

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

```
743 \@@_add_plain_tex_option:nnn
744   { inputTempFileName }
745   { path }
746   { \jobname.markdown.in }
```

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

```
747 \@@_add_plain_tex_option:nnn
748   { outputTempFileName }
```

```
  { path }
  { \jobname.markdown.out }
```

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

```
\str_new:N
  \g_luabridge_standard_output_filename_str
\tl_gset:Nn
  \g_luabridge_standard_output_filename_str
  { \markdownOptionOutputTempFileName }
```

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

```
\@@_add_plain_tex_option:nnn
  { errorTempFileName }
  { path }
  { \jobname.markdown.err }
```

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

```
\str_new:N
  \g_luabridge_error_output_filename_str
\tl_gset:Nn
  \g_luabridge_error_output_filename_str
  { \markdownOptionErrorTempFileName }
```

The `\markdownOptionOutputDir` macro sets the path to the directory that will contain the auxiliary cache files produced by the Lua implementation and also the auxiliary files produced by the plain TEX implementation. The option defaults to `.`.

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

```
\@@_add_plain_tex_option:nnn
  { outputDir }
  { path }
  { . }
```

Here, we automatically define plain TEX macros for the above plain TEX options.

Furthemore, we also define macros that map directly to the options recognized by the Lua interface, such as `\markdownOptionHybrid` for the `hybrid` Lua option (see Section 2.1.3), which are not processed by the plain TEX implementation, only passed along to Lua.

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

```
\cs_new:Nn \@@_plain_tex_define_option_commands:
  {
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
              \@@_plain_tex_define_option_command:n
                { ####1 }
          }
      }
  }
\cs_new:Nn \@@_plain_tex_define_option_command:n
  {
    \@@_get_default_option_value:nN
      { #1 }
      \l_tmpa_tl
    \@@_set_option_value:nV
      { #1 }
      \l_tmpa_tl
  }
\cs_new:Nn
  \@@_set_option_value:nn
  {
    \@@_define_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNTF
      \c_@@_option_type_counter_tl
      \l_tmpa_tl
      {
        \@@_option_tl_to_csname:nN
          { #1 }
          \l_tmpa_tl
        \int_gset:cn
          { \l_tmpa_tl }
          { #2 }
      }
      {
        \@@_option_tl_to_csname:nN
          { #1 }
```

```
          \l_tmpa_tl
        \cs_set:cpn
          { \l_tmpa_tl }
          { #2 }
      }
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\cs_new:Nn
  \@@_define_option:n
  {
    \@@_option_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \cs_if_free:cT
      { \l_tmpa_tl }
      {
        \@@_get_option_type:nN
          { #1 }
          \l_tmpb_tl
        \str_if_eq:NNT
          \c_@@_option_type_counter_tl
          \l_tmpb_tl
          {
            \@@_option_tl_to_csname:nN
              { #1 }
              \l_tmpa_tl
            \int_new:c
              { \l_tmpa_tl }
          }
      }
  }
\@@_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 TEX package documentation using the Doc LATEX package [7] or similar. The recognized values of the macro are `true` (discard) and `false` (retain). It defaults to `false`.

```
\seq_gput_right:Nn
  \g_@@_plain_tex_options_seq
  { stripPercentSigns }
\prop_gput:Nnn
  \g_@@_plain_tex_option_types_prop
```

```
  { stripPercentSigns }
  { boolean }
\prop_gput:Nnx
  \g_@@_default_plain_tex_options_prop
  { stripPercentSigns }
  { false }
\ExplSyntaxOff
```

2.2.3 Token Renderers

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

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

```
\ExplSyntaxOn
\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.

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

2.2.3.1 Attribute Renderers

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

`\markdownRendererAttributeIdentifier` represents the ⟨*identifier*⟩ of a markdown element (`id="`⟨*identifier*⟩`"` in HTML and `#`⟨*identifier*⟩ in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the ⟨*identifier*⟩.

`\markdownRendererAttributeClassName` represents the ⟨*class name*⟩ of a markdown element (`class="`⟨*class name*⟩ ...`"` in HTML and `.`⟨*class name*⟩ in Markdown's `headerAttributes` syntax extension). The macro receives a single attribute that corresponds to the ⟨*class name*⟩.

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

```
\def\markdownRendererAttributeIdentifier{%
  \markdownRendererAttributeIdentifierPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { attributeIdentifier }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { attributeIdentifier }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererAttributeClassName{%
  \markdownRendererAttributeClassNamePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { attributeClassName }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { attributeClassName }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererAttributeKeyValue{%
  \markdownRendererAttributeKeyValuePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { attributeKeyValue }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { attributeKeyValue }
  { 2 }
\ExplSyntaxOff
```

2.2.3.2 Block Quote Renderers

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

```
\def\markdownRendererBlockQuoteBegin{%
  \markdownRendererBlockQuoteBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockQuoteBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockQuoteBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererBlockQuoteEnd{%
  \markdownRendererBlockQuoteEndPrototype}%
\ExplSyntaxOn
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockQuoteEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockQuoteEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.3 Bracketed Spans Attribute Context Renderers

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

The `\markdownRendererBracketedSpanAttributeContextBegin` and `\markdownRendererBrac` macros represent the beginning and the end of an inline bracketed span in which the attributes of the span apply. The macros receive no arguments.

```
\def\markdownRendererBracketedSpanAttributeContextBegin{%
  \markdownRendererBracketedSpanAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { bracketedSpanAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpanAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererBracketedSpanAttributeContextEnd{%
  \markdownRendererBracketedSpanAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { bracketedSpanAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { bracketedSpanAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.4 Bullet List Renderers

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

```
\def\markdownRendererUlBegin{%
  \markdownRendererUlBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
```

```
  { ulBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulBegin }
  { 0 }
\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.

```
\def\markdownRendererUlBeginTight{%
  \markdownRendererUlBeginTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulBeginTight }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererUlItem{%
  \markdownRendererUlItemPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulItem }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererUlItemEnd{%
  \markdownRendererUlItemEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulItemEnd }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { ulItemEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererUlEnd 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.

```
\def\markdownRendererUlEnd{%
  \markdownRendererUlEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulEnd }
  { 0 }
\ExplSyntaxOff
```

The \markdownRendererUlEndTight 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.

```
\def\markdownRendererUlEndTight{%
  \markdownRendererUlEndTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ulEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ulEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.3.5 Code Block Renderers

The \markdownRendererInputVerbatim macro represents a code block. The macro receives a single argument that corresponds to the filename of a file contaning the code block contents.

```
\def\markdownRendererInputVerbatim{%
  \markdownRendererInputVerbatimPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputVerbatim }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputVerbatim }
  { 1 }
\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 contaning the code block contents and to the code fence infostring.

```
\def\markdownRendererInputFencedCode{%
  \markdownRendererInputFencedCodePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputFencedCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputFencedCode }
  { 2 }
\ExplSyntaxOff
```

2.2.3.6 Code Span Renderer

The `\markdownRendererCodeSpan` macro represents inline code span in the input text. It receives a single argument that corresponds to the inline code span.

```
\def\markdownRendererCodeSpan{%
  \markdownRendererCodeSpanPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { codeSpan }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { codeSpan }
  { 1 }
\ExplSyntaxOff
```

2.2.3.7 Content Block Renderers

The `\markdownRendererContentBlock` macro represents an iA,Writer content block. It receives four arguments: the local file or online image filename extension cast to the lower case, the fully escaped URI that can be directly typeset, the raw URI that can be used outside typesetting, and the title of the content block.

```
\def\markdownRendererContentBlock{%
  \markdownRendererContentBlockPrototype}%
```

```
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlock }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlock }
  { 4 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererContentBlockOnlineImage{%
  \markdownRendererContentBlockOnlineImagePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockOnlineImage }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockOnlineImage }
  { 4 }
\ExplSyntaxOff
```

The `\markdownRendererContentBlockCode` macro represents an iA,Writer content block that was recognized as a file in a known programming language by its filename extension s. If any `markdown-languages.json` file found by kpathsea[7] contains a record (k, v), then a non-online-image content block with the filename extension s, `s: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 place a `markdown-languages.json` file inside your working directory or inside your local TEX directory structure. In this file, you will define a mapping between filename extensions and the language names recognized by your favorite syntax highlighter; there may exist other creative uses beside syntax highlighting. The `Languages.json` file provided by Sotkov [3] is a good starting point.

```
\def\markdownRendererContentBlockCode{%
  \markdownRendererContentBlockCodePrototype}%
\ExplSyntaxOn
```

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

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockCode }
  { 5 }
\ExplSyntaxOff
```

2.2.3.8 Definition List Renderers

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

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

```
\def\markdownRendererDlBegin{%
  \markdownRendererDlBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlBegin }
  { 0 }
\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.

```
\def\markdownRendererDlBeginTight{%
  \markdownRendererDlBeginTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlBeginTight }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererDlItem{%
```

```
  \markdownRendererDlItemPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlItem }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererDlItemEnd{%
  \markdownRendererDlItemEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlItemEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererDlDefinitionBegin{%
  \markdownRendererDlDefinitionBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlDefinitionBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlDefinitionBegin }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererDlDefinitionEnd{%
  \markdownRendererDlDefinitionEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlDefinitionEnd }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlDefinitionEnd }
  { 0 }
\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.

```
\def\markdownRendererDlEnd{%
  \markdownRendererDlEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlEnd }
  { 0 }
\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.

```
\def\markdownRendererDlEndTight{%
  \markdownRendererDlEndTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dlEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.3.9 Ellipsis Renderer

The \markdownRendererEllipsis macro replaces any occurance of ASCII ellipses in the input text. This macro will only be produced, when the smartEllipses option is enabled. The macro receives no arguments.

```
\def\markdownRendererEllipsis{%
  \markdownRendererEllipsisPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
```

```
  { ellipsis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ellipsis }
  { 0 }
\ExplSyntaxOff
```

2.2.3.10 Emphasis Renderers

The `\markdownRendererEmphasis` macro represents an emphasized span of text. The macro receives a single argument that corresponds to the emphasized span of text.

```
\def\markdownRendererEmphasis{%
  \markdownRendererEmphasisPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { emphasis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { emphasis }
  { 1 }
\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.

```
\def\markdownRendererStrongEmphasis{%
  \markdownRendererStrongEmphasisPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { strongEmphasis }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { strongEmphasis }
  { 1 }
\ExplSyntaxOff
```

2.2.3.11 Fenced Code Attribute Context Renderers

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

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

```
\def\markdownRendererFencedCodeAttributeContextBegin{%
  \markdownRendererFencedCodeAttributeContextBeginPrototype}%
```

```
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedCodeAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedCodeAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererFencedCodeAttributeContextEnd{%
  \markdownRendererFencedCodeAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedCodeAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedCodeAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.12 Fenced Div Attribute Context Renderers

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

The `\markdownRendererFencedDivAttributeContextBegin` and `\markdownRendererFencedDiv` macros represent the beginning and the end of a div in which the attributes of the div apply. The macros receive no arguments.

```
\def\markdownRendererFencedDivAttributeContextBegin{%
  \markdownRendererFencedDivAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedDivAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedDivAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererFencedDivAttributeContextEnd{%
  \markdownRendererFencedDivAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fencedDivAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fencedDivAttributeContextEnd }
```

```
  { 0 }
\ExplSyntaxOff
```

2.2.3.13 Header Attribute Context Renderers

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

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

```
\def\markdownRendererHeaderAttributeContextBegin{%
  \markdownRendererHeaderAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHeaderAttributeContextEnd{%
  \markdownRendererHeaderAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headerAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headerAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.14 Heading Renderers

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

```
\def\markdownRendererHeadingOne{%
  \markdownRendererHeadingOnePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingOne }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingOne }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererHeadingTwo{%
  \markdownRendererHeadingTwoPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingTwo }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingTwo }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererHeadingThree{%
  \markdownRendererHeadingThreePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingThree }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingThree }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererHeadingFour{%
  \markdownRendererHeadingFourPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingFour }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingFour }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererHeadingFive{%
  \markdownRendererHeadingFivePrototype}%
```

```
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingFive }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingFive }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererHeadingSix{%
  \markdownRendererHeadingSixPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { headingSix }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { headingSix }
  { 1 }
\ExplSyntaxOff
```

2.2.3.15 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 \markdownRendererBlockHtmlCommentEn macros represent the beginning and the end of a block HTML comment. The macros receive no arguments.

```
\def\markdownRendererInlineHtmlComment{%
  \markdownRendererInlineHtmlCommentPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlComment }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlComment }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererBlockHtmlCommentBegin{%
  \markdownRendererBlockHtmlCommentBeginPrototype}%
\ExplSyntaxOn
```

```
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockHtmlCommentBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockHtmlCommentBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererBlockHtmlCommentEnd{%
  \markdownRendererBlockHtmlCommentEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { blockHtmlCommentEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { blockHtmlCommentEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.16 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.

```
\def\markdownRendererInlineHtmlTag{%
  \markdownRendererInlineHtmlTagPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineHtmlTag }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineHtmlTag }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererInputBlockHtmlElement{%
  \markdownRendererInputBlockHtmlElementPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputBlockHtmlElement }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputBlockHtmlElement }
  { 1 }
\ExplSyntaxOff
```

2.2.3.17 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.

```
\def\markdownRendererImage{%
  \markdownRendererImagePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { image }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { image }
  { 4 }
\ExplSyntaxOff
```

2.2.3.18 Interblock Separator Renderer

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

```
\def\markdownRendererInterblockSeparator{%
  \markdownRendererInterblockSeparatorPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { interblockSeparator }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { interblockSeparator }
  { 0 }
\ExplSyntaxOff
```

2.2.3.19 Line Block Renderer

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

The `\markdownRendererLineBlockBegin` and `\markdownRendererLineBlockEnd` macros represent the beginning and the end of a line block. The macros receive no arguments.

```
\def\markdownRendererLineBlockBegin{%
```

```
  \markdownRendererLineBlockBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { lineBlockBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { lineBlockBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererLineBlockEnd{%
  \markdownRendererLineBlockEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { lineBlockEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { lineBlockEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.20 Line Break Renderer

The `\markdownRendererLineBreak` macro represents a forced line break. The macro receives no arguments.

```
\def\markdownRendererLineBreak{%
  \markdownRendererLineBreakPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { lineBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { lineBreak }
  { 0 }
\ExplSyntaxOff
```

2.2.3.21 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.

```
\def\markdownRendererLink{%
  \markdownRendererLinkPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { link }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { link }
  { 4 }
\ExplSyntaxOff
```

2.2.3.22 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.

```
\def\markdownRendererDocumentBegin{%
  \markdownRendererDocumentBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { documentBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { documentBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererDocumentEnd{%
  \markdownRendererDocumentEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { documentEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { documentEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.23 Non-Breaking Space Renderer

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

```
\def\markdownRendererNbsp{%
  \markdownRendererNbspPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { nbsp }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { nbsp }
  { 0 }
\ExplSyntaxOff
```

2.2.3.24 Note Renderer

The `\markdownRendererNote` macro represents a note. This macro will only be produced, when the `notes` option is enabled. The macro receives a single argument that corresponds to the note text.

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

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownRendererNote
  {
    \cs_if_exist:NTF
      \markdownRendererFootnote
      {
        \markdownWarning
          {
            Footnote~renderer~has~been~deprecated,~
            to~be~removed~in~Markdown~3.0.0
          }
        \markdownRendererFootnote
      }
      {
        \cs_if_exist:NTF
          \markdownRendererFootnotePrototype
          {
            \markdownWarning
              {
                Footnote~renderer~prototype~has~been~deprecated,~
                to~be~removed~in~Markdown~3.0.0
              }
            \markdownRendererFootnotePrototype
          }
          {
            \markdownRendererNotePrototype
          }
      }
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { footnote }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
```

```
  { footnote }
  { 1 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { note }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { note }
  { 1 }
\ExplSyntaxOff
```

2.2.3.25 Ordered List Renderers

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

```
\def\markdownRendererOlBegin{%
  \markdownRendererOlBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olBegin }
  { 0 }
\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.

```
\def\markdownRendererOlBeginTight{%
  \markdownRendererOlBeginTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olBeginTight }
  { 0 }
\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`).

```
\def\markdownRendererFancyOlBegin{%
  \markdownRendererFancyOlBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlBegin }
  { 2 }
\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.

```
\def\markdownRendererFancyOlBeginTight{%
  \markdownRendererFancyOlBeginTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlBeginTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlBeginTight }
  { 2 }
\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.

```
\def\markdownRendererOlItem{%
  \markdownRendererOlItemPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olItem }
```

```
  { 0 }
\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.

```
\def\markdownRendererOlItemEnd{%
  \markdownRendererOlItemEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olItemEnd }
  { 0 }
\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.

```
\def\markdownRendererOlItemWithNumber{%
  \markdownRendererOlItemWithNumberPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olItemWithNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olItemWithNumber }
  { 1 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererFancyOlItem{%
  \markdownRendererFancyOlItemPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlItem }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItem }
```

```
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererFancyOlItemEnd{%
  \markdownRendererFancyOlItemEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlItemEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItemEnd }
  { 0 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererFancyOlItemWithNumber{%
  \markdownRendererFancyOlItemWithNumberPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlItemWithNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlItemWithNumber }
  { 1 }
\ExplSyntaxOff
```

The `\markdownRendererOlEnd` 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.

```
\def\markdownRendererOlEnd{%
  \markdownRendererOlEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
```

```
  { olEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererOlEndTight` 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.

```
\def\markdownRendererOlEndTight{%
  \markdownRendererOlEndTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { olEndTight }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { olEndTight }
  { 0 }
\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.

```
\def\markdownRendererFancyOlEnd{%
  \markdownRendererFancyOlEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEnd }
  { 0 }
\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.

```
\def\markdownRendererFancyOlEndTight{%
  \markdownRendererFancyOlEndTightPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { fancyOlEndTight }
```

```
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { fancyOlEndTight }
  { 0 }
\ExplSyntaxOff
```

2.2.3.26 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.

```
\def\markdownRendererCite{%
  \markdownRendererCitePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { cite }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { cite }
  { 1 }
\ExplSyntaxOff
```

2.2.3.27 Raw Content Renderers

The `\markdownRendererInputRawInline` macro represents an inline raw span. The macro receives two arguments: the filename of a file contaning the inline raw span contents and the raw attribute that designates the format of the inline raw span. This macro will only be produced, when the `rawAttribute` option is enabled.

```
\def\markdownRendererInputRawInline{%
  \markdownRendererInputRawInlinePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputRawInline }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputRawInline }
  { 2 }
\ExplSyntaxOff
```

The `\markdownRendererInputRawBlock` macro represents a raw block. The macro receives two arguments: the filename of a file contaning the raw block and the raw

attribute that designates the format of the raw block. This macro will only be produced, when the `rawAttribute` and `fencedCode` options are enabled.

```
\def\markdownRendererInputRawBlock{%
  \markdownRendererInputRawBlockPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inputRawBlock }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inputRawBlock }
  { 2 }
\ExplSyntaxOff
```

2.2.3.28 Replacement Character Renderers

The `\markdownRendererReplacementCharacter` macro represents the U+0000 and U+FFFD Unicode characters. The macro receives no arguments.

```
\def\markdownRendererReplacementCharacter{%
  \markdownRendererReplacementCharacterPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { replacementCharacter }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { replacementCharacter }
  { 0 }
\ExplSyntaxOff
```

2.2.3.29 Special Character Renderers

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

```
\def\markdownRendererLeftBrace{%
  \markdownRendererLeftBracePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { leftBrace }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { leftBrace }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererRightBrace{%
```

```
  \markdownRendererRightBracePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { rightBrace }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { rightBrace }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererDollarSign{%
  \markdownRendererDollarSignPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { dollarSign }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { dollarSign }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererPercentSign{%
  \markdownRendererPercentSignPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { percentSign }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { percentSign }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererAmpersand{%
  \markdownRendererAmpersandPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { ampersand }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { ampersand }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererUnderscore{%
  \markdownRendererUnderscorePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { underscore }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { underscore }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHash{%
  \markdownRendererHashPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { hash }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { hash }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererCircumflex{%
  \markdownRendererCircumflexPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { circumflex }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { circumflex }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererBackslash{%
  \markdownRendererBackslashPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { backslash }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { backslash }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererTilde{%
  \markdownRendererTildePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tilde }
\prop_gput:Nnn
```

```
  \g_@@_renderer_arities_prop
  { tilde }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererPipe{%
  \markdownRendererPipePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { pipe }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { pipe }
  { 0 }
\ExplSyntaxOff
```

2.2.3.30 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.

```
\def\markdownRendererStrikeThrough{%
  \markdownRendererStrikeThroughPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { strikeThrough }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { strikeThrough }
  { 1 }
\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.

```
\def\markdownRendererSubscript{%
  \markdownRendererSubscriptPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { subscript }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { subscript }
```

```
  { 1 }
```

2.2.3.32 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.

```
\def\markdownRendererSuperscript{%
  \markdownRendererSuperscriptPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { superscript }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { superscript }
  { 1 }
\ExplSyntaxOff
```

2.2.3.33 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.

```
\def\markdownRendererTable{%
  \markdownRendererTablePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { table }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { table }
  { 3 }
\ExplSyntaxOff
```

2.2.3.34 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.

```
\def\markdownRendererTextCite{%
  \markdownRendererTextCitePrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { textCite }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { textCite }
  { 1 }
\ExplSyntaxOff
```

2.2.3.35 Thematic Break Renderer

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

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

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownRendererThematicBreak
  {
    \cs_if_exist:NTF
      \markdownRendererHorizontalRule
      {
        \markdownWarning
          {
            Horizontal~rule~renderer~has~been~deprecated,~
            to~be~removed~in~Markdown~3.0.0
          }
        \markdownRendererHorizontalRule
      }
      {
        \cs_if_exist:NTF
          \markdownRendererHorizontalRulePrototype
          {
            \markdownWarning
              {
                Horizontal~rule~renderer~prototype~has~been~deprecated,~
                to~be~removed~in~Markdown~3.0.0
              }
            \markdownRendererHorizontalRulePrototype
          }
          {
            \markdownRendererThematicBreakPrototype
          }
```

```
      }
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { horizontalRule }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { horizontalRule }
  { 0 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { thematicBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { thematicBreak }
  { 0 }
\ExplSyntaxOff
```

2.2.3.36 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.

```
\def\markdownRendererTickedBox{%
  \markdownRendererTickedBoxPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { tickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { tickedBox }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererHalfTickedBox{%
  \markdownRendererHalfTickedBoxPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { halfTickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { halfTickedBox }
  { 0 }
\ExplSyntaxOff
```

```
\def\markdownRendererUntickedBox{%
  \markdownRendererUntickedBoxPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { untickedBox }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { untickedBox }
  { 0 }
\ExplSyntaxOff
```

2.2.3.37 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.

```
\def\markdownRendererJekyllDataBegin{%
  \markdownRendererJekyllDataBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataBegin }
  { 0 }
\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.

```
\def\markdownRendererJekyllDataEnd{%
  \markdownRendererJekyllDataEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererJekyllDataMappingBegin` macro represents the beginning of a mapping in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key

in the parent structure, cast to a string following YAML serialization rules, and the number of items in the mapping.

```
\def\markdownRendererJekyllDataMappingBegin{%
  \markdownRendererJekyllDataMappingBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataMappingBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataMappingBegin }
  { 2 }
\ExplSyntaxOff
```

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

```
\def\markdownRendererJekyllDataMappingEnd{%
  \markdownRendererJekyllDataMappingEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataMappingEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataMappingEnd }
  { 0 }
\ExplSyntaxOff
```

The `\markdownRendererJekyllDataSequenceBegin` macro represents the beginning of a sequence in a YAML document. This macro will only be produced when the `jekyllData` option is enabled. The macro receives two arguments: the scalar key in the parent structure, cast to a string following YAML serialization rules, and the number of items in the sequence.

```
\def\markdownRendererJekyllDataSequenceBegin{%
  \markdownRendererJekyllDataSequenceBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataSequenceBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataSequenceBegin }
  { 2 }
\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.

```
\def\markdownRendererJekyllDataSequenceEnd{%
  \markdownRendererJekyllDataSequenceEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataSequenceEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataSequenceEnd }
  { 0 }
\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.

```
\def\markdownRendererJekyllDataBoolean{%
  \markdownRendererJekyllDataBooleanPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataBoolean }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataBoolean }
  { 2 }
\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.

```
\def\markdownRendererJekyllDataNumber{%
  \markdownRendererJekyllDataNumberPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataNumber }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
```

```
  { jekyllDataNumber }
  { 2 }
\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.

```
\def\markdownRendererJekyllDataString{%
  \markdownRendererJekyllDataStringPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataString }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataString }
  { 2 }
\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.

```
\def\markdownRendererJekyllDataEmpty{%
  \markdownRendererJekyllDataEmptyPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { jekyllDataEmpty }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { jekyllDataEmpty }
  { 1 }
\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 LaTeX3 kernel.

```
\ExplSyntaxOn
\keys_define:nn
```

```
  { markdown/jekyllData }
  { }
\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).

```
\ExplSyntaxOn
\cs_new:Nn \@@_plaintex_define_renderer_prototypes:
  {
    \seq_map_function:NN
      \g_@@_renderers_seq
      \@@_plaintex_define_renderer_prototype:n
    \let\markdownRendererBlockHtmlCommentBeginPrototype=\iffalse
    \let\markdownRendererBlockHtmlCommentBegin=\iffalse
    \let\markdownRendererBlockHtmlCommentEndPrototype=\fi
    \let\markdownRendererBlockHtmlCommentEnd=\fi
```

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

```
    \cs_undefine:N \markdownRendererFootnote
    \cs_undefine:N \markdownRendererFootnotePrototype
```

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

```
    \cs_undefine:N \markdownRendererHorizontalRule
    \cs_undefine:N \markdownRendererHorizontalRulePrototype
  }
\cs_new:Nn \@@_plaintex_define_renderer_prototype:n
  {
    \@@_renderer_prototype_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_plaintex_define_renderer_prototype:cV
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_prototype_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
```

```
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
        Prototype
      }
  }
\cs_new:Nn \@@_plaintex_define_renderer_prototype:Nn
  {
    \cs_generate_from_arg_count:NNnn
      #1
      \cs_set:Npn
      { #2 }
      { }
  }
\cs_generate_variant:Nn
  \@@_plaintex_define_renderer_prototype:Nn
  { cV }
\@@_plaintex_define_renderer_prototypes:
\ExplSyntaxOff
```

2.2.5 Logging Facilities

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

2.2.6 Miscellanea

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

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

The `\markdownReadAndConvert` macro implements the `\markdownBegin` macro. The first argument specifies the token sequence that will terminate the markdown input (`\markdownEnd` in the instance of the `\markdownBegin` macro) when the plain TEX special characters have had their category changed to *other*. The second argument

specifies the token sequence that will actually be inserted into the document, when the ending token sequence has been found.

```
\let\markdownReadAndConvert\relax
\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*.

```
  \catcode`\|=0\catcode`\\=12%
  |gdef|markdownBegin{%
    |markdownReadAndConvert{\markdownEnd}%
                           {|markdownEnd}}%
|endgroup
```

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

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

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

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

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

```
\ExplSyntaxOn
\cs_if_exist:NF
  \markdownMode
  {
    \file_if_exist:nTF
      { lt3luabridge.tex }
      {
        \cs_new:Npn
          \markdownMode
          { 3 }
      }
      {
        \cs_if_exist:NTF
          \directlua
```

```
          {
            \cs_new:Npn
              \markdownMode
              { 2 }
          }
          {
            \cs_new:Npn
              \markdownMode
              { 0 }
          }
      }
  }
\ExplSyntaxOff
```

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

```
\def\markdownLuaRegisterIBCallback#1{\relax}%
\def\markdownLuaUnregisterIBCallback#1{\relax}%
```

2.3 LaTeX Interface

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

The LaTeX implementation redefines the plain TeX logging macros (see Section 3.2.1) to use the LaTeX `\PackageInfo`, `\PackageWarning`, and `\PackageError` macros.

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

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

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

where ⟨*options*⟩ are the LaTeX interface options (see Section 2.3.2). Note that ⟨*options*⟩ inside the `\usepackage` macro may not set the `markdownRenderers` (see Section 2.3.2.5) and `markdownRendererPrototypes` (see Section 2.3.2.6) keys. This limitation is due to the way LaTeX 2_ε parses package options.

2.3.1 Typesetting Markdown

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

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

```
2186 \newenvironment{markdown}\relax\relax
2187 \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` LaTeX environment (and likewise for the starred version).

Note that the `markdown` and `markdown*` LaTeX environments are subject to the same limitations as the `\markdownBegin` and `\markdownEnd` macros exposed by the plain TeX interface.

The following example LaTeX code showcases the usage of the `markdown` and `markdown*` environments:

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown}
_Hello_ **world** ...
\end{markdown}
% ...
\end{document}
```

```
\documentclass{article}
\usepackage{markdown}
\begin{document}
% ...
\begin{markdown*}{smartEllipses}
_Hello_ **world** ...
\end{markdown*}
% ...
\end{document}
```

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

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

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

2.3.2 Options

The LaTeX options are represented by a comma-delimited list of ⟨*key*⟩=⟨*value*⟩ pairs. For boolean options, the =⟨*value*⟩ part is optional, and ⟨*key*⟩ will be interpreted as ⟨*key*⟩=`true` if the =⟨*value*⟩ part has been omitted.

Except for the `plain` option described in Section 2.3.2.1, and the 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:

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_setup:n
  {
    \keys_set:nn
      { markdown/latex-options }
      { #1 }
  }
\let\markdownSetup=\@@_setup:n
\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:

```
\newcommand\markdownSetupSnippet[2]{%
  \markdownIfSnippetExists{#1}%
    {%
      \markdownWarning
        {Redefined setup snippet \markdownLaTeXThemeName#1}%
      \csname markdownLaTeXSetupSnippet%
        \markdownLaTeXThemeName#1\endcsname={#2}%
    }{%
      \newtoks\next
        \next={#2}%
      \expandafter\let\csname markdownLaTeXSetupSnippet%
        \markdownLaTeXThemeName#1\endcsname=\next
    }}%
```

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

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

```
    {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
    {#3}{#2}}%
```

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

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

```
\ExplSyntaxOn
\seq_new:N \g_@@_latex_options_seq
```

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

```
\prop_new:N \g_@@_latex_option_types_prop
\prop_new:N \g_@@_default_latex_options_prop
\tl_const:Nn \c_@@_option_layer_latex_tl { latex }
\seq_gput_right:NV \g_@@_option_layers_seq \c_@@_option_layer_latex_tl
\cs_new:Nn
  \@@_add_latex_option:nnn
  {
    \@@_add_option:Vnnn
      \c_@@_option_layer_latex_tl
      { #1 }
      { #2 }
      { #3 }
  }
```

2.3.2.1 No default token renderer prototypes

Default token renderer prototypes require LaTeX packages that may clash with other packages used in a document. Additionally, if we redefine token renderers and renderer prototypes ourselves, the default definitions will bring no benefit to us. Using the `plain` package option, we can keep the default definitions from the plain TeX implementation (see Section 3.2.2) and prevent the soft LaTeX 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}
```

```
\@@_add_latex_option:nnn
  { plain }
  { boolean }
  { false }
\ExplSyntaxOff
```

2.3.2.2 LaTeX themes

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

The LaTeX option `theme`=⟨*theme name*⟩ loads a LaTeX 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 LaTeX package, which provides similar functionality with its `\usetheme` macro [8, Section 15.1].

Theme names must be qualified to minimize naming conflicts between different themes intended for a single LaTeX document class or for a single LaTeX 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 LaTeX packages are identified only by their filenames, not by their pathnames. [9] Therefore, we can't store the qualified theme names directly using directories, but we must encode the individual segments of the qualified theme in the filename. For example, loading a theme named `witiko/beamer/MU` would load a LaTeX package named `markdownthemewitiko_beamer_MU.sty`.

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

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

```
\newif\ifmarkdownLaTeXLoaded
  \markdownLaTeXLoadedfalse
\AtEndOfPackage{\markdownLaTeXLoadedtrue}
\ExplSyntaxOn
\tl_new:N \markdownLaTeXThemePackageName
```

```
\cs_new:Nn
  \@@_set_latex_theme:n
  {
    \str_if_in:nnF
      { #1 }
      { / }
      {
        \markdownError
        { Won't~load~theme~with~unqualified~name~#1 }
        { Theme~names~must~contain~at~least~one~forward~slash }
      }
    \str_if_in:nnT
      { #1 }
      { _ }
      {
        \markdownError
        { Won't~load~theme~with~an~underscore~in~its~name~#1 }
        { Theme~names~must~not~contain~underscores~in~their~names }
      }
    \tl_set:Nn \markdownLaTeXThemePackageName { #1 }
    \str_replace_all:Nnn
      \markdownLaTeXThemePackageName
      { / }
      { _ }
    \edef\markdownLaTeXThemePackageName{
      markdowntheme\markdownLaTeXThemePackageName}
    \expandafter\markdownLaTeXThemeLoad\expandafter{
      \markdownLaTeXThemePackageName}{#1/}
  }
\keys_define:nn
  { markdown/latex-options }
  {
    theme .code:n = { \@@_set_latex_theme:n { #1 } },
  }
\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 ⟨*theme name*⟩`/` 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.

```
\ExplSyntaxOn
\@onlypreamble
  \@@_set_latex_theme:n
\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 mathemathical 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.

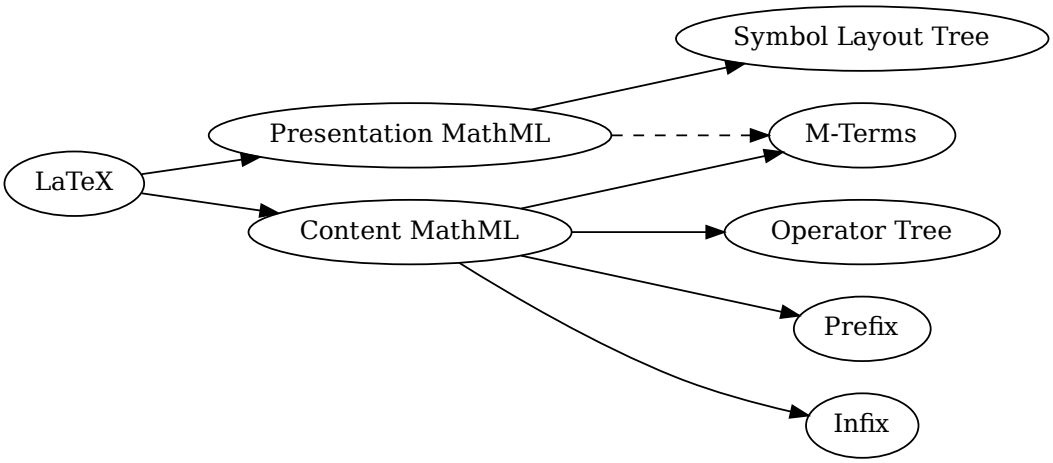

Figure 4: Various formats of mathemathical formulae

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

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

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

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

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

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

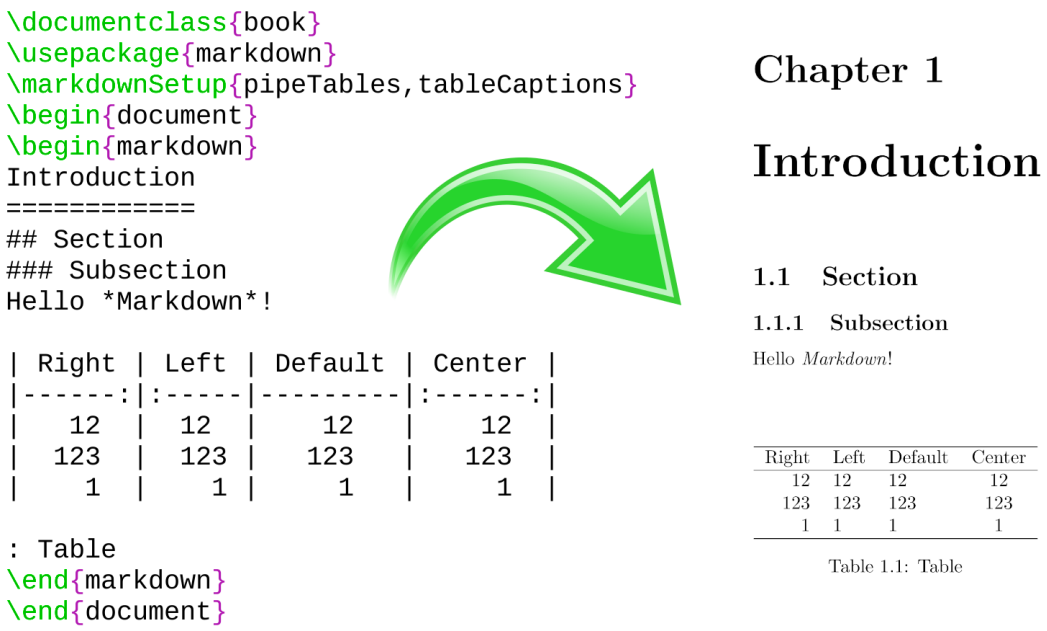

Figure 5: The banner of the Markdown package

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”.

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

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

2.3.2.3 LaTeX setup snippets

The LaTeX option with key `snippet` invokes a snippet named ⟨*value*⟩:

```
\ExplSyntaxOn
```

```
\keys_define:nn
  { markdown/latex-options }
  {
    snippet .code:n = {
      \markdownIfSnippetExists{#1}
        {
          \expandafter\markdownSetup\expandafter{
            \the\csname markdownLaTeXSetupSnippet
            \markdownLaTeXThemeName#1\endcsname}
        }{
          \markdownError
            {Can't~invoke~setup~snippet~#1}
            {The~setup~snippet~is~undefined}
        }
    }
  }
\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 TEX Interface Options

Here, we automatically define plain TEX macros and the ⟨*key*⟩=⟨*value*⟩ interface for the above LATEX options.

```
\ExplSyntaxOn
\cs_new:Nn \@@_latex_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
      \g_@@_latex_options_seq
      {
          \@@_plain_tex_define_option_command:n
            { ##1 }
      }
```

Furthermore, we also define the ⟨*key*⟩=⟨*value*⟩ interface for all option macros recognized by the Lua and plain TEX interfaces.

```
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
```

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

```
              \@@_with_various_cases:nn
                { ####1 }
                {
                  \@@_latex_define_option_keyval:nnn
                    { ##1 }
                    { ####1 }
                    { ########1 }
                }
          }
      }
  }
\cs_new:Nn \@@_latex_define_option_keyval:nnn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \keys_define:nn
      { markdown/latex-options }
      {
        #3 .code:n = {
          \@@_set_option_value:nn
            { #2 }
            { ##1 }
```

```
      },
    }
  \str_if_eq:VVT
    \l_tmpa_tl
    \c_@@_option_type_boolean_tl
    {
      \keys_define:nn
        { markdown/latex-options }
        {
          #3 .default:n = { true },
        }
    }
```

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

```
  \str_if_eq:VVT
    \l_tmpa_tl
    \c_@@_option_type_clist_tl
    {
      \tl_set:Nn
        \l_tmpa_tl
        { #3 }
      \tl_reverse:N
        \l_tmpa_tl
      \str_if_eq:enF
        {
          \tl_head:V
            \l_tmpa_tl
        }
        { s }
        {
            \msg_error:nnn
              { @@ }
              { malformed-name-for-clist-option }
              { #3 }
        }
      \tl_set:Nx
        \l_tmpa_tl
        {
          \tl_tail:V
            \l_tmpa_tl
        }
      \tl_reverse:N
        \l_tmpa_tl
```

```
        \tl_put_right:Nn
          \l_tmpa_tl
          {
            .code:n = {
              \@@_get_option_value:nN
                { #2 }
                \l_tmpa_tl
              \clist_set:NV
                \l_tmpa_clist
                { \l_tmpa_tl, { ##1 } }
              \@@_set_option_value:nV
                { #2 }
                \l_tmpa_clist
            }
          }
        \keys_define:nV
          { markdown/latex-options }
          \l_tmpa_tl
      }
  }
\cs_generate_variant:Nn
  \clist_set:Nn
  { NV }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\prg_generate_conditional_variant:Nnn
  \str_if_eq:nn
  { en }
  { F }
\msg_new:nnn
  { @@ }
  { malformed-name-for-clist-option }
  {
    Clist~option~name~#1~does~not~end~with~-s.
  }
\@@_latex_define_option_commands_and_keyvals:
\ExplSyntaxOff
```

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

To ensure compatibility with the `minted` package [10, Section 5.1], which supports the `finalizecache` and `frozencache` package options with similar semantics, the Markdown package also recognizes these as aliases and recognizes them as document

class options. By passing `finalizecache` and `frozencache` as document class options, you may conveniently control the behavior of both packages at once:

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

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

```
\DeclareOption{finalizecache}{\markdownSetup{finalizeCache}}
\DeclareOption{frozencache}{\markdownSetup{frozenCache}}
```

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

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

2.3.2.5 Plain TeX Markdown Token Renderers

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

```
\ExplSyntaxOn
\cs_new:Nn \@@_latex_define_renderers:
  {
    \seq_map_function:NN
      \g_@@_renderers_seq
      \@@_latex_define_renderer:n
  }
\cs_new:Nn \@@_latex_define_renderer:n
  {
    \@@_renderer_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
```

```
    \@@_latex_define_renderer:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_renderer_tl_to_csname:nN
  {
    \tl_set:Nn
      \l_tmpa_tl
      { \str_uppercase:n { #1 } }
    \tl_set:Nx
      #2
      {
        markdownRenderer
        \tl_head:f { \l_tmpa_tl }
        \tl_tail:n { #1 }
      }
  }
\cs_new:Nn \@@_latex_define_renderer:nNn
  {
    \@@_with_various_cases:nn
      { #1 }
      {
        \keys_define:nn
          { markdown/latex-options/renderers }
          {
            ##1 .code:n = {
              \cs_generate_from_arg_count:NNnn
                #2
                \cs_set:Npn
                { #3 }
                { ####1 }
            },
          }
      }
  }
\cs_generate_variant:Nn
  \@@_latex_define_renderer:nNn
  { ncV }
\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).

```
\ExplSyntaxOn
\cs_new:Nn \@@_latex_define_renderer_prototypes:
  {
    \seq_map_function:NN
      \g_@@_renderers_seq
      \@@_latex_define_renderer_prototype:n
  }
\cs_new:Nn \@@_latex_define_renderer_prototype:n
  {
    \@@_renderer_prototype_tl_to_csname:nN
      { #1 }
      \l_tmpa_tl
    \prop_get:NnN
      \g_@@_renderer_arities_prop
      { #1 }
      \l_tmpb_tl
    \@@_latex_define_renderer_prototype:ncV
      { #1 }
      { \l_tmpa_tl }
      \l_tmpb_tl
  }
\cs_new:Nn \@@_latex_define_renderer_prototype:nNn
  {
    \@@_with_various_cases:nn
      { #1 }
      {
        \keys_define:nn
          { markdown/latex-options/renderer-prototypes }
          {
            ##1 .code:n = {
              \cs_generate_from_arg_count:NNnn
                #2
                \cs_set:Npn
                { #3 }
                { ####1 }
            },
          }
```

```
      }
  }
\cs_generate_variant:Nn
  \@@_latex_define_renderer_prototype:nNn
  { ncV }
\ExplSyntaxOff
```

The following example LaTeX 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 ConTeXt Interface

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

```
\writestatus{loading}{ConTeXt User Module / markdown}%
\startmodule[markdown]
\unprotect
```

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

```
\def\markdownInfo#1{\writestatus{markdown}{#1.}}%
\def\markdownWarning#1{\writestatus{markdown\space warn}{#1.}}%
\def\dospecials{\do\ \do\\\do\{\do\}\do\$\do\&%
  \do\#\do\^\do\_\do\%\do\~}%
\input markdown/markdown
```

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

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

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

```
\let\startmarkdown\relax
\let\stopmarkdown\relax
\let\inputmarkdown\relax
```

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

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

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

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

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

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

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

2.4.2 Options

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

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

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

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_setup:n
  {
    \keys_set:nn
      { markdown/context-options }
      { #1 }
  }
\long\def\setupmarkdown[#1]
  {
    \@@_setup:n
      { #1 }
  }
\ExplSyntaxOff
```

2.4.2.1 ConTEXt Interface Options

We define the ⟨*key*⟩=⟨*value*⟩ interface for all option macros recognized by the Lua and plain TEX interfaces.

```
\ExplSyntaxOn
\cs_new:Nn \@@_context_define_option_commands_and_keyvals:
  {
    \seq_map_inline:Nn
      \g_@@_option_layers_seq
      {
        \seq_map_inline:cn
          { g_@@_ ##1 _options_seq }
          {
```

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

```
            \@@_with_various_cases:nn
              { ####1 }
              {
                \@@_context_define_option_keyval:nnn
                  { ##1 }
                  { ####1 }
                  { ########1 }
              }
          }
      }
```

```
  }
```

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

```
\cs_new:Nn \@@_caseless:N
  {
    \regex_replace_all:nnN
      { ([a-z])([A-Z]) }
      { \1 \c { str_lowercase:n } \cB\{ \2 \cE\} }
      #1
    \tl_set:Nx
      #1
      { #1 }
  }
\seq_gput_right:Nn \g_@@_cases_seq { @@_caseless:N }
\cs_new:Nn \@@_context_define_option_keyval:nnn
  {
    \prop_get:cnN
      { g_@@_ #1 _option_types_prop }
      { #2 }
      \l_tmpa_tl
    \keys_define:nn
      { markdown/context-options }
      {
        #3 .code:n = {
          \tl_set:Nx
            \l_tmpa_tl
            {
              \str_case:nnF
                { ##1 }
                {
                  { yes } { true }
                  { no } { false }
                }
                { ##1 }
            }
          \@@_set_option_value:nV
            { #2 }
            \l_tmpa_tl
        },
      }
    \str_if_eq:VVT
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \keys_define:nn
          { markdown/context-options }
```

```
          {
            #3 .default:n = { true },
          }
      }
  }
\cs_generate_variant:Nn
  \@@_set_option_value:nn
  { nV }
\@@_context_define_option_commands_and_keyvals:
\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).

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

```
2625 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.

```
2626 function util.err(msg, exit_code)
2627   io.stderr:write("markdown.lua: " .. msg .. "\n")
2628   os.exit(exit_code or 1)
2629 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.

```
2630 function util.cache(dir, string, salt, transform, suffix)
2631   local digest = md5.sumhexa(string .. (salt or ""))
2632   local name = util.pathname(dir, digest .. suffix)
2633   local file = io.open(name, "r")
2634   if file == nil then -- If no cache entry exists, then create a new one.
2635     file = assert(io.open(name, "w"),
2636       [[Could not open file "]] .. name .. [[" for writing]])
2637     local result = string
2638     if transform ~= nil then
2639       result = transform(result)
2640     end
2641     assert(file:write(result))
2642     assert(file:close())
2643   end
2644   return name
2645 end
```

The `util.cache_verbatim` method strips whitespaces from the end of `string` and calls `util.cache` with `dir`, `string`, no salt or transformations, and the `.verbatim` suffix.

```
2646 function util.cache_verbatim(dir, string)
2647   local name = util.cache(dir, string, nil, nil, ".verbatim")
2648   return name
2649 end
```

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

```
2650 function util.table_copy(t)
2651   local u = { }
2652   for k, v in pairs(t) do u[k] = v end
2653   return setmetatable(u, getmetatable(t))
2654 end
```

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

```
2655 function util.encode_json_string(s)
2656   s = s:gsub([[\]], [[\\]])
```

```
  s = s:gsub([["]], [[\"]])
  return [["]] .. s .. [["]]
end
```

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

```
util.lookup_files = (function()
  local ran_ok, kpse = pcall(require, "kpse")
  if ran_ok then
    kpse.set_program_name("luatex")
  else
    kpse = { lookup = function(f, _) return f end }
  end

  local function lookup_files(f, options)
    return kpse.lookup(f, options)
  end

  return lookup_files
end)()
```

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

```
function util.expand_tabs_in_line(s, tabstop)
  local tab = tabstop or 4
  local corr = 0
  return (s:gsub("()\t", function(p)
            local sp = tab - (p - 1 + corr) % tab
            corr = corr - 1 + sp
            return string.rep(" ", sp)
          end))
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.

```
function util.walk(t, f)
  local typ = type(t)
  if typ == "string" then
    f(t)
  elseif typ == "table" then
    local i = 1
```

```
2689      local n
2690      n = t[i]
2691      while n do
2692        util.walk(n, f)
2693        i = i + 1
2694        n = t[i]
2695      end
2696    elseif typ == "function" then
2697      local ok, val = pcall(t)
2698      if ok then
2699        util.walk(val,f)
2700      end
2701    else
2702      f(tostring(t))
2703    end
2704  end
```

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

```
2705  function util.flatten(ary)
2706    local new = {}
2707    for _,v in ipairs(ary) do
2708      if type(v) == "table" then
2709        for _,w in ipairs(util.flatten(v)) do
2710          new[#new + 1] = w
2711        end
2712      else
2713        new[#new + 1] = v
2714      end
2715    end
2716    return new
2717  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.

```
2718  function util.rope_to_string(rope)
2719    local buffer = {}
2720    util.walk(rope, function(x) buffer[#buffer + 1] = x end)
2721    return table.concat(buffer)
2722  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.

```
2723  function util.rope_last(rope)
2724    if #rope == 0 then
2725      return nil
2726    else
2727      local l = rope[#rope]
```

```
    if type(l) == "table" then
      return util.rope_last(l)
    else
      return l
    end
  end
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 $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.intersperse(ary, x)
  local new = {}
  local l = #ary
  for i,v in ipairs(ary) do
    local n = #new
    new[n + 1] = v
    if i ~= l then
      new[n + 2] = x
    end
  end
  return new
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 $1 \leqslant$ `i` $\leqslant$ `#ary`.

```
function util.map(ary, f)
  local new = {}
  for i,v in ipairs(ary) do
    new[i] = f(v)
  end
  return new
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 occurances of escapable strings and characters (in this order).

The method uses LPeg, which is faster than the Lua `string.gsub` built-in method.

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

Build a string of escapable characters.

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

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

```
local escapable = S(char_escapes_list) / char_escapes
```

If `string_escapes` is provided, turn `escapable` into the

$$\sum_{\texttt{(k,v)}\in\texttt{string\_escapes}} \texttt{P(k) / v} + \texttt{escapable}$$

capture that replaces any occurance of the string `k` with the string `v` for each $(\texttt{k}, \texttt{v}) \in \texttt{string\_escapes}$. Note that the pattern summation is not commutative and its operands are inspected in the summation order during the matching. As a corrolary, the strings always take precedence over the characters.

```
if string_escapes then
  for k,v in pairs(string_escapes) do
    escapable = P(k) / v + escapable
  end
end
```

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

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

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

```
return function(s)
  return lpeg.match(escape_string, s)
end
end
```

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

```
function util.pathname(dir, file)
  if #dir == 0 then
    return file
  else
    return dir .. "/" .. file
  end
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.

```
local entities = {}

```

```
local character_entities = {
  ["Tab"] = 9,
  ["NewLine"] = 10,
  ["excl"] = 33,
  ["quot"] = 34,
  ["QUOT"] = 34,
  ["num"] = 35,
  ["dollar"] = 36,
  ["percnt"] = 37,
  ["amp"] = 38,
  ["AMP"] = 38,
  ["apos"] = 39,
  ["lpar"] = 40,
  ["rpar"] = 41,
  ["ast"] = 42,
  ["midast"] = 42,
  ["plus"] = 43,
  ["comma"] = 44,
  ["period"] = 46,
  ["sol"] = 47,
  ["colon"] = 58,
  ["semi"] = 59,
  ["lt"] = 60,
  ["LT"] = 60,
  ["equals"] = 61,
  ["gt"] = 62,
  ["GT"] = 62,
  ["quest"] = 63,
  ["commat"] = 64,
  ["lsqb"] = 91,
  ["lbrack"] = 91,
  ["bsol"] = 92,
  ["rsqb"] = 93,
  ["rbrack"] = 93,
  ["Hat"] = 94,
  ["lowbar"] = 95,
  ["grave"] = 96,
  ["DiacriticalGrave"] = 96,
  ["lcub"] = 123,
  ["lbrace"] = 123,
  ["verbar"] = 124,
  ["vert"] = 124,
  ["VerticalLine"] = 124,
  ["rcub"] = 125,
  ["rbrace"] = 125,
  ["nbsp"] = 160,
  ["NonBreakingSpace"] = 160,
```

```
2826    ["iexcl"] = 161,
2827    ["cent"] = 162,
2828    ["pound"] = 163,
2829    ["curren"] = 164,
2830    ["yen"] = 165,
2831    ["brvbar"] = 166,
2832    ["sect"] = 167,
2833    ["Dot"] = 168,
2834    ["die"] = 168,
2835    ["DoubleDot"] = 168,
2836    ["uml"] = 168,
2837    ["copy"] = 169,
2838    ["COPY"] = 169,
2839    ["ordf"] = 170,
2840    ["laquo"] = 171,
2841    ["not"] = 172,
2842    ["shy"] = 173,
2843    ["reg"] = 174,
2844    ["circledR"] = 174,
2845    ["REG"] = 174,
2846    ["macr"] = 175,
2847    ["OverBar"] = 175,
2848    ["strns"] = 175,
2849    ["deg"] = 176,
2850    ["plusmn"] = 177,
2851    ["pm"] = 177,
2852    ["PlusMinus"] = 177,
2853    ["sup2"] = 178,
2854    ["sup3"] = 179,
2855    ["acute"] = 180,
2856    ["DiacriticalAcute"] = 180,
2857    ["micro"] = 181,
2858    ["para"] = 182,
2859    ["middot"] = 183,
2860    ["centerdot"] = 183,
2861    ["CenterDot"] = 183,
2862    ["cedil"] = 184,
2863    ["Cedilla"] = 184,
2864    ["sup1"] = 185,
2865    ["ordm"] = 186,
2866    ["raquo"] = 187,
2867    ["frac14"] = 188,
2868    ["frac12"] = 189,
2869    ["half"] = 189,
2870    ["frac34"] = 190,
2871    ["iquest"] = 191,
2872    ["Agrave"] = 192,
```

```
2873    ["Aacute"] = 193,
2874    ["Acirc"] = 194,
2875    ["Atilde"] = 195,
2876    ["Auml"] = 196,
2877    ["Aring"] = 197,
2878    ["AElig"] = 198,
2879    ["Ccedil"] = 199,
2880    ["Egrave"] = 200,
2881    ["Eacute"] = 201,
2882    ["Ecirc"] = 202,
2883    ["Euml"] = 203,
2884    ["Igrave"] = 204,
2885    ["Iacute"] = 205,
2886    ["Icirc"] = 206,
2887    ["Iuml"] = 207,
2888    ["ETH"] = 208,
2889    ["Ntilde"] = 209,
2890    ["Ograve"] = 210,
2891    ["Oacute"] = 211,
2892    ["Ocirc"] = 212,
2893    ["Otilde"] = 213,
2894    ["Ouml"] = 214,
2895    ["times"] = 215,
2896    ["Oslash"] = 216,
2897    ["Ugrave"] = 217,
2898    ["Uacute"] = 218,
2899    ["Ucirc"] = 219,
2900    ["Uuml"] = 220,
2901    ["Yacute"] = 221,
2902    ["THORN"] = 222,
2903    ["szlig"] = 223,
2904    ["agrave"] = 224,
2905    ["aacute"] = 225,
2906    ["acirc"] = 226,
2907    ["atilde"] = 227,
2908    ["auml"] = 228,
2909    ["aring"] = 229,
2910    ["aelig"] = 230,
2911    ["ccedil"] = 231,
2912    ["egrave"] = 232,
2913    ["eacute"] = 233,
2914    ["ecirc"] = 234,
2915    ["euml"] = 235,
2916    ["igrave"] = 236,
2917    ["iacute"] = 237,
2918    ["icirc"] = 238,
2919    ["iuml"] = 239,
```

```
  ["eth"] = 240,
  ["ntilde"] = 241,
  ["ograve"] = 242,
  ["oacute"] = 243,
  ["ocirc"] = 244,
  ["otilde"] = 245,
  ["ouml"] = 246,
  ["divide"] = 247,
  ["div"] = 247,
  ["oslash"] = 248,
  ["ugrave"] = 249,
  ["uacute"] = 250,
  ["ucirc"] = 251,
  ["uuml"] = 252,
  ["yacute"] = 253,
  ["thorn"] = 254,
  ["yuml"] = 255,
  ["Amacr"] = 256,
  ["amacr"] = 257,
  ["Abreve"] = 258,
  ["abreve"] = 259,
  ["Aogon"] = 260,
  ["aogon"] = 261,
  ["Cacute"] = 262,
  ["cacute"] = 263,
  ["Ccirc"] = 264,
  ["ccirc"] = 265,
  ["Cdot"] = 266,
  ["cdot"] = 267,
  ["Ccaron"] = 268,
  ["ccaron"] = 269,
  ["Dcaron"] = 270,
  ["dcaron"] = 271,
  ["Dstrok"] = 272,
  ["dstrok"] = 273,
  ["Emacr"] = 274,
  ["emacr"] = 275,
  ["Edot"] = 278,
  ["edot"] = 279,
  ["Eogon"] = 280,
  ["eogon"] = 281,
  ["Ecaron"] = 282,
  ["ecaron"] = 283,
  ["Gcirc"] = 284,
  ["gcirc"] = 285,
  ["Gbreve"] = 286,
  ["gbreve"] = 287,
```

```
2967    ["Gdot"] = 288,
2968    ["gdot"] = 289,
2969    ["Gcedil"] = 290,
2970    ["Hcirc"] = 292,
2971    ["hcirc"] = 293,
2972    ["Hstrok"] = 294,
2973    ["hstrok"] = 295,
2974    ["Itilde"] = 296,
2975    ["itilde"] = 297,
2976    ["Imacr"] = 298,
2977    ["imacr"] = 299,
2978    ["Iogon"] = 302,
2979    ["iogon"] = 303,
2980    ["Idot"] = 304,
2981    ["imath"] = 305,
2982    ["inodot"] = 305,
2983    ["IJlig"] = 306,
2984    ["ijlig"] = 307,
2985    ["Jcirc"] = 308,
2986    ["jcirc"] = 309,
2987    ["Kcedil"] = 310,
2988    ["kcedil"] = 311,
2989    ["kgreen"] = 312,
2990    ["Lacute"] = 313,
2991    ["lacute"] = 314,
2992    ["Lcedil"] = 315,
2993    ["lcedil"] = 316,
2994    ["Lcaron"] = 317,
2995    ["lcaron"] = 318,
2996    ["Lmidot"] = 319,
2997    ["lmidot"] = 320,
2998    ["Lstrok"] = 321,
2999    ["lstrok"] = 322,
3000    ["Nacute"] = 323,
3001    ["nacute"] = 324,
3002    ["Ncedil"] = 325,
3003    ["ncedil"] = 326,
3004    ["Ncaron"] = 327,
3005    ["ncaron"] = 328,
3006    ["napos"] = 329,
3007    ["ENG"] = 330,
3008    ["eng"] = 331,
3009    ["Omacr"] = 332,
3010    ["omacr"] = 333,
3011    ["Odblac"] = 336,
3012    ["odblac"] = 337,
3013    ["OElig"] = 338,
```

```
3014    ["oelig"] = 339,
3015    ["Racute"] = 340,
3016    ["racute"] = 341,
3017    ["Rcedil"] = 342,
3018    ["rcedil"] = 343,
3019    ["Rcaron"] = 344,
3020    ["rcaron"] = 345,
3021    ["Sacute"] = 346,
3022    ["sacute"] = 347,
3023    ["Scirc"] = 348,
3024    ["scirc"] = 349,
3025    ["Scedil"] = 350,
3026    ["scedil"] = 351,
3027    ["Scaron"] = 352,
3028    ["scaron"] = 353,
3029    ["Tcedil"] = 354,
3030    ["tcedil"] = 355,
3031    ["Tcaron"] = 356,
3032    ["tcaron"] = 357,
3033    ["Tstrok"] = 358,
3034    ["tstrok"] = 359,
3035    ["Utilde"] = 360,
3036    ["utilde"] = 361,
3037    ["Umacr"] = 362,
3038    ["umacr"] = 363,
3039    ["Ubreve"] = 364,
3040    ["ubreve"] = 365,
3041    ["Uring"] = 366,
3042    ["uring"] = 367,
3043    ["Udblac"] = 368,
3044    ["udblac"] = 369,
3045    ["Uogon"] = 370,
3046    ["uogon"] = 371,
3047    ["Wcirc"] = 372,
3048    ["wcirc"] = 373,
3049    ["Ycirc"] = 374,
3050    ["ycirc"] = 375,
3051    ["Yuml"] = 376,
3052    ["Zacute"] = 377,
3053    ["zacute"] = 378,
3054    ["Zdot"] = 379,
3055    ["zdot"] = 380,
3056    ["Zcaron"] = 381,
3057    ["zcaron"] = 382,
3058    ["fnof"] = 402,
3059    ["imped"] = 437,
3060    ["gacute"] = 501,
```

```
  ["jmath"] = 567,
  ["circ"] = 710,
  ["caron"] = 711,
  ["Hacek"] = 711,
  ["breve"] = 728,
  ["Breve"] = 728,
  ["dot"] = 729,
  ["DiacriticalDot"] = 729,
  ["ring"] = 730,
  ["ogon"] = 731,
  ["tilde"] = 732,
  ["DiacriticalTilde"] = 732,
  ["dblac"] = 733,
  ["DiacriticalDoubleAcute"] = 733,
  ["DownBreve"] = 785,
  ["UnderBar"] = 818,
  ["Alpha"] = 913,
  ["Beta"] = 914,
  ["Gamma"] = 915,
  ["Delta"] = 916,
  ["Epsilon"] = 917,
  ["Zeta"] = 918,
  ["Eta"] = 919,
  ["Theta"] = 920,
  ["Iota"] = 921,
  ["Kappa"] = 922,
  ["Lambda"] = 923,
  ["Mu"] = 924,
  ["Nu"] = 925,
  ["Xi"] = 926,
  ["Omicron"] = 927,
  ["Pi"] = 928,
  ["Rho"] = 929,
  ["Sigma"] = 931,
  ["Tau"] = 932,
  ["Upsilon"] = 933,
  ["Phi"] = 934,
  ["Chi"] = 935,
  ["Psi"] = 936,
  ["Omega"] = 937,
  ["alpha"] = 945,
  ["beta"] = 946,
  ["gamma"] = 947,
  ["delta"] = 948,
  ["epsiv"] = 949,
  ["varepsilon"] = 949,
  ["epsilon"] = 949,
```

```
["zeta"] = 950,
["eta"] = 951,
["theta"] = 952,
["iota"] = 953,
["kappa"] = 954,
["lambda"] = 955,
["mu"] = 956,
["nu"] = 957,
["xi"] = 958,
["omicron"] = 959,
["pi"] = 960,
["rho"] = 961,
["sigmav"] = 962,
["varsigma"] = 962,
["sigmaf"] = 962,
["sigma"] = 963,
["tau"] = 964,
["upsi"] = 965,
["upsilon"] = 965,
["phi"] = 966,
["phiv"] = 966,
["varphi"] = 966,
["chi"] = 967,
["psi"] = 968,
["omega"] = 969,
["thetav"] = 977,
["vartheta"] = 977,
["thetasym"] = 977,
["Upsi"] = 978,
["upsih"] = 978,
["straightphi"] = 981,
["piv"] = 982,
["varpi"] = 982,
["Gammad"] = 988,
["gammad"] = 989,
["digamma"] = 989,
["kappav"] = 1008,
["varkappa"] = 1008,
["rhov"] = 1009,
["varrho"] = 1009,
["epsi"] = 1013,
["straightepsilon"] = 1013,
["bepsi"] = 1014,
["backepsilon"] = 1014,
["IOcy"] = 1025,
["DJcy"] = 1026,
["GJcy"] = 1027,
```

```
3155    ["Jukcy"] = 1028,
3156    ["DScy"] = 1029,
3157    ["Iukcy"] = 1030,
3158    ["YIcy"] = 1031,
3159    ["Jsercy"] = 1032,
3160    ["LJcy"] = 1033,
3161    ["NJcy"] = 1034,
3162    ["TSHcy"] = 1035,
3163    ["KJcy"] = 1036,
3164    ["Ubrcy"] = 1038,
3165    ["DZcy"] = 1039,
3166    ["Acy"] = 1040,
3167    ["Bcy"] = 1041,
3168    ["Vcy"] = 1042,
3169    ["Gcy"] = 1043,
3170    ["Dcy"] = 1044,
3171    ["IEcy"] = 1045,
3172    ["ZHcy"] = 1046,
3173    ["Zcy"] = 1047,
3174    ["Icy"] = 1048,
3175    ["Jcy"] = 1049,
3176    ["Kcy"] = 1050,
3177    ["Lcy"] = 1051,
3178    ["Mcy"] = 1052,
3179    ["Ncy"] = 1053,
3180    ["Ocy"] = 1054,
3181    ["Pcy"] = 1055,
3182    ["Rcy"] = 1056,
3183    ["Scy"] = 1057,
3184    ["Tcy"] = 1058,
3185    ["Ucy"] = 1059,
3186    ["Fcy"] = 1060,
3187    ["KHcy"] = 1061,
3188    ["TScy"] = 1062,
3189    ["CHcy"] = 1063,
3190    ["SHcy"] = 1064,
3191    ["SHCHcy"] = 1065,
3192    ["HARDcy"] = 1066,
3193    ["Ycy"] = 1067,
3194    ["SOFTcy"] = 1068,
3195    ["Ecy"] = 1069,
3196    ["YUcy"] = 1070,
3197    ["YAcy"] = 1071,
3198    ["acy"] = 1072,
3199    ["bcy"] = 1073,
3200    ["vcy"] = 1074,
3201    ["gcy"] = 1075,
```

```
3202  ["dcy"] = 1076,
3203  ["iecy"] = 1077,
3204  ["zhcy"] = 1078,
3205  ["zcy"] = 1079,
3206  ["icy"] = 1080,
3207  ["jcy"] = 1081,
3208  ["kcy"] = 1082,
3209  ["lcy"] = 1083,
3210  ["mcy"] = 1084,
3211  ["ncy"] = 1085,
3212  ["ocy"] = 1086,
3213  ["pcy"] = 1087,
3214  ["rcy"] = 1088,
3215  ["scy"] = 1089,
3216  ["tcy"] = 1090,
3217  ["ucy"] = 1091,
3218  ["fcy"] = 1092,
3219  ["khcy"] = 1093,
3220  ["tscy"] = 1094,
3221  ["chcy"] = 1095,
3222  ["shcy"] = 1096,
3223  ["shchcy"] = 1097,
3224  ["hardcy"] = 1098,
3225  ["ycy"] = 1099,
3226  ["softcy"] = 1100,
3227  ["ecy"] = 1101,
3228  ["yucy"] = 1102,
3229  ["yacy"] = 1103,
3230  ["iocy"] = 1105,
3231  ["djcy"] = 1106,
3232  ["gjcy"] = 1107,
3233  ["jukcy"] = 1108,
3234  ["dscy"] = 1109,
3235  ["iukcy"] = 1110,
3236  ["yicy"] = 1111,
3237  ["jsercy"] = 1112,
3238  ["ljcy"] = 1113,
3239  ["njcy"] = 1114,
3240  ["tshcy"] = 1115,
3241  ["kjcy"] = 1116,
3242  ["ubrcy"] = 1118,
3243  ["dzcy"] = 1119,
3244  ["ensp"] = 8194,
3245  ["emsp"] = 8195,
3246  ["emsp13"] = 8196,
3247  ["emsp14"] = 8197,
3248  ["numsp"] = 8199,
```

```
  ["puncsp"] = 8200,
  ["thinsp"] = 8201,
  ["ThinSpace"] = 8201,
  ["hairsp"] = 8202,
  ["VeryThinSpace"] = 8202,
  ["ZeroWidthSpace"] = 8203,
  ["NegativeVeryThinSpace"] = 8203,
  ["NegativeThinSpace"] = 8203,
  ["NegativeMediumSpace"] = 8203,
  ["NegativeThickSpace"] = 8203,
  ["zwnj"] = 8204,
  ["zwj"] = 8205,
  ["lrm"] = 8206,
  ["rlm"] = 8207,
  ["hyphen"] = 8208,
  ["dash"] = 8208,
  ["ndash"] = 8211,
  ["mdash"] = 8212,
  ["horbar"] = 8213,
  ["Verbar"] = 8214,
  ["Vert"] = 8214,
  ["lsquo"] = 8216,
  ["OpenCurlyQuote"] = 8216,
  ["rsquo"] = 8217,
  ["rsquor"] = 8217,
  ["CloseCurlyQuote"] = 8217,
  ["lsquor"] = 8218,
  ["sbquo"] = 8218,
  ["ldquo"] = 8220,
  ["OpenCurlyDoubleQuote"] = 8220,
  ["rdquo"] = 8221,
  ["rdquor"] = 8221,
  ["CloseCurlyDoubleQuote"] = 8221,
  ["ldquor"] = 8222,
  ["bdquo"] = 8222,
  ["dagger"] = 8224,
  ["Dagger"] = 8225,
  ["ddagger"] = 8225,
  ["bull"] = 8226,
  ["bullet"] = 8226,
  ["nldr"] = 8229,
  ["hellip"] = 8230,
  ["mldr"] = 8230,
  ["permil"] = 8240,
  ["pertenk"] = 8241,
  ["prime"] = 8242,
  ["Prime"] = 8243,
```

```
  ["tprime"] = 8244,
  ["bprime"] = 8245,
  ["backprime"] = 8245,
  ["lsaquo"] = 8249,
  ["rsaquo"] = 8250,
  ["oline"] = 8254,
  ["caret"] = 8257,
  ["hybull"] = 8259,
  ["frasl"] = 8260,
  ["bsemi"] = 8271,
  ["qprime"] = 8279,
  ["MediumSpace"] = 8287,
  ["NoBreak"] = 8288,
  ["ApplyFunction"] = 8289,
  ["af"] = 8289,
  ["InvisibleTimes"] = 8290,
  ["it"] = 8290,
  ["InvisibleComma"] = 8291,
  ["ic"] = 8291,
  ["euro"] = 8364,
  ["tdot"] = 8411,
  ["TripleDot"] = 8411,
  ["DotDot"] = 8412,
  ["Copf"] = 8450,
  ["complexes"] = 8450,
  ["incare"] = 8453,
  ["gscr"] = 8458,
  ["hamilt"] = 8459,
  ["HilbertSpace"] = 8459,
  ["Hscr"] = 8459,
  ["Hfr"] = 8460,
  ["Poincareplane"] = 8460,
  ["quaternions"] = 8461,
  ["Hopf"] = 8461,
  ["planckh"] = 8462,
  ["planck"] = 8463,
  ["hbar"] = 8463,
  ["plankv"] = 8463,
  ["hslash"] = 8463,
  ["Iscr"] = 8464,
  ["imagline"] = 8464,
  ["image"] = 8465,
  ["Im"] = 8465,
  ["imagpart"] = 8465,
  ["Ifr"] = 8465,
  ["Lscr"] = 8466,
  ["lagran"] = 8466,
```

```
3343    ["Laplacetrf"] = 8466,
3344    ["ell"] = 8467,
3345    ["Nopf"] = 8469,
3346    ["naturals"] = 8469,
3347    ["numero"] = 8470,
3348    ["copysr"] = 8471,
3349    ["weierp"] = 8472,
3350    ["wp"] = 8472,
3351    ["Popf"] = 8473,
3352    ["primes"] = 8473,
3353    ["rationals"] = 8474,
3354    ["Qopf"] = 8474,
3355    ["Rscr"] = 8475,
3356    ["realine"] = 8475,
3357    ["real"] = 8476,
3358    ["Re"] = 8476,
3359    ["realpart"] = 8476,
3360    ["Rfr"] = 8476,
3361    ["reals"] = 8477,
3362    ["Ropf"] = 8477,
3363    ["rx"] = 8478,
3364    ["trade"] = 8482,
3365    ["TRADE"] = 8482,
3366    ["integers"] = 8484,
3367    ["Zopf"] = 8484,
3368    ["ohm"] = 8486,
3369    ["mho"] = 8487,
3370    ["Zfr"] = 8488,
3371    ["zeetrf"] = 8488,
3372    ["iiota"] = 8489,
3373    ["angst"] = 8491,
3374    ["bernou"] = 8492,
3375    ["Bernoullis"] = 8492,
3376    ["Bscr"] = 8492,
3377    ["Cfr"] = 8493,
3378    ["Cayleys"] = 8493,
3379    ["escr"] = 8495,
3380    ["Escr"] = 8496,
3381    ["expectation"] = 8496,
3382    ["Fscr"] = 8497,
3383    ["Fouriertrf"] = 8497,
3384    ["phmmat"] = 8499,
3385    ["Mellintrf"] = 8499,
3386    ["Mscr"] = 8499,
3387    ["order"] = 8500,
3388    ["orderof"] = 8500,
3389    ["oscr"] = 8500,
```

```
["alefsym"] = 8501,
["aleph"] = 8501,
["beth"] = 8502,
["gimel"] = 8503,
["daleth"] = 8504,
["CapitalDifferentialD"] = 8517,
["DD"] = 8517,
["DifferentialD"] = 8518,
["dd"] = 8518,
["ExponentialE"] = 8519,
["exponentiale"] = 8519,
["ee"] = 8519,
["ImaginaryI"] = 8520,
["ii"] = 8520,
["frac13"] = 8531,
["frac23"] = 8532,
["frac15"] = 8533,
["frac25"] = 8534,
["frac35"] = 8535,
["frac45"] = 8536,
["frac16"] = 8537,
["frac56"] = 8538,
["frac18"] = 8539,
["frac38"] = 8540,
["frac58"] = 8541,
["frac78"] = 8542,
["larr"] = 8592,
["leftarrow"] = 8592,
["LeftArrow"] = 8592,
["slarr"] = 8592,
["ShortLeftArrow"] = 8592,
["uarr"] = 8593,
["uparrow"] = 8593,
["UpArrow"] = 8593,
["ShortUpArrow"] = 8593,
["rarr"] = 8594,
["rightarrow"] = 8594,
["RightArrow"] = 8594,
["srarr"] = 8594,
["ShortRightArrow"] = 8594,
["darr"] = 8595,
["downarrow"] = 8595,
["DownArrow"] = 8595,
["ShortDownArrow"] = 8595,
["harr"] = 8596,
["leftrightarrow"] = 8596,
["LeftRightArrow"] = 8596,
```

```
  ["varr"] = 8597,
  ["updownarrow"] = 8597,
  ["UpDownArrow"] = 8597,
  ["nwarr"] = 8598,
  ["UpperLeftArrow"] = 8598,
  ["nwarrow"] = 8598,
  ["nearr"] = 8599,
  ["UpperRightArrow"] = 8599,
  ["nearrow"] = 8599,
  ["searr"] = 8600,
  ["searrow"] = 8600,
  ["LowerRightArrow"] = 8600,
  ["swarr"] = 8601,
  ["swarrow"] = 8601,
  ["LowerLeftArrow"] = 8601,
  ["nlarr"] = 8602,
  ["nleftarrow"] = 8602,
  ["nrarr"] = 8603,
  ["nrightarrow"] = 8603,
  ["rarrw"] = 8605,
  ["rightsquigarrow"] = 8605,
  ["Larr"] = 8606,
  ["twoheadleftarrow"] = 8606,
  ["Uarr"] = 8607,
  ["Rarr"] = 8608,
  ["twoheadrightarrow"] = 8608,
  ["Darr"] = 8609,
  ["larrtl"] = 8610,
  ["leftarrowtail"] = 8610,
  ["rarrtl"] = 8611,
  ["rightarrowtail"] = 8611,
  ["LeftTeeArrow"] = 8612,
  ["mapstoleft"] = 8612,
  ["UpTeeArrow"] = 8613,
  ["mapstoup"] = 8613,
  ["map"] = 8614,
  ["RightTeeArrow"] = 8614,
  ["mapsto"] = 8614,
  ["DownTeeArrow"] = 8615,
  ["mapstodown"] = 8615,
  ["larrhk"] = 8617,
  ["hookleftarrow"] = 8617,
  ["rarrhk"] = 8618,
  ["hookrightarrow"] = 8618,
  ["larrlp"] = 8619,
  ["looparrowleft"] = 8619,
  ["rarrlp"] = 8620,
```

```
3484    ["looparrowright"] = 8620,
3485    ["harrw"] = 8621,
3486    ["leftrightsquigarrow"] = 8621,
3487    ["nharr"] = 8622,
3488    ["nleftrightarrow"] = 8622,
3489    ["lsh"] = 8624,
3490    ["Lsh"] = 8624,
3491    ["rsh"] = 8625,
3492    ["Rsh"] = 8625,
3493    ["ldsh"] = 8626,
3494    ["rdsh"] = 8627,
3495    ["crarr"] = 8629,
3496    ["cularr"] = 8630,
3497    ["curvearrowleft"] = 8630,
3498    ["curarr"] = 8631,
3499    ["curvearrowright"] = 8631,
3500    ["olarr"] = 8634,
3501    ["circlearrowleft"] = 8634,
3502    ["orarr"] = 8635,
3503    ["circlearrowright"] = 8635,
3504    ["lharu"] = 8636,
3505    ["LeftVector"] = 8636,
3506    ["leftharpoonup"] = 8636,
3507    ["lhard"] = 8637,
3508    ["leftharpoondown"] = 8637,
3509    ["DownLeftVector"] = 8637,
3510    ["uharr"] = 8638,
3511    ["upharpoonright"] = 8638,
3512    ["RightUpVector"] = 8638,
3513    ["uharl"] = 8639,
3514    ["upharpoonleft"] = 8639,
3515    ["LeftUpVector"] = 8639,
3516    ["rharu"] = 8640,
3517    ["RightVector"] = 8640,
3518    ["rightharpoonup"] = 8640,
3519    ["rhard"] = 8641,
3520    ["rightharpoondown"] = 8641,
3521    ["DownRightVector"] = 8641,
3522    ["dharr"] = 8642,
3523    ["RightDownVector"] = 8642,
3524    ["downharpoonright"] = 8642,
3525    ["dharl"] = 8643,
3526    ["LeftDownVector"] = 8643,
3527    ["downharpoonleft"] = 8643,
3528    ["rlarr"] = 8644,
3529    ["rightleftarrows"] = 8644,
3530    ["RightArrowLeftArrow"] = 8644,
```

```
  ["udarr"] = 8645,
  ["UpArrowDownArrow"] = 8645,
  ["lrarr"] = 8646,
  ["leftrightarrows"] = 8646,
  ["LeftArrowRightArrow"] = 8646,
  ["llarr"] = 8647,
  ["leftleftarrows"] = 8647,
  ["uuarr"] = 8648,
  ["upuparrows"] = 8648,
  ["rrarr"] = 8649,
  ["rightrightarrows"] = 8649,
  ["ddarr"] = 8650,
  ["downdownarrows"] = 8650,
  ["lrhar"] = 8651,
  ["ReverseEquilibrium"] = 8651,
  ["leftrightharpoons"] = 8651,
  ["rlhar"] = 8652,
  ["rightleftharpoons"] = 8652,
  ["Equilibrium"] = 8652,
  ["nlArr"] = 8653,
  ["nLeftarrow"] = 8653,
  ["nhArr"] = 8654,
  ["nLeftrightarrow"] = 8654,
  ["nrArr"] = 8655,
  ["nRightarrow"] = 8655,
  ["lArr"] = 8656,
  ["Leftarrow"] = 8656,
  ["DoubleLeftArrow"] = 8656,
  ["uArr"] = 8657,
  ["Uparrow"] = 8657,
  ["DoubleUpArrow"] = 8657,
  ["rArr"] = 8658,
  ["Rightarrow"] = 8658,
  ["Implies"] = 8658,
  ["DoubleRightArrow"] = 8658,
  ["dArr"] = 8659,
  ["Downarrow"] = 8659,
  ["DoubleDownArrow"] = 8659,
  ["hArr"] = 8660,
  ["Leftrightarrow"] = 8660,
  ["DoubleLeftRightArrow"] = 8660,
  ["iff"] = 8660,
  ["vArr"] = 8661,
  ["Updownarrow"] = 8661,
  ["DoubleUpDownArrow"] = 8661,
  ["nwArr"] = 8662,
  ["neArr"] = 8663,
```

```
3578    ["seArr"] = 8664,
3579    ["swArr"] = 8665,
3580    ["lAarr"] = 8666,
3581    ["Lleftarrow"] = 8666,
3582    ["rAarr"] = 8667,
3583    ["Rrightarrow"] = 8667,
3584    ["zigrarr"] = 8669,
3585    ["larrb"] = 8676,
3586    ["LeftArrowBar"] = 8676,
3587    ["rarrb"] = 8677,
3588    ["RightArrowBar"] = 8677,
3589    ["duarr"] = 8693,
3590    ["DownArrowUpArrow"] = 8693,
3591    ["loarr"] = 8701,
3592    ["roarr"] = 8702,
3593    ["hoarr"] = 8703,
3594    ["forall"] = 8704,
3595    ["ForAll"] = 8704,
3596    ["comp"] = 8705,
3597    ["complement"] = 8705,
3598    ["part"] = 8706,
3599    ["PartialD"] = 8706,
3600    ["exist"] = 8707,
3601    ["Exists"] = 8707,
3602    ["nexist"] = 8708,
3603    ["NotExists"] = 8708,
3604    ["nexists"] = 8708,
3605    ["empty"] = 8709,
3606    ["emptyset"] = 8709,
3607    ["emptyv"] = 8709,
3608    ["varnothing"] = 8709,
3609    ["nabla"] = 8711,
3610    ["Del"] = 8711,
3611    ["isin"] = 8712,
3612    ["isinv"] = 8712,
3613    ["Element"] = 8712,
3614    ["in"] = 8712,
3615    ["notin"] = 8713,
3616    ["NotElement"] = 8713,
3617    ["notinva"] = 8713,
3618    ["niv"] = 8715,
3619    ["ReverseElement"] = 8715,
3620    ["ni"] = 8715,
3621    ["SuchThat"] = 8715,
3622    ["notni"] = 8716,
3623    ["notniva"] = 8716,
3624    ["NotReverseElement"] = 8716,
```

```
["prod"] = 8719,
["Product"] = 8719,
["coprod"] = 8720,
["Coproduct"] = 8720,
["sum"] = 8721,
["Sum"] = 8721,
["minus"] = 8722,
["mnplus"] = 8723,
["mp"] = 8723,
["MinusPlus"] = 8723,
["plusdo"] = 8724,
["dotplus"] = 8724,
["setmn"] = 8726,
["setminus"] = 8726,
["Backslash"] = 8726,
["ssetmn"] = 8726,
["smallsetminus"] = 8726,
["lowast"] = 8727,
["compfn"] = 8728,
["SmallCircle"] = 8728,
["radic"] = 8730,
["Sqrt"] = 8730,
["prop"] = 8733,
["propto"] = 8733,
["Proportional"] = 8733,
["vprop"] = 8733,
["varpropto"] = 8733,
["infin"] = 8734,
["angrt"] = 8735,
["ang"] = 8736,
["angle"] = 8736,
["angmsd"] = 8737,
["measuredangle"] = 8737,
["angsph"] = 8738,
["mid"] = 8739,
["VerticalBar"] = 8739,
["smid"] = 8739,
["shortmid"] = 8739,
["nmid"] = 8740,
["NotVerticalBar"] = 8740,
["nsmid"] = 8740,
["nshortmid"] = 8740,
["par"] = 8741,
["parallel"] = 8741,
["DoubleVerticalBar"] = 8741,
["spar"] = 8741,
["shortparallel"] = 8741,
```

```
["npar"] = 8742,
["nparallel"] = 8742,
["NotDoubleVerticalBar"] = 8742,
["nspar"] = 8742,
["nshortparallel"] = 8742,
["and"] = 8743,
["wedge"] = 8743,
["or"] = 8744,
["vee"] = 8744,
["cap"] = 8745,
["cup"] = 8746,
["int"] = 8747,
["Integral"] = 8747,
["Int"] = 8748,
["tint"] = 8749,
["iiint"] = 8749,
["conint"] = 8750,
["oint"] = 8750,
["ContourIntegral"] = 8750,
["Conint"] = 8751,
["DoubleContourIntegral"] = 8751,
["Cconint"] = 8752,
["cwint"] = 8753,
["cwconint"] = 8754,
["ClockwiseContourIntegral"] = 8754,
["awconint"] = 8755,
["CounterClockwiseContourIntegral"] = 8755,
["there4"] = 8756,
["therefore"] = 8756,
["Therefore"] = 8756,
["becaus"] = 8757,
["because"] = 8757,
["Because"] = 8757,
["ratio"] = 8758,
["Colon"] = 8759,
["Proportion"] = 8759,
["minusd"] = 8760,
["dotminus"] = 8760,
["mDDot"] = 8762,
["homtht"] = 8763,
["sim"] = 8764,
["Tilde"] = 8764,
["thksim"] = 8764,
["thicksim"] = 8764,
["bsim"] = 8765,
["backsim"] = 8765,
["ac"] = 8766,
```

```
3719    ["mstpos"] = 8766,
3720    ["acd"] = 8767,
3721    ["wreath"] = 8768,
3722    ["VerticalTilde"] = 8768,
3723    ["wr"] = 8768,
3724    ["nsim"] = 8769,
3725    ["NotTilde"] = 8769,
3726    ["esim"] = 8770,
3727    ["EqualTilde"] = 8770,
3728    ["eqsim"] = 8770,
3729    ["sime"] = 8771,
3730    ["TildeEqual"] = 8771,
3731    ["simeq"] = 8771,
3732    ["nsime"] = 8772,
3733    ["nsimeq"] = 8772,
3734    ["NotTildeEqual"] = 8772,
3735    ["cong"] = 8773,
3736    ["TildeFullEqual"] = 8773,
3737    ["simne"] = 8774,
3738    ["ncong"] = 8775,
3739    ["NotTildeFullEqual"] = 8775,
3740    ["asymp"] = 8776,
3741    ["ap"] = 8776,
3742    ["TildeTilde"] = 8776,
3743    ["approx"] = 8776,
3744    ["thkap"] = 8776,
3745    ["thickapprox"] = 8776,
3746    ["nap"] = 8777,
3747    ["NotTildeTilde"] = 8777,
3748    ["napprox"] = 8777,
3749    ["ape"] = 8778,
3750    ["approxeq"] = 8778,
3751    ["apid"] = 8779,
3752    ["bcong"] = 8780,
3753    ["backcong"] = 8780,
3754    ["asympeq"] = 8781,
3755    ["CupCap"] = 8781,
3756    ["bump"] = 8782,
3757    ["HumpDownHump"] = 8782,
3758    ["Bumpeq"] = 8782,
3759    ["bumpe"] = 8783,
3760    ["HumpEqual"] = 8783,
3761    ["bumpeq"] = 8783,
3762    ["esdot"] = 8784,
3763    ["DotEqual"] = 8784,
3764    ["doteq"] = 8784,
3765    ["eDot"] = 8785,
```

```
["doteqdot"] = 8785,
["efDot"] = 8786,
["fallingdotseq"] = 8786,
["erDot"] = 8787,
["risingdotseq"] = 8787,
["colone"] = 8788,
["coloneq"] = 8788,
["Assign"] = 8788,
["ecolon"] = 8789,
["eqcolon"] = 8789,
["ecir"] = 8790,
["eqcirc"] = 8790,
["cire"] = 8791,
["circeq"] = 8791,
["wedgeq"] = 8793,
["veeeq"] = 8794,
["trie"] = 8796,
["triangleq"] = 8796,
["equest"] = 8799,
["questeq"] = 8799,
["ne"] = 8800,
["NotEqual"] = 8800,
["equiv"] = 8801,
["Congruent"] = 8801,
["nequiv"] = 8802,
["NotCongruent"] = 8802,
["le"] = 8804,
["leq"] = 8804,
["ge"] = 8805,
["GreaterEqual"] = 8805,
["geq"] = 8805,
["lE"] = 8806,
["LessFullEqual"] = 8806,
["leqq"] = 8806,
["gE"] = 8807,
["GreaterFullEqual"] = 8807,
["geqq"] = 8807,
["lnE"] = 8808,
["lneqq"] = 8808,
["gnE"] = 8809,
["gneqq"] = 8809,
["Lt"] = 8810,
["NestedLessLess"] = 8810,
["ll"] = 8810,
["Gt"] = 8811,
["NestedGreaterGreater"] = 8811,
["gg"] = 8811,
```

```
3813    ["twixt"] = 8812,
3814    ["between"] = 8812,
3815    ["NotCupCap"] = 8813,
3816    ["nlt"] = 8814,
3817    ["NotLess"] = 8814,
3818    ["nless"] = 8814,
3819    ["ngt"] = 8815,
3820    ["NotGreater"] = 8815,
3821    ["ngtr"] = 8815,
3822    ["nle"] = 8816,
3823    ["NotLessEqual"] = 8816,
3824    ["nleq"] = 8816,
3825    ["nge"] = 8817,
3826    ["NotGreaterEqual"] = 8817,
3827    ["ngeq"] = 8817,
3828    ["lsim"] = 8818,
3829    ["LessTilde"] = 8818,
3830    ["lesssim"] = 8818,
3831    ["gsim"] = 8819,
3832    ["gtrsim"] = 8819,
3833    ["GreaterTilde"] = 8819,
3834    ["nlsim"] = 8820,
3835    ["NotLessTilde"] = 8820,
3836    ["ngsim"] = 8821,
3837    ["NotGreaterTilde"] = 8821,
3838    ["lg"] = 8822,
3839    ["lessgtr"] = 8822,
3840    ["LessGreater"] = 8822,
3841    ["gl"] = 8823,
3842    ["gtrless"] = 8823,
3843    ["GreaterLess"] = 8823,
3844    ["ntlg"] = 8824,
3845    ["NotLessGreater"] = 8824,
3846    ["ntgl"] = 8825,
3847    ["NotGreaterLess"] = 8825,
3848    ["pr"] = 8826,
3849    ["Precedes"] = 8826,
3850    ["prec"] = 8826,
3851    ["sc"] = 8827,
3852    ["Succeeds"] = 8827,
3853    ["succ"] = 8827,
3854    ["prcue"] = 8828,
3855    ["PrecedesSlantEqual"] = 8828,
3856    ["preccurlyeq"] = 8828,
3857    ["sccue"] = 8829,
3858    ["SucceedsSlantEqual"] = 8829,
3859    ["succcurlyeq"] = 8829,
```

```
3860    ["prsim"] = 8830,
3861    ["precsim"] = 8830,
3862    ["PrecedesTilde"] = 8830,
3863    ["scsim"] = 8831,
3864    ["succsim"] = 8831,
3865    ["SucceedsTilde"] = 8831,
3866    ["npr"] = 8832,
3867    ["nprec"] = 8832,
3868    ["NotPrecedes"] = 8832,
3869    ["nsc"] = 8833,
3870    ["nsucc"] = 8833,
3871    ["NotSucceeds"] = 8833,
3872    ["sub"] = 8834,
3873    ["subset"] = 8834,
3874    ["sup"] = 8835,
3875    ["supset"] = 8835,
3876    ["Superset"] = 8835,
3877    ["nsub"] = 8836,
3878    ["nsup"] = 8837,
3879    ["sube"] = 8838,
3880    ["SubsetEqual"] = 8838,
3881    ["subseteq"] = 8838,
3882    ["supe"] = 8839,
3883    ["supseteq"] = 8839,
3884    ["SupersetEqual"] = 8839,
3885    ["nsube"] = 8840,
3886    ["nsubseteq"] = 8840,
3887    ["NotSubsetEqual"] = 8840,
3888    ["nsupe"] = 8841,
3889    ["nsupseteq"] = 8841,
3890    ["NotSupersetEqual"] = 8841,
3891    ["subne"] = 8842,
3892    ["subsetneq"] = 8842,
3893    ["supne"] = 8843,
3894    ["supsetneq"] = 8843,
3895    ["cupdot"] = 8845,
3896    ["uplus"] = 8846,
3897    ["UnionPlus"] = 8846,
3898    ["sqsub"] = 8847,
3899    ["SquareSubset"] = 8847,
3900    ["sqsubset"] = 8847,
3901    ["sqsup"] = 8848,
3902    ["SquareSuperset"] = 8848,
3903    ["sqsupset"] = 8848,
3904    ["sqsube"] = 8849,
3905    ["SquareSubsetEqual"] = 8849,
3906    ["sqsubseteq"] = 8849,
```

```
3907    ["sqsupe"] = 8850,
3908    ["SquareSupersetEqual"] = 8850,
3909    ["sqsupseteq"] = 8850,
3910    ["sqcap"] = 8851,
3911    ["SquareIntersection"] = 8851,
3912    ["sqcup"] = 8852,
3913    ["SquareUnion"] = 8852,
3914    ["oplus"] = 8853,
3915    ["CirclePlus"] = 8853,
3916    ["ominus"] = 8854,
3917    ["CircleMinus"] = 8854,
3918    ["otimes"] = 8855,
3919    ["CircleTimes"] = 8855,
3920    ["osol"] = 8856,
3921    ["odot"] = 8857,
3922    ["CircleDot"] = 8857,
3923    ["ocir"] = 8858,
3924    ["circledcirc"] = 8858,
3925    ["oast"] = 8859,
3926    ["circledast"] = 8859,
3927    ["odash"] = 8861,
3928    ["circleddash"] = 8861,
3929    ["plusb"] = 8862,
3930    ["boxplus"] = 8862,
3931    ["minusb"] = 8863,
3932    ["boxminus"] = 8863,
3933    ["timesb"] = 8864,
3934    ["boxtimes"] = 8864,
3935    ["sdotb"] = 8865,
3936    ["dotsquare"] = 8865,
3937    ["vdash"] = 8866,
3938    ["RightTee"] = 8866,
3939    ["dashv"] = 8867,
3940    ["LeftTee"] = 8867,
3941    ["top"] = 8868,
3942    ["DownTee"] = 8868,
3943    ["bottom"] = 8869,
3944    ["bot"] = 8869,
3945    ["perp"] = 8869,
3946    ["UpTee"] = 8869,
3947    ["models"] = 8871,
3948    ["vDash"] = 8872,
3949    ["DoubleRightTee"] = 8872,
3950    ["Vdash"] = 8873,
3951    ["Vvdash"] = 8874,
3952    ["VDash"] = 8875,
3953    ["nvdash"] = 8876,
```

```
["nvDash"] = 8877,
["nVdash"] = 8878,
["nVDash"] = 8879,
["prurel"] = 8880,
["vltri"] = 8882,
["vartriangleleft"] = 8882,
["LeftTriangle"] = 8882,
["vrtri"] = 8883,
["vartriangleright"] = 8883,
["RightTriangle"] = 8883,
["ltrie"] = 8884,
["trianglelefteq"] = 8884,
["LeftTriangleEqual"] = 8884,
["rtrie"] = 8885,
["trianglerighteq"] = 8885,
["RightTriangleEqual"] = 8885,
["origof"] = 8886,
["imof"] = 8887,
["mumap"] = 8888,
["multimap"] = 8888,
["hercon"] = 8889,
["intcal"] = 8890,
["intercal"] = 8890,
["veebar"] = 8891,
["barvee"] = 8893,
["angrtvb"] = 8894,
["lrtri"] = 8895,
["xwedge"] = 8896,
["Wedge"] = 8896,
["bigwedge"] = 8896,
["xvee"] = 8897,
["Vee"] = 8897,
["bigvee"] = 8897,
["xcap"] = 8898,
["Intersection"] = 8898,
["bigcap"] = 8898,
["xcup"] = 8899,
["Union"] = 8899,
["bigcup"] = 8899,
["diam"] = 8900,
["diamond"] = 8900,
["Diamond"] = 8900,
["sdot"] = 8901,
["sstarf"] = 8902,
["Star"] = 8902,
["divonx"] = 8903,
["divideontimes"] = 8903,
```

```
["bowtie"] = 8904,
["ltimes"] = 8905,
["rtimes"] = 8906,
["lthree"] = 8907,
["leftthreetimes"] = 8907,
["rthree"] = 8908,
["rightthreetimes"] = 8908,
["bsime"] = 8909,
["backsimeq"] = 8909,
["cuvee"] = 8910,
["curlyvee"] = 8910,
["cuwed"] = 8911,
["curlywedge"] = 8911,
["Sub"] = 8912,
["Subset"] = 8912,
["Sup"] = 8913,
["Supset"] = 8913,
["Cap"] = 8914,
["Cup"] = 8915,
["fork"] = 8916,
["pitchfork"] = 8916,
["epar"] = 8917,
["ltdot"] = 8918,
["lessdot"] = 8918,
["gtdot"] = 8919,
["gtrdot"] = 8919,
["Ll"] = 8920,
["Gg"] = 8921,
["ggg"] = 8921,
["leg"] = 8922,
["LessEqualGreater"] = 8922,
["lesseqgtr"] = 8922,
["gel"] = 8923,
["gtreqless"] = 8923,
["GreaterEqualLess"] = 8923,
["cuepr"] = 8926,
["curlyeqprec"] = 8926,
["cuesc"] = 8927,
["curlyeqsucc"] = 8927,
["nprcue"] = 8928,
["NotPrecedesSlantEqual"] = 8928,
["nsccue"] = 8929,
["NotSucceedsSlantEqual"] = 8929,
["nsqsube"] = 8930,
["NotSquareSubsetEqual"] = 8930,
["nsqsupe"] = 8931,
["NotSquareSupersetEqual"] = 8931,
```

```
["lnsim"] = 8934,
["gnsim"] = 8935,
["prnsim"] = 8936,
["precnsim"] = 8936,
["scnsim"] = 8937,
["succnsim"] = 8937,
["nltri"] = 8938,
["ntriangleleft"] = 8938,
["NotLeftTriangle"] = 8938,
["nrtri"] = 8939,
["ntriangleright"] = 8939,
["NotRightTriangle"] = 8939,
["nltrie"] = 8940,
["ntrianglelefteq"] = 8940,
["NotLeftTriangleEqual"] = 8940,
["nrtrie"] = 8941,
["ntrianglerighteq"] = 8941,
["NotRightTriangleEqual"] = 8941,
["vellip"] = 8942,
["ctdot"] = 8943,
["utdot"] = 8944,
["dtdot"] = 8945,
["disin"] = 8946,
["isinsv"] = 8947,
["isins"] = 8948,
["isindot"] = 8949,
["notinvc"] = 8950,
["notinvb"] = 8951,
["isinE"] = 8953,
["nisd"] = 8954,
["xnis"] = 8955,
["nis"] = 8956,
["notnivc"] = 8957,
["notnivb"] = 8958,
["barwed"] = 8965,
["barwedge"] = 8965,
["Barwed"] = 8966,
["doublebarwedge"] = 8966,
["lceil"] = 8968,
["LeftCeiling"] = 8968,
["rceil"] = 8969,
["RightCeiling"] = 8969,
["lfloor"] = 8970,
["LeftFloor"] = 8970,
["rfloor"] = 8971,
["RightFloor"] = 8971,
["drcrop"] = 8972,
```

```
  ["dlcrop"] = 8973,
  ["urcrop"] = 8974,
  ["ulcrop"] = 8975,
  ["bnot"] = 8976,
  ["profline"] = 8978,
  ["profsurf"] = 8979,
  ["telrec"] = 8981,
  ["target"] = 8982,
  ["ulcorn"] = 8988,
  ["ulcorner"] = 8988,
  ["urcorn"] = 8989,
  ["urcorner"] = 8989,
  ["dlcorn"] = 8990,
  ["llcorner"] = 8990,
  ["drcorn"] = 8991,
  ["lrcorner"] = 8991,
  ["frown"] = 8994,
  ["sfrown"] = 8994,
  ["smile"] = 8995,
  ["ssmile"] = 8995,
  ["cylcty"] = 9005,
  ["profalar"] = 9006,
  ["topbot"] = 9014,
  ["ovbar"] = 9021,
  ["solbar"] = 9023,
  ["angzarr"] = 9084,
  ["lmoust"] = 9136,
  ["lmoustache"] = 9136,
  ["rmoust"] = 9137,
  ["rmoustache"] = 9137,
  ["tbrk"] = 9140,
  ["OverBracket"] = 9140,
  ["bbrk"] = 9141,
  ["UnderBracket"] = 9141,
  ["bbrktbrk"] = 9142,
  ["OverParenthesis"] = 9180,
  ["UnderParenthesis"] = 9181,
  ["OverBrace"] = 9182,
  ["UnderBrace"] = 9183,
  ["trpezium"] = 9186,
  ["elinters"] = 9191,
  ["blank"] = 9251,
  ["oS"] = 9416,
  ["circledS"] = 9416,
  ["boxh"] = 9472,
  ["HorizontalLine"] = 9472,
  ["boxv"] = 9474,
```

```
  ["boxdr"] = 9484,
  ["boxdl"] = 9488,
  ["boxur"] = 9492,
  ["boxul"] = 9496,
  ["boxvr"] = 9500,
  ["boxvl"] = 9508,
  ["boxhd"] = 9516,
  ["boxhu"] = 9524,
  ["boxvh"] = 9532,
  ["boxH"] = 9552,
  ["boxV"] = 9553,
  ["boxdR"] = 9554,
  ["boxDr"] = 9555,
  ["boxDR"] = 9556,
  ["boxdL"] = 9557,
  ["boxDl"] = 9558,
  ["boxDL"] = 9559,
  ["boxuR"] = 9560,
  ["boxUr"] = 9561,
  ["boxUR"] = 9562,
  ["boxuL"] = 9563,
  ["boxUl"] = 9564,
  ["boxUL"] = 9565,
  ["boxvR"] = 9566,
  ["boxVr"] = 9567,
  ["boxVR"] = 9568,
  ["boxvL"] = 9569,
  ["boxVl"] = 9570,
  ["boxVL"] = 9571,
  ["boxHd"] = 9572,
  ["boxhD"] = 9573,
  ["boxHD"] = 9574,
  ["boxHu"] = 9575,
  ["boxhU"] = 9576,
  ["boxHU"] = 9577,
  ["boxvH"] = 9578,
  ["boxVh"] = 9579,
  ["boxVH"] = 9580,
  ["uhblk"] = 9600,
  ["lhblk"] = 9604,
  ["block"] = 9608,
  ["blk14"] = 9617,
  ["blk12"] = 9618,
  ["blk34"] = 9619,
  ["squ"] = 9633,
  ["square"] = 9633,
  ["Square"] = 9633,
```

```
  ["squf"] = 9642,
  ["squarf"] = 9642,
  ["blacksquare"] = 9642,
  ["FilledVerySmallSquare"] = 9642,
  ["EmptyVerySmallSquare"] = 9643,
  ["rect"] = 9645,
  ["marker"] = 9646,
  ["fltns"] = 9649,
  ["xutri"] = 9651,
  ["bigtriangleup"] = 9651,
  ["utrif"] = 9652,
  ["blacktriangle"] = 9652,
  ["utri"] = 9653,
  ["triangle"] = 9653,
  ["rtrif"] = 9656,
  ["blacktriangleright"] = 9656,
  ["rtri"] = 9657,
  ["triangleright"] = 9657,
  ["xdtri"] = 9661,
  ["bigtriangledown"] = 9661,
  ["dtrif"] = 9662,
  ["blacktriangledown"] = 9662,
  ["dtri"] = 9663,
  ["triangledown"] = 9663,
  ["ltrif"] = 9666,
  ["blacktriangleleft"] = 9666,
  ["ltri"] = 9667,
  ["triangleleft"] = 9667,
  ["loz"] = 9674,
  ["lozenge"] = 9674,
  ["cir"] = 9675,
  ["tridot"] = 9708,
  ["xcirc"] = 9711,
  ["bigcirc"] = 9711,
  ["ultri"] = 9720,
  ["urtri"] = 9721,
  ["lltri"] = 9722,
  ["EmptySmallSquare"] = 9723,
  ["FilledSmallSquare"] = 9724,
  ["starf"] = 9733,
  ["bigstar"] = 9733,
  ["star"] = 9734,
  ["phone"] = 9742,
  ["female"] = 9792,
  ["male"] = 9794,
  ["spades"] = 9824,
  ["spadesuit"] = 9824,
```

```
4236  ["clubs"] = 9827,
4237  ["clubsuit"] = 9827,
4238  ["hearts"] = 9829,
4239  ["heartsuit"] = 9829,
4240  ["diams"] = 9830,
4241  ["diamondsuit"] = 9830,
4242  ["sung"] = 9834,
4243  ["flat"] = 9837,
4244  ["natur"] = 9838,
4245  ["natural"] = 9838,
4246  ["sharp"] = 9839,
4247  ["check"] = 10003,
4248  ["checkmark"] = 10003,
4249  ["cross"] = 10007,
4250  ["malt"] = 10016,
4251  ["maltese"] = 10016,
4252  ["sext"] = 10038,
4253  ["VerticalSeparator"] = 10072,
4254  ["lbbrk"] = 10098,
4255  ["rbbrk"] = 10099,
4256  ["lobrk"] = 10214,
4257  ["LeftDoubleBracket"] = 10214,
4258  ["robrk"] = 10215,
4259  ["RightDoubleBracket"] = 10215,
4260  ["lang"] = 10216,
4261  ["LeftAngleBracket"] = 10216,
4262  ["langle"] = 10216,
4263  ["rang"] = 10217,
4264  ["RightAngleBracket"] = 10217,
4265  ["rangle"] = 10217,
4266  ["Lang"] = 10218,
4267  ["Rang"] = 10219,
4268  ["loang"] = 10220,
4269  ["roang"] = 10221,
4270  ["xlarr"] = 10229,
4271  ["longleftarrow"] = 10229,
4272  ["LongLeftArrow"] = 10229,
4273  ["xrarr"] = 10230,
4274  ["longrightarrow"] = 10230,
4275  ["LongRightArrow"] = 10230,
4276  ["xharr"] = 10231,
4277  ["longleftrightarrow"] = 10231,
4278  ["LongLeftRightArrow"] = 10231,
4279  ["xlArr"] = 10232,
4280  ["Longleftarrow"] = 10232,
4281  ["DoubleLongLeftArrow"] = 10232,
4282  ["xrArr"] = 10233,
```

```
4283  ["Longrightarrow"] = 10233,
4284  ["DoubleLongRightArrow"] = 10233,
4285  ["xhArr"] = 10234,
4286  ["Longleftrightarrow"] = 10234,
4287  ["DoubleLongLeftRightArrow"] = 10234,
4288  ["xmap"] = 10236,
4289  ["longmapsto"] = 10236,
4290  ["dzigrarr"] = 10239,
4291  ["nvlArr"] = 10498,
4292  ["nvrArr"] = 10499,
4293  ["nvHarr"] = 10500,
4294  ["Map"] = 10501,
4295  ["lbarr"] = 10508,
4296  ["rbarr"] = 10509,
4297  ["bkarow"] = 10509,
4298  ["lBarr"] = 10510,
4299  ["rBarr"] = 10511,
4300  ["dbkarow"] = 10511,
4301  ["RBarr"] = 10512,
4302  ["drbkarow"] = 10512,
4303  ["DDotrahd"] = 10513,
4304  ["UpArrowBar"] = 10514,
4305  ["DownArrowBar"] = 10515,
4306  ["Rarrtl"] = 10518,
4307  ["latail"] = 10521,
4308  ["ratail"] = 10522,
4309  ["lAtail"] = 10523,
4310  ["rAtail"] = 10524,
4311  ["larrfs"] = 10525,
4312  ["rarrfs"] = 10526,
4313  ["larrbfs"] = 10527,
4314  ["rarrbfs"] = 10528,
4315  ["nwarhk"] = 10531,
4316  ["nearhk"] = 10532,
4317  ["searhk"] = 10533,
4318  ["hksearow"] = 10533,
4319  ["swarhk"] = 10534,
4320  ["hkswarow"] = 10534,
4321  ["nwnear"] = 10535,
4322  ["nesear"] = 10536,
4323  ["toea"] = 10536,
4324  ["seswar"] = 10537,
4325  ["tosa"] = 10537,
4326  ["swnwar"] = 10538,
4327  ["rarrc"] = 10547,
4328  ["cudarrr"] = 10549,
4329  ["ldca"] = 10550,
```

```
["rdca"] = 10551,
["cudarrl"] = 10552,
["larrpl"] = 10553,
["curarrm"] = 10556,
["cularrp"] = 10557,
["rarrpl"] = 10565,
["harrcir"] = 10568,
["Uarrocir"] = 10569,
["lurdshar"] = 10570,
["ldrushar"] = 10571,
["LeftRightVector"] = 10574,
["RightUpDownVector"] = 10575,
["DownLeftRightVector"] = 10576,
["LeftUpDownVector"] = 10577,
["LeftVectorBar"] = 10578,
["RightVectorBar"] = 10579,
["RightUpVectorBar"] = 10580,
["RightDownVectorBar"] = 10581,
["DownLeftVectorBar"] = 10582,
["DownRightVectorBar"] = 10583,
["LeftUpVectorBar"] = 10584,
["LeftDownVectorBar"] = 10585,
["LeftTeeVector"] = 10586,
["RightTeeVector"] = 10587,
["RightUpTeeVector"] = 10588,
["RightDownTeeVector"] = 10589,
["DownLeftTeeVector"] = 10590,
["DownRightTeeVector"] = 10591,
["LeftUpTeeVector"] = 10592,
["LeftDownTeeVector"] = 10593,
["lHar"] = 10594,
["uHar"] = 10595,
["rHar"] = 10596,
["dHar"] = 10597,
["luruhar"] = 10598,
["ldrdhar"] = 10599,
["ruluhar"] = 10600,
["rdldhar"] = 10601,
["lharul"] = 10602,
["llhard"] = 10603,
["rharul"] = 10604,
["lrhard"] = 10605,
["udhar"] = 10606,
["UpEquilibrium"] = 10606,
["duhar"] = 10607,
["ReverseUpEquilibrium"] = 10607,
["RoundImplies"] = 10608,
```

```
4377    ["erarr"] = 10609,
4378    ["simrarr"] = 10610,
4379    ["larrsim"] = 10611,
4380    ["rarrsim"] = 10612,
4381    ["rarrap"] = 10613,
4382    ["ltlarr"] = 10614,
4383    ["gtrarr"] = 10616,
4384    ["subrarr"] = 10617,
4385    ["suplarr"] = 10619,
4386    ["lfisht"] = 10620,
4387    ["rfisht"] = 10621,
4388    ["ufisht"] = 10622,
4389    ["dfisht"] = 10623,
4390    ["lopar"] = 10629,
4391    ["ropar"] = 10630,
4392    ["lbrke"] = 10635,
4393    ["rbrke"] = 10636,
4394    ["lbrkslu"] = 10637,
4395    ["rbrksld"] = 10638,
4396    ["lbrksld"] = 10639,
4397    ["rbrkslu"] = 10640,
4398    ["langd"] = 10641,
4399    ["rangd"] = 10642,
4400    ["lparlt"] = 10643,
4401    ["rpargt"] = 10644,
4402    ["gtlPar"] = 10645,
4403    ["ltrPar"] = 10646,
4404    ["vzigzag"] = 10650,
4405    ["vangrt"] = 10652,
4406    ["angrtvbd"] = 10653,
4407    ["ange"] = 10660,
4408    ["range"] = 10661,
4409    ["dwangle"] = 10662,
4410    ["uwangle"] = 10663,
4411    ["angmsdaa"] = 10664,
4412    ["angmsdab"] = 10665,
4413    ["angmsdac"] = 10666,
4414    ["angmsdad"] = 10667,
4415    ["angmsdae"] = 10668,
4416    ["angmsdaf"] = 10669,
4417    ["angmsdag"] = 10670,
4418    ["angmsdah"] = 10671,
4419    ["bemptyv"] = 10672,
4420    ["demptyv"] = 10673,
4421    ["cemptyv"] = 10674,
4422    ["raemptyv"] = 10675,
4423    ["laemptyv"] = 10676,
```

```
4424  ["ohbar"] = 10677,
4425  ["omid"] = 10678,
4426  ["opar"] = 10679,
4427  ["operp"] = 10681,
4428  ["olcross"] = 10683,
4429  ["odsold"] = 10684,
4430  ["olcir"] = 10686,
4431  ["ofcir"] = 10687,
4432  ["olt"] = 10688,
4433  ["ogt"] = 10689,
4434  ["cirscir"] = 10690,
4435  ["cirE"] = 10691,
4436  ["solb"] = 10692,
4437  ["bsolb"] = 10693,
4438  ["boxbox"] = 10697,
4439  ["trisb"] = 10701,
4440  ["rtriltri"] = 10702,
4441  ["LeftTriangleBar"] = 10703,
4442  ["RightTriangleBar"] = 10704,
4443  ["race"] = 10714,
4444  ["iinfin"] = 10716,
4445  ["infintie"] = 10717,
4446  ["nvinfin"] = 10718,
4447  ["eparsl"] = 10723,
4448  ["smeparsl"] = 10724,
4449  ["eqvparsl"] = 10725,
4450  ["lozf"] = 10731,
4451  ["blacklozenge"] = 10731,
4452  ["RuleDelayed"] = 10740,
4453  ["dsol"] = 10742,
4454  ["xodot"] = 10752,
4455  ["bigodot"] = 10752,
4456  ["xoplus"] = 10753,
4457  ["bigoplus"] = 10753,
4458  ["xotime"] = 10754,
4459  ["bigotimes"] = 10754,
4460  ["xuplus"] = 10756,
4461  ["biguplus"] = 10756,
4462  ["xsqcup"] = 10758,
4463  ["bigsqcup"] = 10758,
4464  ["qint"] = 10764,
4465  ["iiiint"] = 10764,
4466  ["fpartint"] = 10765,
4467  ["cirfnint"] = 10768,
4468  ["awint"] = 10769,
4469  ["rppolint"] = 10770,
4470  ["scpolint"] = 10771,
```

```
4471    ["npolint"] = 10772,
4472    ["pointint"] = 10773,
4473    ["quatint"] = 10774,
4474    ["intlarhk"] = 10775,
4475    ["pluscir"] = 10786,
4476    ["plusacir"] = 10787,
4477    ["simplus"] = 10788,
4478    ["plusdu"] = 10789,
4479    ["plussim"] = 10790,
4480    ["plustwo"] = 10791,
4481    ["mcomma"] = 10793,
4482    ["minusdu"] = 10794,
4483    ["loplus"] = 10797,
4484    ["roplus"] = 10798,
4485    ["Cross"] = 10799,
4486    ["timesd"] = 10800,
4487    ["timesbar"] = 10801,
4488    ["smashp"] = 10803,
4489    ["lotimes"] = 10804,
4490    ["rotimes"] = 10805,
4491    ["otimesas"] = 10806,
4492    ["Otimes"] = 10807,
4493    ["odiv"] = 10808,
4494    ["triplus"] = 10809,
4495    ["triminus"] = 10810,
4496    ["tritime"] = 10811,
4497    ["iprod"] = 10812,
4498    ["intprod"] = 10812,
4499    ["amalg"] = 10815,
4500    ["capdot"] = 10816,
4501    ["ncup"] = 10818,
4502    ["ncap"] = 10819,
4503    ["capand"] = 10820,
4504    ["cupor"] = 10821,
4505    ["cupcap"] = 10822,
4506    ["capcup"] = 10823,
4507    ["cupbrcap"] = 10824,
4508    ["capbrcup"] = 10825,
4509    ["cupcup"] = 10826,
4510    ["capcap"] = 10827,
4511    ["ccups"] = 10828,
4512    ["ccaps"] = 10829,
4513    ["ccupssm"] = 10832,
4514    ["And"] = 10835,
4515    ["Or"] = 10836,
4516    ["andand"] = 10837,
4517    ["oror"] = 10838,
```

```
4518  ["orslope"] = 10839,
4519  ["andslope"] = 10840,
4520  ["andv"] = 10842,
4521  ["orv"] = 10843,
4522  ["andd"] = 10844,
4523  ["ord"] = 10845,
4524  ["wedbar"] = 10847,
4525  ["sdote"] = 10854,
4526  ["simdot"] = 10858,
4527  ["congdot"] = 10861,
4528  ["easter"] = 10862,
4529  ["apacir"] = 10863,
4530  ["apE"] = 10864,
4531  ["eplus"] = 10865,
4532  ["pluse"] = 10866,
4533  ["Esim"] = 10867,
4534  ["Colone"] = 10868,
4535  ["Equal"] = 10869,
4536  ["eDDot"] = 10871,
4537  ["ddotseq"] = 10871,
4538  ["equivDD"] = 10872,
4539  ["ltcir"] = 10873,
4540  ["gtcir"] = 10874,
4541  ["ltquest"] = 10875,
4542  ["gtquest"] = 10876,
4543  ["les"] = 10877,
4544  ["LessSlantEqual"] = 10877,
4545  ["leqslant"] = 10877,
4546  ["ges"] = 10878,
4547  ["GreaterSlantEqual"] = 10878,
4548  ["geqslant"] = 10878,
4549  ["lesdot"] = 10879,
4550  ["gesdot"] = 10880,
4551  ["lesdoto"] = 10881,
4552  ["gesdoto"] = 10882,
4553  ["lesdotor"] = 10883,
4554  ["gesdotol"] = 10884,
4555  ["lap"] = 10885,
4556  ["lessapprox"] = 10885,
4557  ["gap"] = 10886,
4558  ["gtrapprox"] = 10886,
4559  ["lne"] = 10887,
4560  ["lneq"] = 10887,
4561  ["gne"] = 10888,
4562  ["gneq"] = 10888,
4563  ["lnap"] = 10889,
4564  ["lnapprox"] = 10889,
```

```
4565    ["gnap"] = 10890,
4566    ["gnapprox"] = 10890,
4567    ["lEg"] = 10891,
4568    ["lesseqqgtr"] = 10891,
4569    ["gEl"] = 10892,
4570    ["gtreqqless"] = 10892,
4571    ["lsime"] = 10893,
4572    ["gsime"] = 10894,
4573    ["lsimg"] = 10895,
4574    ["gsiml"] = 10896,
4575    ["lgE"] = 10897,
4576    ["glE"] = 10898,
4577    ["lesges"] = 10899,
4578    ["gesles"] = 10900,
4579    ["els"] = 10901,
4580    ["eqslantless"] = 10901,
4581    ["egs"] = 10902,
4582    ["eqslantgtr"] = 10902,
4583    ["elsdot"] = 10903,
4584    ["egsdot"] = 10904,
4585    ["el"] = 10905,
4586    ["eg"] = 10906,
4587    ["siml"] = 10909,
4588    ["simg"] = 10910,
4589    ["simlE"] = 10911,
4590    ["simgE"] = 10912,
4591    ["LessLess"] = 10913,
4592    ["GreaterGreater"] = 10914,
4593    ["glj"] = 10916,
4594    ["gla"] = 10917,
4595    ["ltcc"] = 10918,
4596    ["gtcc"] = 10919,
4597    ["lescc"] = 10920,
4598    ["gescc"] = 10921,
4599    ["smt"] = 10922,
4600    ["lat"] = 10923,
4601    ["smte"] = 10924,
4602    ["late"] = 10925,
4603    ["bumpE"] = 10926,
4604    ["pre"] = 10927,
4605    ["preceq"] = 10927,
4606    ["PrecedesEqual"] = 10927,
4607    ["sce"] = 10928,
4608    ["succeq"] = 10928,
4609    ["SucceedsEqual"] = 10928,
4610    ["prE"] = 10931,
4611    ["scE"] = 10932,
```

```
4612  ["prnE"] = 10933,
4613  ["precneqq"] = 10933,
4614  ["scnE"] = 10934,
4615  ["succneqq"] = 10934,
4616  ["prap"] = 10935,
4617  ["precapprox"] = 10935,
4618  ["scap"] = 10936,
4619  ["succapprox"] = 10936,
4620  ["prnap"] = 10937,
4621  ["precnapprox"] = 10937,
4622  ["scnap"] = 10938,
4623  ["succnapprox"] = 10938,
4624  ["Pr"] = 10939,
4625  ["Sc"] = 10940,
4626  ["subdot"] = 10941,
4627  ["supdot"] = 10942,
4628  ["subplus"] = 10943,
4629  ["supplus"] = 10944,
4630  ["submult"] = 10945,
4631  ["supmult"] = 10946,
4632  ["subedot"] = 10947,
4633  ["supedot"] = 10948,
4634  ["subE"] = 10949,
4635  ["subseteqq"] = 10949,
4636  ["supE"] = 10950,
4637  ["supseteqq"] = 10950,
4638  ["subsim"] = 10951,
4639  ["supsim"] = 10952,
4640  ["subnE"] = 10955,
4641  ["subsetneqq"] = 10955,
4642  ["supnE"] = 10956,
4643  ["supsetneqq"] = 10956,
4644  ["csub"] = 10959,
4645  ["csup"] = 10960,
4646  ["csube"] = 10961,
4647  ["csupe"] = 10962,
4648  ["subsup"] = 10963,
4649  ["supsub"] = 10964,
4650  ["subsub"] = 10965,
4651  ["supsup"] = 10966,
4652  ["suphsub"] = 10967,
4653  ["supdsub"] = 10968,
4654  ["forkv"] = 10969,
4655  ["topfork"] = 10970,
4656  ["mlcp"] = 10971,
4657  ["Dashv"] = 10980,
4658  ["DoubleLeftTee"] = 10980,
```

```
  ["Vdashl"] = 10982,
  ["Barv"] = 10983,
  ["vBar"] = 10984,
  ["vBarv"] = 10985,
  ["Vbar"] = 10987,
  ["Not"] = 10988,
  ["bNot"] = 10989,
  ["rnmid"] = 10990,
  ["cirmid"] = 10991,
  ["midcir"] = 10992,
  ["topcir"] = 10993,
  ["nhpar"] = 10994,
  ["parsim"] = 10995,
  ["parsl"] = 11005,
  ["fflig"] = 64256,
  ["filig"] = 64257,
  ["fllig"] = 64258,
  ["ffilig"] = 64259,
  ["ffllig"] = 64260,
  ["Ascr"] = 119964,
  ["Cscr"] = 119966,
  ["Dscr"] = 119967,
  ["Gscr"] = 119970,
  ["Jscr"] = 119973,
  ["Kscr"] = 119974,
  ["Nscr"] = 119977,
  ["Oscr"] = 119978,
  ["Pscr"] = 119979,
  ["Qscr"] = 119980,
  ["Sscr"] = 119982,
  ["Tscr"] = 119983,
  ["Uscr"] = 119984,
  ["Vscr"] = 119985,
  ["Wscr"] = 119986,
  ["Xscr"] = 119987,
  ["Yscr"] = 119988,
  ["Zscr"] = 119989,
  ["ascr"] = 119990,
  ["bscr"] = 119991,
  ["cscr"] = 119992,
  ["dscr"] = 119993,
  ["fscr"] = 119995,
  ["hscr"] = 119997,
  ["iscr"] = 119998,
  ["jscr"] = 119999,
  ["kscr"] = 120000,
  ["lscr"] = 120001,
```

```
  ["mscr"] = 120002,
  ["nscr"] = 120003,
  ["pscr"] = 120005,
  ["qscr"] = 120006,
  ["rscr"] = 120007,
  ["sscr"] = 120008,
  ["tscr"] = 120009,
  ["uscr"] = 120010,
  ["vscr"] = 120011,
  ["wscr"] = 120012,
  ["xscr"] = 120013,
  ["yscr"] = 120014,
  ["zscr"] = 120015,
  ["Afr"] = 120068,
  ["Bfr"] = 120069,
  ["Dfr"] = 120071,
  ["Efr"] = 120072,
  ["Ffr"] = 120073,
  ["Gfr"] = 120074,
  ["Jfr"] = 120077,
  ["Kfr"] = 120078,
  ["Lfr"] = 120079,
  ["Mfr"] = 120080,
  ["Nfr"] = 120081,
  ["Ofr"] = 120082,
  ["Pfr"] = 120083,
  ["Qfr"] = 120084,
  ["Sfr"] = 120086,
  ["Tfr"] = 120087,
  ["Ufr"] = 120088,
  ["Vfr"] = 120089,
  ["Wfr"] = 120090,
  ["Xfr"] = 120091,
  ["Yfr"] = 120092,
  ["afr"] = 120094,
  ["bfr"] = 120095,
  ["cfr"] = 120096,
  ["dfr"] = 120097,
  ["efr"] = 120098,
  ["ffr"] = 120099,
  ["gfr"] = 120100,
  ["hfr"] = 120101,
  ["ifr"] = 120102,
  ["jfr"] = 120103,
  ["kfr"] = 120104,
  ["lfr"] = 120105,
  ["mfr"] = 120106,
```

```
4753    ["nfr"] = 120107,
4754    ["ofr"] = 120108,
4755    ["pfr"] = 120109,
4756    ["qfr"] = 120110,
4757    ["rfr"] = 120111,
4758    ["sfr"] = 120112,
4759    ["tfr"] = 120113,
4760    ["ufr"] = 120114,
4761    ["vfr"] = 120115,
4762    ["wfr"] = 120116,
4763    ["xfr"] = 120117,
4764    ["yfr"] = 120118,
4765    ["zfr"] = 120119,
4766    ["Aopf"] = 120120,
4767    ["Bopf"] = 120121,
4768    ["Dopf"] = 120123,
4769    ["Eopf"] = 120124,
4770    ["Fopf"] = 120125,
4771    ["Gopf"] = 120126,
4772    ["Iopf"] = 120128,
4773    ["Jopf"] = 120129,
4774    ["Kopf"] = 120130,
4775    ["Lopf"] = 120131,
4776    ["Mopf"] = 120132,
4777    ["Oopf"] = 120134,
4778    ["Sopf"] = 120138,
4779    ["Topf"] = 120139,
4780    ["Uopf"] = 120140,
4781    ["Vopf"] = 120141,
4782    ["Wopf"] = 120142,
4783    ["Xopf"] = 120143,
4784    ["Yopf"] = 120144,
4785    ["aopf"] = 120146,
4786    ["bopf"] = 120147,
4787    ["copf"] = 120148,
4788    ["dopf"] = 120149,
4789    ["eopf"] = 120150,
4790    ["fopf"] = 120151,
4791    ["gopf"] = 120152,
4792    ["hopf"] = 120153,
4793    ["iopf"] = 120154,
4794    ["jopf"] = 120155,
4795    ["kopf"] = 120156,
4796    ["lopf"] = 120157,
4797    ["mopf"] = 120158,
4798    ["nopf"] = 120159,
4799    ["oopf"] = 120160,
```

```
4800   ["popf"] = 120161,
4801   ["qopf"] = 120162,
4802   ["ropf"] = 120163,
4803   ["sopf"] = 120164,
4804   ["topf"] = 120165,
4805   ["uopf"] = 120166,
4806   ["vopf"] = 120167,
4807   ["wopf"] = 120168,
4808   ["xopf"] = 120169,
4809   ["yopf"] = 120170,
4810   ["zopf"] = 120171,
4811 }
```

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

```
4812 function entities.dec_entity(s)
4813   return unicode.utf8.char(tonumber(s))
4814 end
```

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

```
4815 function entities.hex_entity(s)
4816   return unicode.utf8.char(tonumber("0x"..s))
4817 end
```

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

```
4818 function entities.char_entity(s)
4819   local n = character_entities[s]
4820   if n == nil then
4821     return "&" .. s .. ";"
4822   end
4823   return unicode.utf8.char(n)
4824 end
```

3.1.3 Plain TEX Writer

This section documents the `writer` object, which implements the routines for producing the TEX output. The object is an amalgamate of the generic, TEX, LATEX writer objects that were located in the `lunamark/writer/generic.lua`, `lunamark/writer/tex.lua`, and `lunamark/writer/latex.lua` files in the Lunamark Lua module.

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

```
4825 M.writer = {}
```

The `writer.new` method creates and returns a new TEX 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.

```
4826 function M.writer.new(options)
4827   local self = {}
```

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

```
4828   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.

```
4829   local slice_specifiers = {}
4830   for specifier in options.slice:gmatch("[^%s]+") do
4831     table.insert(slice_specifiers, specifier)
4832   end
4833
4834   if #slice_specifiers == 2 then
4835     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
4836     local slice_begin_type = self.slice_begin:sub(1, 1)
4837     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
4838       self.slice_begin = "^" .. self.slice_begin
4839     end
4840     local slice_end_type = self.slice_end:sub(1, 1)
4841     if slice_end_type ~= "^" and slice_end_type ~= "$" then
4842       self.slice_end = "$" .. self.slice_end
4843     end
4844   elseif #slice_specifiers == 1 then
4845     self.slice_begin = "^" .. slice_specifiers[1]
4846     self.slice_end = "$" .. slice_specifiers[1]
4847   end
4848
4849   if self.slice_begin == "^" and self.slice_end ~= "^" then
4850     self.is_writing = true
4851   else
4852     self.is_writing = false
4853   end
```

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

```
4854   self.suffix = ".tex"
```

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

```
4855   self.space = " "
```

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

```
4856  self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
4857  function self.plain(s)
4858    return s
4859  end
```

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

```
4860  function self.paragraph(s)
4861    if not self.is_writing then return "" end
4862    return s
4863  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.

```
4864  function self.pack(name)
4865    return [[\input{]] .. name .. [[}\relax]]
4866  end
```

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

```
4867  function self.interblocksep()
4868    if not self.is_writing then return "" end
4869    return "\\markdownRendererInterblockSeparator\n{}"
4870  end
```

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

```
4871  self.linebreak = "\\markdownRendererLineBreak\n{}"
```

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

```
4872  self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

```
4873  function self.thematic_break()
4874    if not self.is_writing then return "" end
4875    return "\\markdownRendererThematicBreak{}"
4876  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.

```
4877  self.escaped_uri_chars = {
4878    ["{"] = "\\markdownRendererLeftBrace{}",
4879    ["}"] = "\\markdownRendererRightBrace{}",
4880    ["\\"] = "\\markdownRendererBackslash{}",
4881  }
4882  self.escaped_minimal_strings = {
```

```
    ["^^"] = "\\markdownRendererCircumflex\\markdownRendererCircumflex ",
    ["☒"] = "\\markdownRendererTickedBox{}",
    ["☐"] = "\\markdownRendererHalfTickedBox{}",
    ["▫"] = "\\markdownRendererUntickedBox{}",
    [entities.hex_entity('FFFD')] = "\\markdownRendererReplacementCharacter{}",
  }
```

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

```
  self.escaped_strings = util.table_copy(self.escaped_minimal_strings)
  self.escaped_strings[entities.hex_entity('00A0')] = self.nbsp
```

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

```
  self.escaped_chars = {
    ["{"] = "\\markdownRendererLeftBrace{}",
    ["}"] = "\\markdownRendererRightBrace{}",
    ["%"] = "\\markdownRendererPercentSign{}",
    ["\\"] = "\\markdownRendererBackslash{}",
    ["#"] = "\\markdownRendererHash{}",
    ["$"] = "\\markdownRendererDollarSign{}",
    ["&"] = "\\markdownRendererAmpersand{}",
    ["_"] = "\\markdownRendererUnderscore{}",
    ["^"] = "\\markdownRendererCircumflex{}",
    ["~"] = "\\markdownRendererTilde{}",
    ["|"] = "\\markdownRendererPipe{}",
    [entities.hex_entity('0000')] = "\\markdownRendererReplacementCharacter{}",
  }
```

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

```
  self.escape = util.escaper(self.escaped_chars, self.escaped_strings)
  self.escape_uri = util.escaper(self.escaped_uri_chars, self.escaped_minimal_strings)
  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.

```
  if options.hybrid then
    self.string = self.escape_minimal
    self.uri = self.escape_minimal
  else
    self.string = self.escape
    self.uri = self.escape_uri
```

```
4914    end
```

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

```
4915    function self.code(s)
4916      return {"\\markdownRendererCodeSpan{",self.escape(s),"}"}
4917    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.

```
4918    function self.link(lab,src,tit)
4919      return {"\\markdownRendererLink{",lab,"}",
4920                              "{",self.escape(src),"}",
4921                              "{",self.uri(src),"}",
4922                              "{",self.string(tit or ""),"}"}
4923    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.

```
4924    function self.image(lab,src,tit)
4925      return {"\\markdownRendererImage{",lab,"}",
4926                               "{",self.string(src),"}",
4927                               "{",self.uri(src),"}",
4928                               "{",self.string(tit or ""),"}"}
4929    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.

```
4930    function self.bulletlist(items,tight)
4931      if not self.is_writing then return "" end
4932      local buffer = {}
4933      for _,item in ipairs(items) do
4934        buffer[#buffer + 1] = self.bulletitem(item)
4935      end
4936      local contents = util.intersperse(buffer,"\n")
4937      if tight and options.tightLists then
4938        return {"\\markdownRendererUlBeginTight\n",contents,
4939          "\n\\markdownRendererUlEndTight "}
4940      else
4941        return {"\\markdownRendererUlBegin\n",contents,
4942          "\n\\markdownRendererUlEnd "}
4943      end
4944    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.

```
4945  function self.bulletitem(s)
4946    return {"\\markdownRendererUlItem ",s,
4947            "\\markdownRendererUlItemEnd "}
4948  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.

```
4949  function self.orderedlist(items,tight,startnum)
4950    if not self.is_writing then return "" end
4951    local buffer = {}
4952    local num = startnum
4953    for _,item in ipairs(items) do
4954      buffer[#buffer + 1] = self.ordereditem(item,num)
4955      if num ~= nil then
4956        num = num + 1
4957      end
4958    end
4959    local contents = util.intersperse(buffer,"\n")
4960    if tight and options.tightLists then
4961      return {"\\markdownRendererOlBeginTight\n",contents,
4962              "\n\\markdownRendererOlEndTight "}
4963    else
4964      return {"\\markdownRendererOlBegin\n",contents,
4965              "\n\\markdownRendererOlEnd "}
4966    end
4967  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.

```
4968  function self.ordereditem(s,num)
4969    if num ~= nil then
4970      return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
4971              "\\markdownRendererOlItemEnd "}
4972    else
4973      return {"\\markdownRendererOlItem ",s,
4974              "\\markdownRendererOlItemEnd "}
4975    end
4976  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.

```
4977    function self.inline_html_comment(contents)
4978      return {"\\markdownRendererInlineHtmlComment{",contents,"}"}
4979    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.

```
4980    function self.block_html_comment(contents)
4981      if not self.is_writing then return "" end
4982      return {"\\markdownRendererBlockHtmlCommentBegin\n",contents,
4983              "\n\\markdownRendererBlockHtmlCommentEnd "}
4984    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.

```
4985    function self.inline_html_tag(contents)
4986      return {"\\markdownRendererInlineHtmlTag{",self.string(contents),"}"}
4987    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.

```
4988    function self.block_html_element(s)
4989      if not self.is_writing then return "" end
4990      local name = util.cache(options.cacheDir, s, nil, nil, ".verbatim")
4991      return {"\\markdownRendererInputBlockHtmlElement{",name,"}"}
4992    end
```

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

```
4993    function self.emphasis(s)
4994      return {"\\markdownRendererEmphasis{",s,"}"}
4995    end
```

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

```
4996    function self.tickbox(f)
4997      if f == 1.0 then
4998        return "☒ "
4999      elseif f == 0.0 then
5000        return "☐ "
5001      else
5002        return "☒ "
5003      end
5004    end
```

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

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

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

```
  function self.blockquote(s)
    if #util.rope_to_string(s) == 0 then return "" end
    return {"\\markdownRendererBlockQuoteBegin\n",s,
      "\n\\markdownRendererBlockQuoteEnd "}
  end
```

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

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

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

```
  function self.document(d)
    local active_attributes = self.active_attributes
    local buf = {"\\markdownRendererDocumentBegin\n", d}

    -- pop attributes for sections that have ended
    if options.headerAttributes and self.is_writing then
      while #active_attributes > 0 do
        local attributes = active_attributes[#active_attributes]
        if #attributes > 0 then
          table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
        end
        table.remove(active_attributes, #active_attributes)
      end
    end

    table.insert(buf, "\\markdownRendererDocumentEnd")

    return buf
  end
```

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

```
  function self.attributes(attr)
    local buf = {}

```

```
    table.sort(attr)
    local key, value
    for i = 1, #attr do
      if attr[i]:sub(1, 1) == "#" then
        table.insert(buf, {"\\markdownRendererAttributeIdentifier{",
                           attr[i]:sub(2), "}"})
      elseif attr[i]:sub(1, 1) == "." then
        table.insert(buf, {"\\markdownRendererAttributeClassName{",
                           attr[i]:sub(2), "}"})
      else
        key, value = attr[i]:match("([^= ]+)%s*=%s*(.*)")
        table.insert(buf, {"\\markdownRendererAttributeKeyValue{",
                           key, "}{", value, "}"})
      end
    end

    return buf
  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.

```
  self.active_attributes = {}
```

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

```
  function self.heading(s, level, attributes)
    attributes = attributes or {}
    for i = 1, #attributes do
      attributes[attributes[i]] = true
    end

    local active_attributes = self.active_attributes
    local slice_begin_type = self.slice_begin:sub(1, 1)
    local slice_begin_identifier = self.slice_begin:sub(2) or ""
    local slice_end_type = self.slice_end:sub(1, 1)
    local slice_end_identifier = self.slice_end:sub(2) or ""

    local buf = {}

    -- push empty attributes for implied sections
    while #active_attributes < level-1 do
      table.insert(active_attributes, {})
    end

    -- pop attributes for sections that have ended
    while #active_attributes >= level do
      local active_identifiers = active_attributes[#active_attributes]
      -- tear down all active attributes at slice end
```

```
5083        if active_identifiers["#" .. slice_end_identifier] ~= nil
5084            and slice_end_type == "$" then
5085          for header_level = #active_attributes, 1, -1 do
5086            if options.headerAttributes and #active_attributes[header_level] > 0 then
5087              table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
5088            end
5089          end
5090          self.is_writing = false
5091        end
5092        table.remove(active_attributes, #active_attributes)
5093        if self.is_writing and options.headerAttributes and #active_identifiers > 0 then
5094          table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
5095        end
5096        -- apply all active attributes at slice beginning
5097        if active_identifiers["#" .. slice_begin_identifier] ~= nil
5098            and slice_begin_type == "$" then
5099          for header_level = 1, #active_attributes do
5100            if options.headerAttributes and #active_attributes[header_level] > 0 then
5101              table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
5102            end
5103          end
5104          self.is_writing = true
5105        end
5106      end
5107
5108      -- tear down all active attributes at slice end
5109      if attributes["#" .. slice_end_identifier] ~= nil
5110          and slice_end_type == "^" then
5111        for header_level = #active_attributes, 1, -1 do
5112          if options.headerAttributes and #active_attributes[header_level] > 0 then
5113            table.insert(buf, "\\markdownRendererHeaderAttributeContextEnd")
5114          end
5115        end
5116        self.is_writing = false
5117      end
5118
5119      -- push attributes for the new section
5120      table.insert(active_attributes, attributes)
5121      if self.is_writing and options.headerAttributes and #attributes > 0 then
5122        table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
5123      end
5124
5125      -- apply all active attributes at slice beginning
5126      if attributes["#" .. slice_begin_identifier] ~= nil
5127          and slice_begin_type == "^" then
5128        for header_level = 1, #active_attributes do
5129          if options.headerAttributes and #active_attributes[header_level] > 0 then
```

```
          table.insert(buf, "\\markdownRendererHeaderAttributeContextBegin")
        end
      end
      self.is_writing = true
    end

    if self.is_writing then
      table.insert(buf, self.attributes(attributes))
    end

    local cmd
    level = level + options.shiftHeadings
    if level <= 1 then
      cmd = "\\markdownRendererHeadingOne"
    elseif level == 2 then
      cmd = "\\markdownRendererHeadingTwo"
    elseif level == 3 then
      cmd = "\\markdownRendererHeadingThree"
    elseif level == 4 then
      cmd = "\\markdownRendererHeadingFour"
    elseif level == 5 then
      cmd = "\\markdownRendererHeadingFive"
    elseif level >= 6 then
      cmd = "\\markdownRendererHeadingSix"
    else
      cmd = ""
    end
    if self.is_writing then
      table.insert(buf, {cmd, "{", s, "}"})
    end

    return buf
  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.

```
  function self.get_state()
    return {
      is_writing=self.is_writing,
      active_attributes={table.unpack(self.active_attributes)},
    }
  end
```

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

```
  function self.set_state(s)
    local previous_state = self.get_state()
    for key, value in pairs(s) do
```

```
5172        self[key] = value
5173      end
5174      return previous_state
5175    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`.

```
5176    function self.defer_call(f)
5177      local previous_state = self.get_state()
5178      return function(...)
5179        local state = self.set_state(previous_state)
5180        local return_value = f(...)
5181        self.set_state(state)
5182        return return_value
5183      end
5184    end
5185
5186    return self
5187  end
```

3.1.4 Parsers

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

```
5188  local parsers                 = {}
```

3.1.4.1 Basic Parsers

```
5189  parsers.percent               = P("%")
5190  parsers.at                    = P("@")
5191  parsers.comma                 = P(",")
5192  parsers.asterisk              = P("*")
5193  parsers.dash                  = P("-")
5194  parsers.plus                  = P("+")
5195  parsers.underscore            = P("_")
5196  parsers.period                = P(".")
5197  parsers.hash                  = P("#")
5198  parsers.ampersand             = P("&")
5199  parsers.backtick              = P("`")
5200  parsers.less                  = P("<")
5201  parsers.more                  = P(">")
5202  parsers.space                 = P(" ")
5203  parsers.squote                = P("'")
5204  parsers.dquote                = P('"')
5205  parsers.lparent               = P("(")
5206  parsers.rparent               = P(")")
```

```
parsers.lbracket               = P("[")
parsers.rbracket               = P("]")
parsers.lbrace                 = P("{")
parsers.rbrace                 = P("}")
parsers.circumflex             = P("^")
parsers.slash                  = P("/")
parsers.equal                  = P("=")
parsers.colon                  = P(":")
parsers.semicolon              = P(";")
parsers.exclamation            = P("!")
parsers.pipe                   = P("|")
parsers.tilde                  = P("~")
parsers.backslash              = P("\\")
parsers.tab                    = P("\t")
parsers.newline                = P("\n")
parsers.tightblocksep          = P("\001")

parsers.digit                  = R("09")
parsers.hexdigit               = R("09","af","AF")
parsers.letter                 = R("AZ","az")
parsers.alphanumeric           = R("AZ","az","09")
parsers.keyword                = parsers.letter
                               * parsers.alphanumeric^0
parsers.internal_punctuation   = S(":;,.?")

parsers.doubleasterisks        = P("**")
parsers.doubleunderscores      = P("__")
parsers.doubletildes           = P("~~")
parsers.fourspaces             = P("    ")

parsers.any                    = P(1)
parsers.succeed                = P(true)
parsers.fail                   = P(false)

parsers.escapable              = S("!\"#$%&'()*+,-./:;<=>?@[\\]^_`{|}~")
parsers.anyescaped             = parsers.backslash / "" * parsers.escapable
                               + parsers.any

parsers.spacechar              = S("\t ")
parsers.spacing                = S(" \n\r\t")
parsers.nonspacechar           = parsers.any - parsers.spacing
parsers.optionalspace          = parsers.spacechar^0

parsers.normalchar             = parsers.any - (V("SpecialChar")
                                               + parsers.spacing
                                               + parsers.tightblocksep)
parsers.eof                    = -parsers.any
```

```
parsers.nonindentspace          = parsers.space^-3 * - parsers.spacechar
parsers.indent                  = parsers.space^-3 * parsers.tab
                                + parsers.fourspaces / ""
parsers.linechar                = P(1 - parsers.newline)

parsers.blankline               = parsers.optionalspace
                                * parsers.newline / "\n"
parsers.blanklines              = parsers.blankline^0
parsers.skipblanklines          = (parsers.optionalspace * parsers.newline)^0
parsers.indentedline            = parsers.indent    /""
                                * C(parsers.linechar^1 * parsers.newline^-
1)
parsers.optionallyindentedline = parsers.indent^-1 /""
                                * C(parsers.linechar^1 * parsers.newline^-
1)
parsers.sp                      = parsers.spacing^0
parsers.spnl                    = parsers.optionalspace
                                * (parsers.newline * parsers.optionalspace)^-
1
parsers.line                    = parsers.linechar^0 * parsers.newline
parsers.nonemptyline            = parsers.line - parsers.blankline
```

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

```
parsers.commented_line_letter  = parsers.linechar
                               + parsers.newline
                               - parsers.backslash
                               - parsers.percent
parsers.commented_line         = Cg(Cc(""), "backslashes")
                               * ((#(parsers.commented_line_letter
                                     - parsers.newline)
                                  * Cb("backslashes")
                                  * Cs(parsers.commented_line_letter
                                    - parsers.newline)^1  -- initial
                                  * Cg(Cc(""), "backslashes"))
                                 + #(parsers.backslash * parsers.backslash)
                                 * Cg((parsers.backslash  -- even backslash
                                       * parsers.backslash)^1, "backslashes")
                                 + (parsers.backslash
                                   * (#parsers.percent
                                     * Cb("backslashes")
                                     / function(backslashes)
                                       return string.rep("\\", #backslashes / 2)
                                     end
                                     * C(parsers.percent)
                                     + #parsers.commented_line_letter
                                     * Cb("backslashes")
```

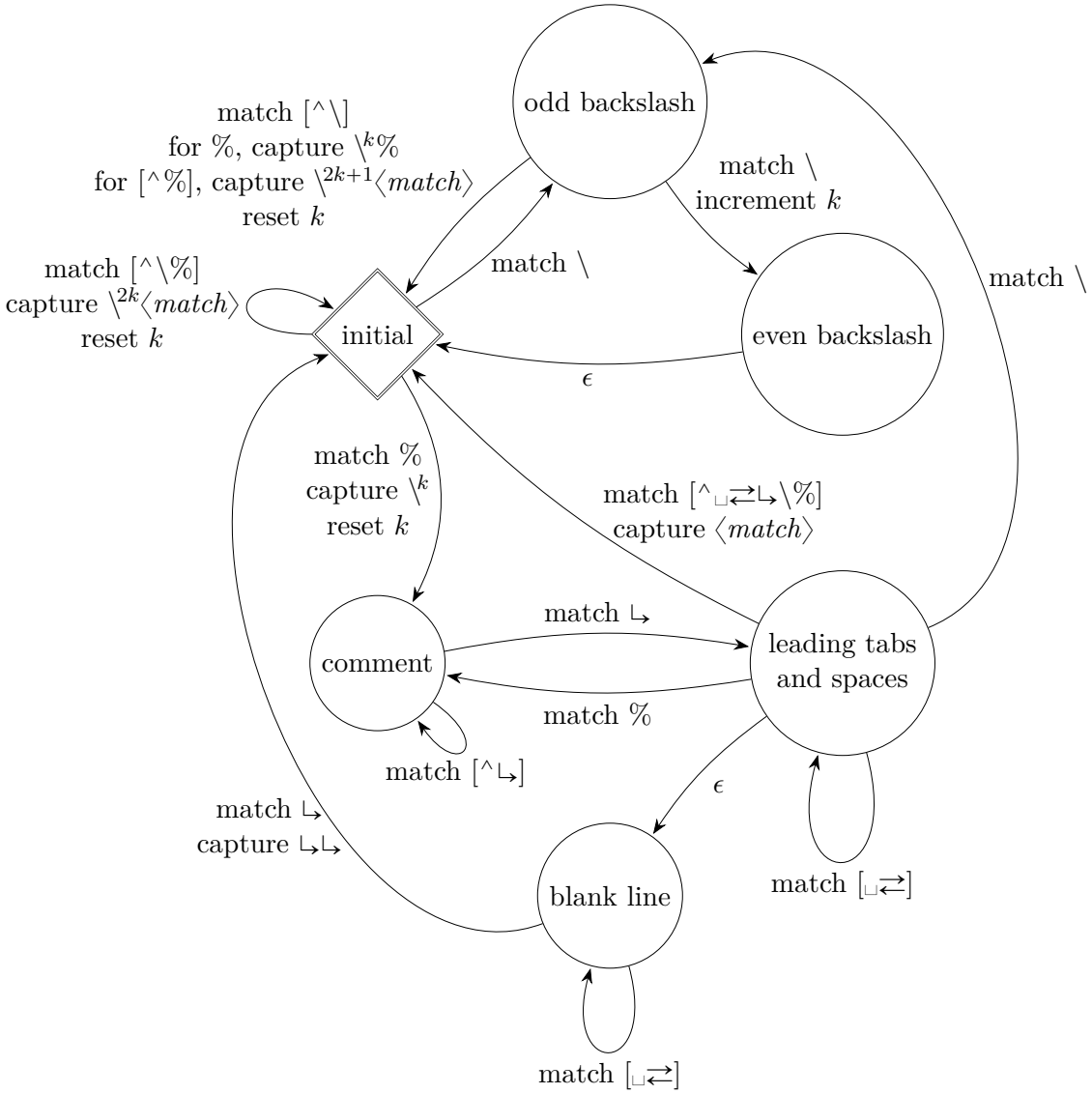

Figure 6: A pushdown automaton that recognizes TEX comments

```
                                      * Cc("\\")
                                      * C(parsers.commented_line_letter))
                                    * Cg(Cc(""), "backslashes")))^0
                                * (#parsers.percent
                                  * Cb("backslashes")
                                  / function(backslashes)
                                    return string.rep("\\", #backslashes / 2)
                                  end
                                  * ((parsers.percent  -- comment
                                     * parsers.line
                                     * #parsers.blankline) -- blank line
                                    / "\n"
                                    + parsers.percent  -- comment
                                    * parsers.line
                                    * parsers.optionalspace)  -- leading tabs and space
                                  + #(parsers.newline)
                                  * Cb("backslashes")
                                  * C(parsers.newline))

parsers.chunk                   = parsers.line * (parsers.optionallyindentedline
                                                  - parsers.blankline)^0

parsers.attribute_key_char      = parsers.alphanumeric + S("_-")
parsers.attribute_key           = (parsers.attribute_key_char
                                  - parsers.dash - parsers.digit)
                                * parsers.attribute_key_char^0
parsers.attribute_value         = ( (parsers.dquote / "")
                                  * (parsers.anyescaped - parsers.dquote)^0
                                  * (parsers.dquote / ""))
                                + ( parsers.anyescaped - parsers.dquote - parsers.rbrac
                                  - parsers.space)^0

parsers.attribute = (parsers.dash * Cc(".unnumbered"))
                  + C((parsers.hash + parsers.period)
                     * parsers.attribute_key)
                  + Cs( parsers.attribute_key
                      * parsers.optionalspace * parsers.equal * parsers.optionalspace
                      * parsers.attribute_value)
parsers.attributes = parsers.lbrace
                   * parsers.optionalspace
                   * parsers.attribute
                   * (parsers.spacechar^1
                     * parsers.attribute)^0
                   * parsers.optionalspace
                   * parsers.rbrace
```

```
parsers.raw_attribute = parsers.lbrace
                      * parsers.optionalspace
                      * parsers.equal
                      * C(parsers.attribute_key)
                      * parsers.optionalspace
                      * parsers.rbrace

-- block followed by 0 or more optionally
-- indented blocks with first line indented.
parsers.indented_blocks = function(bl)
  return Cs( bl
         * (parsers.blankline^1 * parsers.indent * -parsers.blankline * bl)^0
         * (parsers.blankline^1 + parsers.eof) )
end
```

3.1.4.2 Parsers Used for Markdown Lists

```
parsers.bulletchar = C(parsers.plus + parsers.asterisk + parsers.dash)

parsers.bullet = ( parsers.bulletchar * #parsers.spacing
                                      * (parsers.tab + parsers.space^-
3)
                 + parsers.space * parsers.bulletchar * #parsers.spacing
                                 * (parsers.tab + parsers.space^-2)
                 + parsers.space * parsers.space * parsers.bulletchar
                                 * #parsers.spacing
                                 * (parsers.tab + parsers.space^-1)
                 + parsers.space * parsers.space * parsers.space
                                 * parsers.bulletchar * #parsers.spacing
                 )

local function tickbox(interior)
  return parsers.optionalspace * parsers.lbracket
       * interior * parsers.rbracket * parsers.spacechar^1
end

parsers.ticked_box = tickbox(S("xX")) * Cc(1.0)
parsers.halfticked_box = tickbox(S("./")) * Cc(0.5)
parsers.unticked_box = tickbox(parsers.spacechar^1) * Cc(0.0)

```

3.1.4.3 Parsers Used for Markdown Code Spans

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

local function captures_equal_length(_,i,a,b)
  return #a == #b and i
end
```

```
5383
5384 parsers.closeticks  = parsers.space^-1
5385                   * Cmt(C(parsers.backtick^1)
5386                       * Cb("ticks"), captures_equal_length)
5387
5388 parsers.intickschar = (parsers.any - S(" \n\r`"))
5389                   + (parsers.newline * -parsers.blankline)
5390                   + (parsers.space - parsers.closeticks)
5391                   + (parsers.backtick^1 - parsers.closeticks)
5392
5393 parsers.inticks     = parsers.openticks * parsers.space^-1
5394                   * C(parsers.intickschar^0) * parsers.closeticks
```

3.1.4.4 Parsers Used for Markdown Tags and Links

```
5395 parsers.leader      = parsers.space^-3
5396
5397 -- content in balanced brackets, parentheses, or quotes:
5398 parsers.bracketed   = P{ parsers.lbracket
5399                        * (( parsers.backslash / "" * parsers.rbracket
5400                           + parsers.any - (parsers.lbracket
5401                                         + parsers.rbracket
5402                                         + parsers.blankline^2)
5403                           ) + V(1))^0
5404                        * parsers.rbracket }
5405
5406 parsers.inparens    = P{ parsers.lparent
5407                        * ((parsers.anyescaped - (parsers.lparent
5408                                                + parsers.rparent
5409                                                + parsers.blankline^2)
5410                           ) + V(1))^0
5411                        * parsers.rparent }
5412
5413 parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
5414                        * ((parsers.anyescaped - (parsers.squote
5415                                                + parsers.blankline^2)
5416                           ) + V(1))^0
5417                        * parsers.squote }
5418
5419 parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
5420                        * ((parsers.anyescaped - (parsers.dquote
5421                                                + parsers.blankline^2)
5422                           ) + V(1))^0
5423                        * parsers.dquote }
5424
5425 -- bracketed tag for markdown links, allowing nested brackets:
5426 parsers.tag         = parsers.lbracket
```

```
                    * Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + ( parsers.backslash / "" * parsers.rbracket
                           + parsers.any
                           - (parsers.rbracket + parsers.blankline^2)))^0)
                    * parsers.rbracket

-- url for markdown links, allowing nested brackets:
parsers.url         = parsers.less * Cs((parsers.anyescaped
                                       - parsers.more)^0)
                                   * parsers.more
                    + Cs((parsers.inparens + (parsers.anyescaped
                                              - parsers.spacing
                                              - parsers.rparent))^1)

-- quoted text, possibly with nested quotes:
parsers.title_s     = parsers.squote * Cs(((parsers.anyescaped-parsers.squote)
                                           + parsers.squoted)^0)
                                     * parsers.squote

parsers.title_d     = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
                                           + parsers.dquoted)^0)
                                     * parsers.dquote

parsers.title_p     = parsers.lparent
                    * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
                    * parsers.rparent

parsers.title       = parsers.title_d + parsers.title_s + parsers.title_p

parsers.optionaltitle
                    = parsers.spnl * parsers.title * parsers.spacechar^0
                    + Cc("")
```

3.1.4.5 Parsers Used for HTML

```
-- case-insensitive match (we assume s is lowercase). must be single byte encoding
parsers.keyword_exact = function(s)
  local parser = P(0)
  for i=1,#s do
    local c = s:sub(i,i)
    local m = c .. upper(c)
    parser = parser * S(m)
  end
  return parser
end
```

```
5471
5472 parsers.block_keyword =
5473     parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5474     parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5475     parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5476     parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5477     parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5478     parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5479     parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5480     parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5481     parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
5482     parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
5483     parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
5484     parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
5485     parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
5486     parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
5487     parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
5488     parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
5489     parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
5490     parsers.keyword_exact("td") + parsers.keyword_exact("tr")
5491
5492 -- There is no reason to support bad html, so we expect quoted attributes
5493 parsers.htmlattributevalue
5494                           = parsers.squote * (parsers.any - (parsers.blankline
5495                                                            + parsers.squote))^0
5496                                           * parsers.squote
5497                           + parsers.dquote * (parsers.any - (parsers.blankline
5498                                                            + parsers.dquote))^0
5499                                           * parsers.dquote
5500
5501 parsers.htmlattribute     = parsers.spacing^1
5502                           * (parsers.alphanumeric + S("_-"))^1
5503                           * parsers.sp * parsers.equal * parsers.sp
5504                           * parsers.htmlattributevalue
5505
5506 parsers.htmlcomment       = P("<!--")
5507                           * parsers.optionalspace
5508                           * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
5509                           * parsers.optionalspace
5510                           * P("-->")
5511
5512 parsers.htmlinstruction   = P("<?") * (parsers.any - P("?>"))^0 * P("?>")
5513
5514 parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
5515                     * parsers.sp * parsers.more
5516
5517 parsers.openelt_exact = function(s)
```

```
5518   return parsers.less * parsers.sp * parsers.keyword_exact(s)
5519         * parsers.htmlattribute^0 * parsers.sp * parsers.more
5520 end
5521
5522 parsers.openelt_block = parsers.sp * parsers.block_keyword
5523                       * parsers.htmlattribute^0 * parsers.sp * parsers.more
5524
5525 parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
5526                      * parsers.keyword * parsers.sp * parsers.more
5527
5528 parsers.closeelt_exact = function(s)
5529   return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
5530         * parsers.sp * parsers.more
5531 end
5532
5533 parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
5534                      * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5535                      * parsers.more
5536
5537 parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
5538                        * parsers.htmlattribute^0 * parsers.sp * parsers.slash
5539                        * parsers.more
5540
5541 parsers.displaytext = (parsers.any - parsers.less)^1
5542
5543 -- return content between two matched HTML tags
5544 parsers.in_matched = function(s)
5545   return { parsers.openelt_exact(s)
5546          * (V(1) + parsers.displaytext
5547            + (parsers.less - parsers.closeelt_exact(s)))^0
5548          * parsers.closeelt_exact(s) }
5549 end
5550
5551 local function parse_matched_tags(s,pos)
5552   local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
5553   return lpeg.match(parsers.in_matched(t),s,pos-1)
5554 end
5555
5556 parsers.in_matched_block_tags = parsers.less
5557                               * Cmt(#parsers.openelt_block, parse_matched_tags)
5558
```

3.1.4.6 Parsers Used for HTML Entities

```
5559 parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
5560                   * C(parsers.hexdigit^1) * parsers.semicolon
5561 parsers.decentity = parsers.ampersand * parsers.hash
```

```
                  * C(parsers.digit^1) * parsers.semicolon
parsers.tagentity = parsers.ampersand * C(parsers.alphanumeric^1)
                  * parsers.semicolon
```

3.1.4.7 Helpers for References

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

3.1.4.8 Inline Elements

```
parsers.Inline         = V("Inline")
parsers.IndentedInline = V("IndentedInline")

-- parse many p between starter and ender
parsers.between = function(p, starter, ender)
  local ender2 = B(parsers.nonspacechar) * ender
  return (starter * #parsers.nonspacechar * Ct(p * (p - ender2)^0) * ender2)
end

parsers.urlchar      = parsers.anyescaped - parsers.newline - parsers.more
```

3.1.4.9 Block Elements

```
parsers.lineof = function(c)
    return (parsers.leader * (P(c) * parsers.optionalspace)^3
            * (parsers.newline * parsers.blankline^1
              + parsers.newline^-1 * parsers.eof))
end
```

3.1.4.10 Headings

```
-- parse Atx heading start and return level
parsers.heading_start = #parsers.hash * C(parsers.hash^-6)
                      * -parsers.hash / length

-- parse setext header ending and return level
parsers.heading_level = parsers.equal^1 * Cc(1) + parsers.dash^1 * Cc(2)

local function strip_atx_end(s)
  return s:gsub("[#%s]*\n$","")
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*⟩.

```
5594 M.reader = {}
5595 function M.reader.new(writer, options)
5596   local self = {}
```

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

```
5597   self.writer = writer
5598   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.

```
5599   self.parsers = {}
5600   (function(parsers)
5601     setmetatable(self.parsers, {
5602       __index = function (_, key)
5603         return parsers[key]
5604       end
5605     })
5606   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.

```
5607   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.

```
5608   function self.normalize_tag(tag)
5609     return string.lower(
5610       gsub(util.rope_to_string(tag), "[ \n\r\t]+", " "))
5611   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.

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

```
if options.preserveTabs then
  self.expandtabs = function(s) return s end
else
  self.expandtabs = function(s)
                      if s:find("\t") then
                        return iterlines(s, util.expand_tabs_in_line)
                      else
                        return s
                      end
                    end
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.

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

```
    if toplevel and options.stripIndent then
        local min_prefix_length, min_prefix = nil, ''
        str = iterlines(str, function(line)
            if lpeg.match(parsers.nonemptyline, line) == nil then
                return line
            end
            line = util.expand_tabs_in_line(line)
            local prefix = lpeg.match(C(parsers.optionalspace), line)
            local prefix_length = #prefix
            local is_shorter = min_prefix_length == nil
```

```
5640              is_shorter = is_shorter or prefix_length < min_prefix_length
5641              if is_shorter then
5642                  min_prefix_length, min_prefix = prefix_length, prefix
5643              end
5644              return line
5645          end)
5646          str = str:gsub('^' .. min_prefix, '')
5647      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.

```
5648      if toplevel and (options.texComments or options.hybrid) then
5649        str = lpeg.match(Ct(parsers.commented_line^1), str)
5650        str = util.rope_to_string(str)
5651      end
5652      local res = lpeg.match(grammar(), str)
5653      if res == nil then
5654        error(format("%s failed on:\n%s", name, str:sub(1,20)))
5655      else
5656        return res
5657      end
5658    end
5659  end
5660
5661  self.create_parser("parse_blocks",
5662                     function()
5663                       return parsers.blocks
5664                     end, true)
5665
5666  self.create_parser("parse_blocks_nested",
5667                     function()
5668                       return parsers.blocks_nested
5669                     end, false)
5670
5671  self.create_parser("parse_inlines",
5672                     function()
5673                       return parsers.inlines
5674                     end, false)
5675
5676  self.create_parser("parse_inlines_no_link",
5677                     function()
5678                       return parsers.inlines_no_link
5679                     end, false)
5680
5681  self.create_parser("parse_inlines_no_inline_note",
5682                     function()
```

```
                      return parsers.inlines_no_inline_note
                    end, false)

  self.create_parser("parse_inlines_no_html",
                    function()
                      return parsers.inlines_no_html
                    end, false)

  self.create_parser("parse_inlines_nbsp",
                    function()
                      return parsers.inlines_nbsp
                    end, false)
```

3.1.5.3 Parsers Used for Markdown Lists (local)

```
  if options.hashEnumerators then
    parsers.dig = parsers.digit + parsers.hash
  else
    parsers.dig = parsers.digit
  end

  parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
                     + C(parsers.dig^2 * parsers.period) * #parsers.spacing
                                         * (parsers.tab + parsers.space^1)
                     + C(parsers.dig * parsers.period) * #parsers.spacing
                                       * (parsers.tab + parsers.space^-2)
                     + parsers.space * C(parsers.dig^2 * parsers.period)
                                       * #parsers.spacing
                     + parsers.space * C(parsers.dig * parsers.period)
                                       * #parsers.spacing
                                       * (parsers.tab + parsers.space^-1)
                     + parsers.space * parsers.space * C(parsers.dig^1
                                       * parsers.period) * #parsers.spacing
```

3.1.5.4 Parsers Used for Blockquotes (local)

```
  -- strip off leading > and indents, and run through blocks
  parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
1)/""
                             * parsers.linechar^0 * parsers.newline)^1
                           * (-V("BlockquoteExceptions") * parsers.linechar^1
                             * parsers.newline)^0

  if not options.breakableBlockquotes then
    parsers.blockquote_body = parsers.blockquote_body
                           * (parsers.blankline^0 / "")
  end
```

3.1.5.5 Helpers for Links and References (local)

```
5723  -- List of references defined in the document
5724  local references
5725
5726  -- add a reference to the list
5727  local function register_link(tag,url,title)
5728      references[self.normalize_tag(tag)] = { url = url, title = title }
5729      return ""
5730  end
5731
5732  -- lookup link reference and return either
5733  -- the link or nil and fallback text.
5734  local function lookup_reference(label,sps,tag)
5735      local tagpart
5736      if not tag then
5737          tag = label
5738          tagpart = ""
5739      elseif tag == "" then
5740          tag = label
5741          tagpart = "[]"
5742      else
5743          tagpart = {"[",
5744            self.parser_functions.parse_inlines(tag),
5745            "]"}
5746      end
5747      if sps then
5748        tagpart = {sps, tagpart}
5749      end
5750      local r = references[self.normalize_tag(tag)]
5751      if r then
5752        return r
5753      else
5754        return nil, {"[",
5755          self.parser_functions.parse_inlines(label),
5756          "]", tagpart}
5757      end
5758  end
5759
5760  -- lookup link reference and return a link, if the reference is found,
5761  -- or a bracketed label otherwise.
5762  local function indirect_link(label,sps,tag)
5763    return writer.defer_call(function()
5764      local r,fallback = lookup_reference(label,sps,tag)
5765      if r then
5766        return writer.link(
5767          self.parser_functions.parse_inlines_no_link(label),
5768          r.url, r.title)
```

```
      else
        return fallback
      end
    end)
  end

  -- lookup image reference and return an image, if the reference is found,
  -- or a bracketed label otherwise.
  local function indirect_image(label,sps,tag)
    return writer.defer_call(function()
      local r,fallback = lookup_reference(label,sps,tag)
      if r then
        return writer.image(writer.string(label), r.url, r.title)
      else
        return {"!", fallback}
      end
    end)
  end
```

3.1.5.6 Inline Elements (local)

```
  parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
                   / writer.string

  parsers.Symbol   = (V("SpecialChar") - parsers.tightblocksep)
                   / writer.string

  parsers.Ellipsis = P("...") / writer.ellipsis

  parsers.Smart    = parsers.Ellipsis

  parsers.Code     = parsers.inticks / writer.code

  if options.blankBeforeBlockquote then
    parsers.bqstart = parsers.fail
  else
    parsers.bqstart = parsers.more
  end

  if options.blankBeforeHeading then
    parsers.headerstart = parsers.fail
  else
    parsers.headerstart = parsers.hash
                        + (parsers.line * (parsers.equal^1 + parsers.dash^1)
                        * parsers.optionalspace * parsers.newline)
  end

```

```
parsers.EndlineExceptions
                    = parsers.blankline -- paragraph break
                    + parsers.tightblocksep  -- nested list
                    + parsers.eof       -- end of document
                    + parsers.bqstart
                    + parsers.headerstart

parsers.Endline   = parsers.newline
                  * -V("EndlineExceptions")
                  * parsers.spacechar^0
                  / (options.hardLineBreaks and writer.linebreak
                                            or writer.space)

parsers.OptionalIndent
                    = parsers.spacechar^1 / writer.space

parsers.Space      = parsers.spacechar^2 * parsers.Endline / writer.linebreak
                   + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
                   + parsers.spacechar^1 * parsers.Endline
                                         * parsers.optionalspace
                                         / (options.hardLineBreaks
                                            and writer.linebreak
                                             or writer.space)
                   + parsers.spacechar^1 * parsers.optionalspace
                                         / writer.space

parsers.NonbreakingEndline
                  = parsers.newline
                  * -V("EndlineExceptions")
                  * parsers.spacechar^0
                  / (options.hardLineBreaks and writer.linebreak
                                            or writer.nbsp)

parsers.NonbreakingSpace
                = parsers.spacechar^2 * parsers.Endline / writer.linebreak
                + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
                + parsers.spacechar^1 * parsers.Endline
                                      * parsers.optionalspace
                                      / (options.hardLineBreaks
                                         and writer.linebreak
                                          or writer.nbsp)
                + parsers.spacechar^1 * parsers.optionalspace
                                      / writer.nbsp

if options.underscores then
  parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
                                     parsers.doubleasterisks)
```

```
                    + parsers.between(parsers.Inline, parsers.doubleunderscores,
                                      parsers.doubleunderscores)
                    ) / writer.strong

  parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
                                      parsers.asterisk)
                    + parsers.between(parsers.Inline, parsers.underscore,
                                      parsers.underscore)
                    ) / writer.emphasis
else
  parsers.Strong = ( parsers.between(parsers.Inline, parsers.doubleasterisks,
                                      parsers.doubleasterisks)
                    ) / writer.strong

  parsers.Emph   = ( parsers.between(parsers.Inline, parsers.asterisk,
                                      parsers.asterisk)
                    ) / writer.emphasis
end

parsers.AutoLinkUrl    = parsers.less
                         * C(parsers.alphanumeric^1 * P("://") * parsers.urlchar^1)
                         * parsers.more
                         / function(url)
                             return writer.link(writer.escape(url), url)
                           end

parsers.AutoLinkEmail = parsers.less
                        * C((parsers.alphanumeric + S("-._+"))^1
                        * P("@") * parsers.urlchar^1)
                        * parsers.more
                        / function(email)
                            return writer.link(writer.escape(email),
                                               "mailto:"..email)
                          end

parsers.AutoLinkRelativeReference
                         = parsers.less
                         * C(parsers.urlchar^1)
                         * parsers.more
                         / function(url)
                             return writer.link(writer.escape(url), url)
                           end

parsers.DirectLink    = (parsers.tag / self.parser_functions.parse_inlines_no_link)
                        * parsers.spnl
                        * parsers.lparent
                        * (parsers.url + Cc(""))  -- link can be empty [foo]()
```

```
5907                        * parsers.optionaltitle
5908                        * parsers.rparent
5909                        / writer.link
5910
5911  parsers.IndirectLink  = parsers.tag * (C(parsers.spnl) * parsers.tag)^-
    1
5912                        / indirect_link
5913
5914  -- parse a link or image (direct or indirect)
5915  parsers.Link          = parsers.DirectLink + parsers.IndirectLink
5916
5917  parsers.DirectImage   = parsers.exclamation
5918                        * (parsers.tag / self.parser_functions.parse_inlines)
5919                        * parsers.spnl
5920                        * parsers.lparent
5921                        * (parsers.url + Cc("")  -- link can be empty [foo]()
5922                        * parsers.optionaltitle
5923                        * parsers.rparent
5924                        / writer.image
5925
5926  parsers.IndirectImage = parsers.exclamation * parsers.tag
5927                        * (C(parsers.spnl) * parsers.tag)^-1 / indirect_image
5928
5929  parsers.Image         = parsers.DirectImage + parsers.IndirectImage
5930
5931  -- avoid parsing long strings of * or _ as emph/strong
5932  parsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
5933                        / writer.string
5934
5935  parsers.EscapedChar   = parsers.backslash * C(parsers.escapable) / writer.string
5936
5937  parsers.InlineHtml    = parsers.emptyelt_any / writer.inline_html_tag
5938                        + (parsers.htmlcomment / self.parser_functions.parse_inlines_n
5939                        / writer.inline_html_comment
5940                        + parsers.htmlinstruction
5941                        + parsers.openelt_any / writer.inline_html_tag
5942                        + parsers.closeelt_any / writer.inline_html_tag
5943
5944  parsers.HtmlEntity    = parsers.hexentity / entities.hex_entity  / writer.string
5945                        + parsers.decentity / entities.dec_entity  / writer.string
5946                        + parsers.tagentity / entities.char_entity / writer.string
```

3.1.5.7 Block Elements (local)

```
5947  parsers.DisplayHtml  = (parsers.htmlcomment / self.parser_functions.parse_blocks_nes
5948                       / writer.block_html_comment
5949                       + parsers.emptyelt_block / writer.block_html_element
```

```
                        + parsers.openelt_exact("hr") / writer.block_html_element
                        + parsers.in_matched_block_tags / writer.block_html_element
                        + parsers.htmlinstruction

  parsers.Verbatim     = Cs( (parsers.blanklines
                           * ((parsers.indentedline - parsers.blankline))^1)^1
                           ) / self.expandtabs / writer.verbatim

  parsers.BlockquoteExceptions = parsers.leader * parsers.more
                               + parsers.blankline

  parsers.Blockquote   = Cs(parsers.blockquote_body^1)
                       / self.parser_functions.parse_blocks_nested
                       / writer.blockquote

  parsers.ThematicBreak = ( parsers.lineof(parsers.asterisk)
                          + parsers.lineof(parsers.dash)
                          + parsers.lineof(parsers.underscore)
                          ) / writer.thematic_break

  parsers.Reference    = parsers.define_reference_parser / register_link

  parsers.Paragraph    = parsers.nonindentspace * Ct(parsers.Inline^1)
                       * ( parsers.newline
                       * ( parsers.blankline^1
                         + #parsers.hash
                         + #(parsers.leader * parsers.more * parsers.space^-
1)
                         + parsers.eof
                         )
                       + parsers.eof )
                       / writer.paragraph

  parsers.Plain        = parsers.nonindentspace * Ct(parsers.Inline^1)
                       / writer.plain
```

3.1.5.8 Lists (local)

```
  parsers.starter = parsers.bullet + parsers.enumerator

  if options.taskLists then
    parsers.tickbox = ( parsers.ticked_box
                      + parsers.halfticked_box
                      + parsers.unticked_box
                      ) / writer.tickbox
  else
     parsers.tickbox = parsers.fail
```

```
5993   end
5994
5995   -- we use \001 as a separator between a tight list item and a
5996   -- nested list under it.
5997   parsers.NestedList            = Cs((parsers.optionallyindentedline
5998                                       - parsers.starter)^1)
5999                                 / function(a) return "\001"..a end
6000
6001   parsers.ListBlockLine         = parsers.optionallyindentedline
6002                                 - parsers.blankline - (parsers.indent^-
    1
6003                                                        * parsers.starter)
6004
6005   parsers.ListBlock             = parsers.line * parsers.ListBlockLine^0
6006
6007   parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6008                                 * parsers.ListBlock
6009
6010   parsers.TightListItem = function(starter)
6011       return -parsers.ThematicBreak
6012              * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Nes
    1)
6013                / self.parser_functions.parse_blocks_nested)
6014              * -(parsers.blanklines * parsers.indent)
6015   end
6016
6017   parsers.LooseListItem = function(starter)
6018       return -parsers.ThematicBreak
6019              * Cs( starter / "" * parsers.tickbox^-1 * parsers.ListBlock * Cc("\n")
6020                * (parsers.NestedList + parsers.ListContinuationBlock^0)
6021                * (parsers.blanklines / "\n\n")
6022                ) / self.parser_functions.parse_blocks_nested
6023   end
6024
6025   parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
6026                        * parsers.skipblanklines * -parsers.bullet
6027                        + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
6028                        * parsers.skipblanklines )
6029                      / writer.bulletlist
6030
6031   local function ordered_list(items,tight,startnum)
6032     if options.startNumber then
6033       startnum = tonumber(startnum) or 1  -- fallback for '#'
6034       if startnum ~= nil then
6035         startnum = math.floor(startnum)
6036       end
6037     else
```

```
      startnum = nil
    end
    return writer.orderedlist(items,tight,startnum)
  end

  parsers.OrderedList = Cg(parsers.enumerator, "listtype") *
                        ( Ct(parsers.TightListItem(Cb("listtype"))
                            * parsers.TightListItem(parsers.enumerator)^0)
                        * Cc(true) * parsers.skipblanklines * -parsers.enumerator
                        + Ct(parsers.LooseListItem(Cb("listtype"))
                            * parsers.LooseListItem(parsers.enumerator)^0)
                        * Cc(false) * parsers.skipblanklines
                        ) * Cb("listtype") / ordered_list
```

3.1.5.9 Blank (local)

```
  parsers.Blank        = parsers.blankline / ""
                       + parsers.Reference
                       + (parsers.tightblocksep / "\n")
```

3.1.5.10 Headings (local)

```
  -- parse atx header
  parsers.AtxHeading = Cg(parsers.heading_start, "level")
                      * parsers.optionalspace
                      * (C(parsers.line)
                        / strip_atx_end
                        / self.parser_functions.parse_inlines)
                      * Cb("level")
                      / writer.heading

  parsers.SetextHeading = #(parsers.line * S("=-"))
                          * Ct(parsers.linechar^1
                              / self.parser_functions.parse_inlines)
                          * parsers.newline
                          * parsers.heading_level
                          * parsers.optionalspace
                          * parsers.newline
                          / writer.heading

  parsers.Heading = parsers.AtxHeading + parsers.SetextHeading
```

3.1.5.11 Syntax Specification Define `reader->finalize_grammar` as a function that constructs the PEG grammar of markdown, applies syntax extensions `extensions` and returns a conversion function that takes a markdown string and turns it into a plain TEX output.

```
  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.

```
local walkable_syntax = (function(global_walkable_syntax)
  local local_walkable_syntax = {}
  for lhs, rule in pairs(global_walkable_syntax) do
    local_walkable_syntax[lhs] = util.table_copy(rule)
  end
  return local_walkable_syntax
end)(walkable_syntax)
```

The `reader->insert_pattern` method adds a pattern to `walkable_syntax[`*left-hand side terminal symbol*`]` before, instead of, or after a right-hand-side terminal symbol.

```
local current_extension_name = nil
self.insert_pattern = function(selector, pattern, pattern_name)
  assert(pattern_name == nil or type(pattern_name) == "string")
  local _, _, lhs, pos, rhs = selector:find("^(%a+)%s+([%a%s]+%a+)%s+(%a+)$")
  assert(lhs ~= nil,
    [[Expected selector in form "LHS (before|after|instead of) RHS", not "]]
    .. selector .. [["]])
  assert(walkable_syntax[lhs] ~= nil,
    [[Rule ]] .. lhs .. [[ -> ... does not exist in markdown grammar]])
  assert(pos == "before" or pos == "after" or pos == "instead of",
    [[Expected positional specifier "before", "after", or "instead of", not "]]
    .. pos .. [["]])
  local rule = walkable_syntax[lhs]
  local index = nil
  for current_index, current_rhs in ipairs(rule) do
    if type(current_rhs) == "string" and current_rhs == rhs then
      index = current_index
      if pos == "after" then
        index = index + 1
      end
      break
    end
  end
  assert(index ~= nil,
    [[Rule ]] .. lhs .. [[ -> ]] .. rhs
      .. [[ does not exist in markdown grammar]])
  local accountable_pattern
  if current_extension_name then
    accountable_pattern = { pattern, current_extension_name, pattern_name }
  else
    assert(type(pattern) == "string",
```

```
6112          [[reader->insert_pattern() was called outside an extension with ]]
6113          .. [[a PEG pattern instead of a rule name]])
6114        accountable_pattern = pattern
6115      end
6116      if pos == "instead of" then
6117        rule[index] = accountable_pattern
6118      else
6119        table.insert(rule, index, accountable_pattern)
6120      end
6121    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.

```
6122    local syntax =
6123      { "Blocks",
6124
6125        Blocks                = V("InitializeState")
6126                              * ( V("ExpectedJekyllData")
6127                                * (V("Blank")^0 / writer.interblocksep))^-
    1
6128                              * V("Blank")^0
6129                              * V("Block")^-1
6130                              * ( V("Blank")^0 / writer.interblocksep
6131                                * V("Block"))^0
6132                              * V("Blank")^0 * parsers.eof,
6133
6134        ExpectedJekyllData    = parsers.fail,
6135
6136        Blank                 = parsers.Blank,
6137
6138        Blockquote            = parsers.Blockquote,
6139        Verbatim              = parsers.Verbatim,
6140        ThematicBreak         = parsers.ThematicBreak,
6141        BulletList            = parsers.BulletList,
6142        OrderedList           = parsers.OrderedList,
6143        Heading               = parsers.Heading,
6144        DisplayHtml           = parsers.DisplayHtml,
6145        Paragraph             = parsers.Paragraph,
6146        Plain                 = parsers.Plain,
6147
6148        EndlineExceptions     = parsers.EndlineExceptions,
6149        BlockquoteExceptions  = parsers.BlockquoteExceptions,
6150
6151        Str                   = parsers.Str,
6152        Space                 = parsers.Space,
6153        OptionalIndent        = parsers.OptionalIndent,
6154        Endline               = parsers.Endline,
```

```
        UlOrStarLine          = parsers.UlOrStarLine,
        Strong                = parsers.Strong,
        Emph                  = parsers.Emph,
        Link                  = parsers.Link,
        Image                 = parsers.Image,
        Code                  = parsers.Code,
        AutoLinkUrl           = parsers.AutoLinkUrl,
        AutoLinkEmail         = parsers.AutoLinkEmail,
        AutoLinkRelativeReference
                              = parsers.AutoLinkRelativeReference,
        InlineHtml            = parsers.InlineHtml,
        HtmlEntity            = parsers.HtmlEntity,
        EscapedChar           = parsers.EscapedChar,
        Smart                 = parsers.Smart,
        Symbol                = parsers.Symbol,
        SpecialChar           = parsers.fail,
        InitializeState       = parsers.succeed,
      }
```

Define `reader->update_rule` as a function that receives two arguments: a left-hand side terminal symbol and a function that accepts the current PEG pattern in `walkable_syntax[`left-hand side terminal symbol`]` if defined or `nil` otherwise and returns a PEG pattern that will (re)define `walkable_syntax[`left-hand side terminal symbol`]`.

```
    self.update_rule = function(rule_name, get_pattern)
      assert(current_extension_name ~= nil)
      assert(syntax[rule_name] ~= nil,
        [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
      local previous_pattern
      local extension_name
      if walkable_syntax[rule_name] then
        local previous_accountable_pattern = walkable_syntax[rule_name][1]
        previous_pattern = previous_accountable_pattern[1]
        extension_name = previous_accountable_pattern[2] .. ", " .. current_extension_
      else
        previous_pattern = nil
        extension_name = current_extension_name
      end
      local pattern = get_pattern(previous_pattern)
      local accountable_pattern = { pattern, extension_name, rule_name }
      walkable_syntax[rule_name] = { accountable_pattern }
    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.

```
    local special_characters = {}
```

```
    self.add_special_character = function(c)
      table.insert(special_characters, c)
      syntax.SpecialChar = S(table.concat(special_characters, ""))
    end

    self.add_special_character("*")
    self.add_special_character("[")
    self.add_special_character("]")
    self.add_special_character("<")
    self.add_special_character("!")
    self.add_special_character("\\")
```

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

```
    self.initialize_named_group = function(name, value)
      syntax.InitializeState = syntax.InitializeState
                             * Cg(Ct("") / value, name)
    end
```

Apply syntax extensions.

```
    for _, extension in ipairs(extensions) do
      current_extension_name = extension.name
      extension.extend_writer(writer)
      extension.extend_reader(self)
    end
    current_extension_name = nil
```

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

```
    if options.debugExtensions then
      local sorted_lhs = {}
      for lhs, _ in pairs(walkable_syntax) do
        table.insert(sorted_lhs, lhs)
      end
      table.sort(sorted_lhs)

      local output_lines = {"{"}
      for lhs_index, lhs in ipairs(sorted_lhs) do
        local encoded_lhs = util.encode_json_string(lhs)
        table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
        local rule = walkable_syntax[lhs]
        for rhs_index, rhs in ipairs(rule) do
          local human_readable_rhs
          if type(rhs) == "string" then
            human_readable_rhs = rhs
          else
            local pattern_name
            if rhs[3] then
```

```
6232              pattern_name = rhs[3]
6233            else
6234              pattern_name = "Anonymous Pattern"
6235            end
6236            local extension_name = rhs[2]
6237            human_readable_rhs = pattern_name .. [[ (]] .. extension_name .. [[)]]
6238          end
6239          local encoded_rhs = util.encode_json_string(human_readable_rhs)
6240          local output_line = [[        ]] .. encoded_rhs
6241          if rhs_index < #rule then
6242            output_line = output_line .. ","
6243          end
6244          table.insert(output_lines, output_line)
6245        end
6246        local output_line = "    ]"
6247        if lhs_index < #sorted_lhs then
6248          output_line = output_line .. ","
6249        end
6250        table.insert(output_lines, output_line)
6251      end
6252      table.insert(output_lines, "}")
6253
6254      local output = table.concat(output_lines, "\n")
6255      local output_filename = options.debugExtensionsFileName
6256      local output_file = assert(io.open(output_filename, "w"),
6257        [[Could not open file "]] .. output_filename .. [[" for writing]])
6258      assert(output_file:write(output))
6259      assert(output_file:close())
6260    end
```

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

```
6261    walkable_syntax["IndentedInline"] = util.table_copy(
6262      walkable_syntax["Inline"])
6263    self.insert_pattern(
6264      "IndentedInline instead of Space",
6265      "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`.

```
6266    for lhs, rule in pairs(walkable_syntax) do
6267      syntax[lhs] = parsers.fail
6268      for _, rhs in ipairs(rule) do
6269        local pattern
```

Although the interface of the `reader->insert_pattern` method does document this (see Section 2.1.2), we allow the `reader->insert_pattern` and

reader->update_rule methods to insert not just PEG patterns, but also rule names that reference the PEG grammar of Markdown.

```
6270          if type(rhs) == "string" then
6271            pattern = V(rhs)
6272          else
6273            pattern = rhs[1]
6274            if type(pattern) == "string" then
6275              pattern = V(pattern)
6276            end
6277          end
6278          syntax[lhs] = syntax[lhs] + pattern
6279        end
6280      end
```

Finalize the parser by reacting to options and by producing special parsers for difficult edge cases such as blocks nested in definition lists or inline content nested in link, note, and image labels.

```
6281      if options.underscores then
6282        self.add_special_character("_")
6283      end
6284
6285      if not options.codeSpans then
6286        syntax.Code = parsers.fail
6287      else
6288        self.add_special_character("`")
6289      end
6290
6291      if not options.html then
6292        syntax.DisplayHtml = parsers.fail
6293        syntax.InlineHtml = parsers.fail
6294        syntax.HtmlEntity  = parsers.fail
6295      else
6296        self.add_special_character("&")
6297      end
6298
6299      if options.preserveTabs then
6300        options.stripIndent = false
6301      end
6302
6303      if not options.smartEllipses then
6304        syntax.Smart = parsers.fail
6305      else
6306        self.add_special_character(".")
6307      end
6308
6309      if not options.relativeReferences then
6310        syntax.AutoLinkRelativeReference = parsers.fail
```

```
6311      end
6312
6313      local blocks_nested_t = util.table_copy(syntax)
6314      blocks_nested_t.ExpectedJekyllData = parsers.fail
6315      parsers.blocks_nested = Ct(blocks_nested_t)
6316
6317      parsers.blocks = Ct(syntax)
6318
6319      local inlines_t = util.table_copy(syntax)
6320      inlines_t[1] = "Inlines"
6321      inlines_t.Inlines = V("InitializeState")
6322                        * parsers.Inline^0
6323                        * ( parsers.spacing^0
6324                          * parsers.eof / "")
6325      parsers.inlines = Ct(inlines_t)
6326
6327      local inlines_no_link_t = util.table_copy(inlines_t)
6328      inlines_no_link_t.Link = parsers.fail
6329      parsers.inlines_no_link = Ct(inlines_no_link_t)
6330
6331      local inlines_no_inline_note_t = util.table_copy(inlines_t)
6332      inlines_no_inline_note_t.InlineNote = parsers.fail
6333      parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)
6334
6335      local inlines_no_html_t = util.table_copy(inlines_t)
6336      inlines_no_html_t.DisplayHtml = parsers.fail
6337      inlines_no_html_t.InlineHtml = parsers.fail
6338      inlines_no_html_t.HtmlEntity = parsers.fail
6339      parsers.inlines_no_html = Ct(inlines_no_html_t)
6340
6341      local inlines_nbsp_t = util.table_copy(inlines_t)
6342      inlines_nbsp_t.Endline = parsers.NonbreakingEndline
6343      inlines_nbsp_t.Space = parsers.NonbreakingSpace
6344      parsers.inlines_nbsp = Ct(inlines_nbsp_t)
```

Return a function that converts markdown string `input` into a plain TEX output and returns it..

```
6345      return function(input)
```

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.

```
6346        input = input:gsub("\r\n?", "\n")
6347        if input:sub(-1) ~= "\n" then
6348          input = input .. "\n"
6349        end
```

When determining the name of the cache file, create salt for the hashing function out of the package version and the passed options recognized by the Lua interface (see Section 2.1.3). The `cacheDir` option is disregarded.

```
      references = {}
      local opt_string = {}
      for k, _ in pairs(defaultOptions) do
        local v = options[k]
        if type(v) == "table" then
          for _, i in ipairs(v) do
            opt_string[#opt_string+1] = k .. "=" .. tostring(i)
          end
        elseif k ~= "cacheDir" then
          opt_string[#opt_string+1] = k .. "=" .. tostring(v)
        end
      end
      table.sort(opt_string)
      local salt = table.concat(opt_string, ",") .. "," .. metadata.version
      local output
```

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

```
      local function convert(input)
        local document = self.parser_functions.parse_blocks(input)
        return util.rope_to_string(writer.document(document))
      end
      if options.eagerCache or options.finalizeCache then
        local name = util.cache(options.cacheDir, input, salt, convert,
                                ".md" .. writer.suffix)
        output = writer.pack(name)
```

Otherwise, return the result of the conversion directly.

```
      else
        output = convert(input)
      end
```

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

```
      if options.finalizeCache then
        local file, mode
        if options.frozenCacheCounter > 0 then
          mode = "a"
        else
          mode = "w"
        end
        file = assert(io.open(options.frozenCacheFileName, mode),
          [[Could not open file "]] .. options.frozenCacheFileName
```

```
          .. [[" for writing]])
        assert(file:write([[\expandafter\global\expandafter\def\csname ]]
          .. [[markdownFrozenCache]] .. options.frozenCacheCounter
          .. [[\endcsname{]] .. output .. [[}]] .. "\n"))
        assert(file:close())
      end
      return output
    end
  end
  return self
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.

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

3.1.6.1 Bracketed Spans

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

```
M.extensions.bracketed_spans = function()
  return {
    name = "built-in bracketed_spans syntax extension",
    extend_writer = function(self)
```

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

```
      function self.span(s, attr)
        return {"\\markdownRendererBracketedSpanAttributeContextBegin",
                self.attributes(attr),
                s,
                "\\markdownRendererBracketedSpanAttributeContextEnd{}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Span = parsers.between(parsers.Inline,
                                   parsers.lbracket,
                                   parsers.rbracket)
                 * Ct(parsers.attributes)
                 / writer.span

```

```
6417       self.insert_pattern("Inline after Emph",
6418                           Span, "Span")
6419     end
6420   }
6421 end
```

3.1.6.2 Citations

The `extensions.citations` function implements the Pandoc citation syntax extension. When the `citation_nbsps` parameter is enabled, the syntax extension will replace regular spaces with non-breaking spaces inside the prenotes and postnotes of citations.

```
6422 M.extensions.citations = function(citation_nbsps)
```

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

```
6423   local escaped_citation_chars = {
6424     ["{"] = "\\markdownRendererLeftBrace{}",
6425     ["}"] = "\\markdownRendererRightBrace{}",
6426     ["%"] = "\\markdownRendererPercentSign{}",
6427     ["\\"] = "\\markdownRendererBackslash{}",
6428     ["#"] = "\\markdownRendererHash{}",
6429   }
6430   return {
6431     name = "built-in citations syntax extension",
6432     extend_writer = function(self)
6433       local options = self.options
6434
```

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

```
6435       local escape_citation = util.escaper(
6436         escaped_citation_chars,
6437         self.escaped_minimal_strings)
```

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

```
6438       if options.hybrid then
6439         self.citation = self.escape_minimal
6440       else
6441         self.citation = escape_citation
6442       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.

```
6443      function self.citations(text_cites, cites)
6444        local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
6445          "{", #cites, "}"}
6446        for _,cite in ipairs(cites) do
6447          buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6448            cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}"}
6449        end
6450        return buffer
6451      end
6452    end, extend_reader = function(self)
6453      local parsers = self.parsers
6454      local writer = self.writer
6455
6456      local citation_chars
6457                    = parsers.alphanumeric
6458                    + S("#$%&-+<>~/_")
6459
6460      local citation_name
6461                    = Cs(parsers.dash^-1) * parsers.at
6462                    * Cs(citation_chars
6463                        * (((citation_chars + parsers.internal_punctuation
6464                            - parsers.comma - parsers.semicolon)
6465                           * -#((parsers.internal_punctuation - parsers.comma
6466                                - parsers.semicolon)^0
6467                               * -(citation_chars + parsers.internal_punctuation
6468                                  - parsers.comma - parsers.semicolon)))^0
6469                          * citation_chars)^-1)
6470
6471      local citation_body_prenote
6472                    = Cs((parsers.alphanumeric^1
6473                         + parsers.bracketed
6474                         + parsers.inticks
6475                         + (parsers.anyescaped
6476                           - (parsers.rbracket + parsers.blankline^2))
6477                         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6478
6479      local citation_body_postnote
```

```
                    = Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + (parsers.anyescaped
                           - (parsers.rbracket + parsers.semicolon
                             + parsers.blankline^2))
                         - (parsers.spnl * parsers.rbracket))^0)

      local citation_body_chunk
                    = citation_body_prenote
                    * parsers.spnl * citation_name
                    * (parsers.internal_punctuation - parsers.semicolon)^-
1
                    * parsers.spnl * citation_body_postnote

      local citation_body
                    = citation_body_chunk
                    * (parsers.semicolon * parsers.spnl
                      * citation_body_chunk)^0

      local citation_headless_body_postnote
                    = Cs((parsers.alphanumeric^1
                         + parsers.bracketed
                         + parsers.inticks
                         + (parsers.anyescaped
                           - (parsers.rbracket + parsers.at
                             + parsers.semicolon + parsers.blankline^2))
                         - (parsers.spnl * parsers.rbracket))^0)

      local citation_headless_body
                    = citation_headless_body_postnote
                    * (parsers.sp * parsers.semicolon * parsers.spnl
                      * citation_body_chunk)^0

      local citations
                    = function(text_cites, raw_cites)
          local function normalize(str)
              if str == "" then
                  str = nil
              else
                  str = (citation_nbsps and
                    self.parser_functions.parse_inlines_nbsp or
                    self.parser_functions.parse_inlines)(str)
              end
              return str
          end
```

```
6526            local cites = {}
6527            for i = 1,#raw_cites,4 do
6528                cites[#cites+1] = {
6529                    prenote = normalize(raw_cites[i]),
6530                    suppress_author = raw_cites[i+1] == "-",
6531                    name = writer.citation(raw_cites[i+2]),
6532                    postnote = normalize(raw_cites[i+3]),
6533                }
6534            end
6535            return writer.citations(text_cites, cites)
6536        end
6537
6538        local TextCitations
6539                      = Ct((parsers.spnl
6540                      * Cc("")
6541                      * citation_name
6542                      * ((parsers.spnl
6543                          * parsers.lbracket
6544                          * citation_headless_body
6545                          * parsers.rbracket) + Cc("")))^1)
6546                      / function(raw_cites)
6547                          return citations(true, raw_cites)
6548                        end
6549
6550        local ParenthesizedCitations
6551                      = Ct((parsers.spnl
6552                      * parsers.lbracket
6553                      * citation_body
6554                      * parsers.rbracket)^1)
6555                      / function(raw_cites)
6556                          return citations(false, raw_cites)
6557                        end
6558
6559        local Citations = TextCitations + ParenthesizedCitations
6560
6561        self.insert_pattern("Inline after Emph",
6562                            Citations, "Citations")
6563
6564        self.add_special_character("@")
6565        self.add_special_character("-")
6566      end
6567    }
6568  end
```

3.1.6.3 Content Blocks

The `extensions.content_blocks` function implements the iA,Writer content blocks syntax extension. The `language_map` parameter specifies

the filename of the JSON file that maps filename extensions to programming language names.

```
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.

```
  local languages_json = (function()
    local base, prev, curr
    for _, pathname in ipairs{util.lookup_files(language_map, { all=true })} do
      local file = io.open(pathname, "r")
      if not file then goto continue end
      local input = assert(file:read("*a"))
      assert(file:close())
      local json = input:gsub('("[^\n]-"):','[%1]=')
      curr = load("_ENV = {}; return "..json)()
      if type(curr) == "table" then
        if base == nil then
          base = curr
        else
          setmetatable(prev, { __index = curr })
        end
        prev = curr
      end
      ::continue::
    end
    return base or {}
  end)()

  return {
    name = "built-in content_blocks syntax extension",
    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.

```
      function self.contentblock(src,suf,type,tit)
        if not self.is_writing then return "" end
        src = src.."."..suf
        suf = suf:lower()
        if type == "onlineimage" then
          return {"\\markdownRendererContentBlockOnlineImage{",suf,"}",
                                  "{",self.string(src),"}",
                                  "{",self.uri(src),"}",
                                  "{",self.string(tit or ""),"}"}
```

```
6604        elseif languages_json[suf] then
6605          return {"\\markdownRendererContentBlockCode{",suf,"}",
6606                                  "{",self.string(languages_json[suf]),"}",
6607                                  "{",self.string(src),"}",
6608                                  "{",self.uri(src),"}",
6609                                  "{",self.string(tit or ""),"}"}
6610        else
6611          return {"\\markdownRendererContentBlock{",suf,"}",
6612                                  "{",self.string(src),"}",
6613                                  "{",self.uri(src),"}",
6614                                  "{",self.string(tit or ""),"}"}
6615        end
6616      end
6617    end, extend_reader = function(self)
6618      local parsers = self.parsers
6619      local writer = self.writer
6620
6621      local contentblock_tail
6622                    = parsers.optionaltitle
6623                    * (parsers.newline + parsers.eof)
6624
6625      -- case insensitive online image suffix:
6626      local onlineimagesuffix
6627                    = (function(...)
6628                        local parser = nil
6629                        for _, suffix in ipairs({...}) do
6630                          local pattern=nil
6631                          for i=1,#suffix do
6632                            local char=suffix:sub(i,i)
6633                            char = S(char:lower()..char:upper())
6634                            if pattern == nil then
6635                              pattern = char
6636                            else
6637                              pattern = pattern * char
6638                            end
6639                          end
6640                          if parser == nil then
6641                            parser = pattern
6642                          else
6643                            parser = parser + pattern
6644                          end
6645                        end
6646                        return parser
6647                      end)("png", "jpg", "jpeg", "gif", "tif", "tiff")
6648
6649      -- online image url for iA Writer content blocks with mandatory suffix,
6650      -- allowing nested brackets:
```

```
      local onlineimageurl
                    = (parsers.less
                      * Cs((parsers.anyescaped
                           - parsers.more
                           - #(parsers.period
                              * onlineimagesuffix
                              * parsers.more
                              * contentblock_tail))^0)
                      * parsers.period
                      * Cs(onlineimagesuffix)
                      * parsers.more
                      + (Cs((parsers.inparens
                            + (parsers.anyescaped
                              - parsers.spacing
                              - parsers.rparent
                              - #(parsers.period
                                 * onlineimagesuffix
                                 * contentblock_tail)))^0)
                        * parsers.period
                        * Cs(onlineimagesuffix))
                      ) * Cc("onlineimage")

      -- filename for iA Writer content blocks with mandatory suffix:
      local localfilepath
                    = parsers.slash
                    * Cs((parsers.anyescaped
                         - parsers.tab
                         - parsers.newline
                         - #(parsers.period
                            * parsers.alphanumeric^1
                            * contentblock_tail))^1)
                    * parsers.period
                    * Cs(parsers.alphanumeric^1)
                    * Cc("localfile")

      local ContentBlock
                    = parsers.leader
                    * (localfilepath + onlineimageurl)
                    * contentblock_tail
                    / writer.contentblock

      self.insert_pattern("Block before Blockquote",
                          ContentBlock, "ContentBlock")
    end
  }
end
```

3.1.6.4 Definition Lists

The `extensions.definition_lists` function implements the Pandoc definition list syntax extension. If the `tight_lists` parameter is `true`, tight lists will produce special right item renderers.

```
M.extensions.definition_lists = function(tight_lists)
  return {
    name = "built-in definition_lists syntax extension",
    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.

```
      local function dlitem(term, defs)
        local retVal = {"\\markdownRendererDlItem{",term,"}"}
        for _, def in ipairs(defs) do
          retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
                               "\\markdownRendererDlDefinitionEnd "}
        end
        retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
        return retVal
      end

      function self.definitionlist(items,tight)
        if not self.is_writing then return "" end
        local buffer = {}
        for _,item in ipairs(items) do
          buffer[#buffer + 1] = dlitem(item.term, item.definitions)
        end
        if tight and tight_lists then
          return {"\\markdownRendererDlBeginTight\n", buffer,
            "\n\\markdownRendererDlEndTight"}
        else
          return {"\\markdownRendererDlBegin\n", buffer,
            "\n\\markdownRendererDlEnd"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local defstartchar = S("~:")

      local defstart = ( defstartchar * #parsers.spacing
                                      * (parsers.tab + parsers.space^-
    3)
                       + parsers.space * defstartchar * #parsers.spacing
```

```
                                        * (parsers.tab + parsers.space^-
2)
                        + parsers.space * parsers.space * defstartchar
                                        * #parsers.spacing
                                        * (parsers.tab + parsers.space^-
1)
                        + parsers.space * parsers.space * parsers.space
                                        * defstartchar * #parsers.spacing
                        )

      local dlchunk = Cs(parsers.line * (parsers.indentedline - parsers.blankline)^0)

      local function definition_list_item(term, defs, _)
        return { term = self.parser_functions.parse_inlines(term),
                 definitions = defs }
      end

      local DefinitionListItemLoose
                    = C(parsers.line) * parsers.skipblanklines
                    * Ct((defstart
                         * parsers.indented_blocks(dlchunk)
                         / self.parser_functions.parse_blocks_nested)^1)
                    * Cc(false) / definition_list_item

      local DefinitionListItemTight
                    = C(parsers.line)
                    * Ct((defstart * dlchunk
                         / self.parser_functions.parse_blocks_nested)^1)
                    * Cc(true) / definition_list_item

      local DefinitionList
                    = ( Ct(DefinitionListItemLoose^1) * Cc(false)
                      + Ct(DefinitionListItemTight^1)
                      * (parsers.skipblanklines
                        * -DefinitionListItemLoose * Cc(true))
                      ) / writer.definitionlist

      self.insert_pattern("Block after Heading",
                          DefinitionList, "DefinitionList")
    end
  }
end
```

3.1.6.5 Fancy Lists

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

```
M.extensions.fancy_lists = function()
```

```
 return {
   name = "built-in fancy_lists syntax extension",
   extend_writer = function(self)
     local options = self.options

```

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.

```
     function self.fancylist(items,tight,startnum,numstyle,numdelim)
       if not self.is_writing then return "" end
       local buffer = {}
       local num = startnum
       for _,item in ipairs(items) do
         buffer[#buffer + 1] = self.fancyitem(item,num)
         if num ~= nil then
           num = num + 1
         end
       end
       local contents = util.intersperse(buffer,"\n")
       if tight and options.tightLists then
         return {"\\markdownRendererFancyOlBeginTight{",
                 numstyle,"}{",numdelim,"}",contents,
                 "\n\\markdownRendererFancyOlEndTight "}
       else
         return {"\\markdownRendererFancyOlBegin{",
                 numstyle,"}{",numdelim,"}",contents,
                 "\n\\markdownRendererFancyOlEnd "}
       end
     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.

```
6801        function self.fancyitem(s,num)
6802          if num ~= nil then
6803            return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
6804                    "\\markdownRendererFancyOlItemEnd "}
6805          else
6806            return {"\\markdownRendererFancyOlItem ",s,"\\markdownRendererFancyOlItemEnd
6807          end
6808        end
6809      end, extend_reader = function(self)
6810        local parsers = self.parsers
6811        local options = self.options
6812        local writer = self.writer
6813
6814        local label = parsers.dig + parsers.letter
6815        local numdelim = parsers.period + parsers.rparent
6816        local enumerator = C(label^3 * numdelim) * #parsers.spacing
6817                         + C(label^2 * numdelim) * #parsers.spacing
6818                                                 * (parsers.tab + parsers.space^1)
6819                         + C(label * numdelim) * #parsers.spacing
6820                                               * (parsers.tab + parsers.space^-
    2)
6821                         + parsers.space * C(label^2 * numdelim)
6822                                               * #parsers.spacing
6823                         + parsers.space * C(label * numdelim)
6824                                               * #parsers.spacing
6825                                               * (parsers.tab + parsers.space^-
    1)
6826                         + parsers.space * parsers.space * C(label^1
6827                                               * numdelim) * #parsers.spacing
6828        local starter = parsers.bullet + enumerator
6829
6830        local NestedList = Cs((parsers.optionallyindentedline
6831                               - starter)^1)
6832                         / function(a) return "\001"..a end
6833
6834        local ListBlockLine  = parsers.optionallyindentedline
6835                             - parsers.blankline - (parsers.indent^-1
6836                                                    * starter)
6837
6838        local ListBlock = parsers.line * ListBlockLine^0
6839
6840        local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
6841                                    * ListBlock
6842
6843        local TightListItem = function(starter)
```

```
6844            return -parsers.ThematicBreak
6845                   * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
    1)
6846                     / self.parser_functions.parse_blocks_nested)
6847                   * -(parsers.blanklines * parsers.indent)
6848        end
6849
6850        local LooseListItem = function(starter)
6851            return -parsers.ThematicBreak
6852                   * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
6853                     * (NestedList + ListContinuationBlock^0)
6854                     * (parsers.blanklines / "\n\n")
6855                     ) / self.parser_functions.parse_blocks_nested
6856        end
6857
6858        local function roman2number(roman)
6859          local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
6860          local numeral = 0
6861
6862          local i = 1
6863          local len = string.len(roman)
6864          while i < len do
6865            local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
6866            if z1 < z2 then
6867                numeral = numeral + (z2 - z1)
6868                i = i + 2
6869            else
6870                numeral = numeral + z1
6871                i = i + 1
6872            end
6873          end
6874          if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
6875          return numeral
6876        end
6877
6878        local function sniffstyle(itemprefix)
6879          local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.)]*)")
6880          local numdelim
6881          if delimend == ")" then
6882            numdelim = "OneParen"
6883          elseif delimend == "." then
6884            numdelim = "Period"
6885          else
6886            numdelim = "Default"
6887          end
6888          numstr = numstr or itemprefix
6889
```

```
6890          local num
6891          num = numstr:match("^([IVXL]+)")
6892          if num then
6893            return roman2number(num), "UpperRoman", numdelim
6894          end
6895          num = numstr:match("^([ivxl]+)")
6896          if num then
6897            return roman2number(string.upper(num)), "LowerRoman", numdelim
6898          end
6899          num = numstr:match("^([A-Z])")
6900          if num then
6901            return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
6902          end
6903          num = numstr:match("^([a-z])")
6904          if num then
6905            return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
6906          end
6907          return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
6908        end
6909
6910        local function fancylist(items,tight,start)
6911          local startnum, numstyle, numdelim = sniffstyle(start)
6912          return writer.fancylist(items,tight,
6913                                  options.startNumber and startnum,
6914                                  numstyle or "Decimal",
6915                                  numdelim or "Default")
6916        end
6917
6918        local FancyList = Cg(enumerator, "listtype") *
6919                          ( Ct(TightListItem(Cb("listtype"))
6920                              * TightListItem(enumerator)^0)
6921                          * Cc(true) * parsers.skipblanklines * -enumerator
6922                          + Ct(LooseListItem(Cb("listtype"))
6923                              * LooseListItem(enumerator)^0)
6924                          * Cc(false) * parsers.skipblanklines
6925                          ) * Cb("listtype") / fancylist
6926
6927        self.update_rule("OrderedList", function() return FancyList end)
6928      end
6929    }
6930  end
```

3.1.6.6 Fenced Code

The `extensions.fenced_code` function implements the commonmark fenced code block syntax extension. When the `blank_before_code_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

When the `allow_attributes` option is `true`, the syntax extension permits attributes following the infostring. When the `allow_raw_blocks` option is `true`, the syntax extension permits the specification of raw blocks using Pandoc's raw attribute syntax extension.

```
M.extensions.fenced_code = function(blank_before_code_fence,
                                    allow_attributes,
                                    allow_raw_blocks)
  return {
    name = "built-in fenced_code syntax extension",
    extend_writer = function(self)
      local options = self.options

```

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

```
      function self.fencedCode(s, i, attr)
        if not self.is_writing then return "" end
        s = s:gsub("\n$", "")
        local buf = {}
        if attr ~= nil then
          table.insert(buf, {"\\markdownRendererFencedCodeAttributeContextBegin",
                             self.attributes(attr)})
        end
        local name = util.cache_verbatim(options.cacheDir, s)
        table.insert(buf, {"\\markdownRendererInputFencedCode{",
                           name,"}{",self.string(i),"}"})
        if attr ~= nil then
          table.insert(buf, "\\markdownRendererFencedCodeAttributeContextEnd")
        end
        return buf
      end

```

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

```
      if allow_raw_blocks then
        function self.rawBlock(s, attr)
          if not self.is_writing then return "" end
          s = s:gsub("\n$", "")
          local name = util.cache_verbatim(options.cacheDir, s)
          return {"\\markdownRendererInputRawBlock{",
                  name,"}{", self.string(attr),"}"}
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer
```

```
6968
6969        local function captures_geq_length(_,i,a,b)
6970          return #a >= #b and i
6971        end
6972
6973        local tilde_infostring
6974                              = C((parsers.linechar
6975                                 - (parsers.spacechar^1 * parsers.newline))^0)
6976
6977        local backtick_infostring
6978                              = C((parsers.linechar
6979                                 - (parsers.backtick
6980                                   + parsers.spacechar^1 * parsers.newline))^0)
6981
6982        local fenceindent
6983        local fencehead       = function(char, infostring)
6984          return                C(parsers.nonindentspace) / function(s) fenceindent = #s
6985                              * Cg(char^3, "fencelength")
6986                              * parsers.optionalspace
6987                              * infostring
6988                              * (parsers.newline + parsers.eof)
6989        end
6990
6991        local fencetail       = function(char)
6992          return                parsers.nonindentspace
6993                              * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
6994                              * parsers.optionalspace * (parsers.newline + parsers.eof)
6995                              + parsers.eof
6996        end
6997
6998        local fencedline      = function(char)
6999          return                C(parsers.line - fencetail(char))
7000                              / function(s)
7001                                  local i = 1
7002                                  local remaining = fenceindent
7003                                  while true do
7004                                    local c = s:sub(i, i)
7005                                    if c == " " and remaining > 0 then
7006                                      remaining = remaining - 1
7007                                      i = i + 1
7008                                    elseif c == "\t" and remaining > 3 then
7009                                      remaining = remaining - 4
7010                                      i = i + 1
7011                                    else
7012                                      break
7013                                    end
7014                                  end
```

```
                              return s:sub(i)
                            end
      end

      local TildeFencedCode
             = fencehead(parsers.tilde, tilde_infostring)
             * Cs(fencedline(parsers.tilde)^0)
             * fencetail(parsers.tilde)

      local BacktickFencedCode
             = fencehead(parsers.backtick, backtick_infostring)
             * Cs(fencedline(parsers.backtick)^0)
             * fencetail(parsers.backtick)

            local infostring_with_attributes
                            = Ct(C((parsers.linechar
                                   - ( parsers.optionalspace
                                     * parsers.attributes))^0)
                                 * parsers.optionalspace
                                 * Ct(parsers.attributes))

      local FencedCode
               = (TildeFencedCode + BacktickFencedCode)
               / function(infostring, code)
                   local expanded_code = self.expandtabs(code)

                   if allow_raw_blocks then
                     local raw_attr = lpeg.match(parsers.raw_attribute,
                                                 infostring)
                     if raw_attr then
                       return writer.rawBlock(expanded_code, raw_attr)
                     end
                   end

                   local attr = nil
                   if allow_attributes then
                     local match = lpeg.match(infostring_with_attributes,
                                              infostring)
                     if match then
                       infostring, attr = table.unpack(match)
                     end
                   end
                   return writer.fencedCode(expanded_code, infostring, attr)
                 end

      self.insert_pattern("Block after Verbatim",
                          FencedCode, "FencedCode")
```

```
7062
7063        local fencestart
7064        if blank_before_code_fence then
7065          fencestart = parsers.fail
7066        else
7067          fencestart = fencehead(parsers.backtick, backtick_infostring)
7068                     + fencehead(parsers.tilde, tilde_infostring)
7069        end
7070
7071        self.update_rule("EndlineExceptions", function(previous_pattern)
7072          if previous_pattern == nil then
7073            previous_pattern = parsers.EndlineExceptions
7074          end
7075          return previous_pattern + fencestart
7076        end)
7077
7078        self.add_special_character("`")
7079        self.add_special_character("~")
7080      end
7081    }
7082  end
```

3.1.6.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced divs syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```
7083  M.extensions.fenced_divs = function(blank_before_div_fence)
7084    return {
7085      name = "built-in fenced_divs syntax extension",
7086      extend_writer = function(self)
```

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

```
7087        function self.div(c, attr)
7088          return {"\\markdownRendererFencedDivAttributeContextBegin",
7089                  self.attributes(attr),
7090                  c,
7091                  "\\markdownRendererFencedDivAttributeContextEnd"}
7092        end
7093      end, extend_reader = function(self)
7094        local parsers = self.parsers
7095        local writer = self.writer
```

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

```
7096        local fenced_div_infostring
7097                                  = C((parsers.linechar
7098                                     - ( parsers.spacechar^1
```

```
                                  * parsers.colon^1))^1)

      local fenced_div_begin = parsers.nonindentspace
                             * parsers.colon^3
                             * parsers.optionalspace
                             * fenced_div_infostring
                             * ( parsers.spacechar^1
                               * parsers.colon^1)^0
                             * parsers.optionalspace
                             * (parsers.newline + parsers.eof)

      local fenced_div_end = parsers.nonindentspace
                           * parsers.colon^3
                           * parsers.optionalspace
                           * (parsers.newline + parsers.eof)
```

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

```
      self.initialize_named_group("div_level", "0")

      local function increment_div_level(increment)
        local function update_div_level(s, i, current_level) -- luacheck: ignore s i
          current_level = tonumber(current_level)
          local next_level = tostring(current_level + increment)
          return true, next_level
        end

        return Cg( Cmt(Cb("div_level"), update_div_level)
                 , "div_level")
      end

      local FencedDiv = fenced_div_begin * increment_div_level(1)
                      * parsers.skipblanklines
                      * Ct( (V("Block") - fenced_div_end)^-1
                          * ( parsers.blanklines
                            / function()
                                return writer.interblocksep
                              end
                            * (V("Block") - fenced_div_end))^0)
                      * parsers.skipblanklines
                      * fenced_div_end * increment_div_level(-1)
                      / function (infostring, div)
                          local attr = lpeg.match(Ct(parsers.attributes), infostring)
                          if attr == nil then
                            attr = {"." .. infostring}
                          end
                          return div, attr
```

```
                        end
                      / writer.div

      self.insert_pattern("Block after Verbatim",
                          FencedDiv, "FencedDiv")

      self.add_special_character(":")

```

Patch blockquotes, so that they allow the end of a fenced div immediately afterwards.

```
      local function check_div_level(s, i, current_level) -- luacheck: ignore s i
        current_level = tonumber(current_level)
        return current_level > 0
      end

      local is_inside_div = Cmt(Cb("div_level"), check_div_level)
      local fencestart = is_inside_div * fenced_div_end

      self.update_rule("BlockquoteExceptions", function(previous_pattern)
        if previous_pattern == nil then
          previous_pattern = parsers.BlockquoteExceptions
        end
        return previous_pattern + fencestart
      end)

```

If the `blank_before_div_fence` parameter is `false`, we will have the closing div at the beginning of a line break the current paragraph if we are currently nested in a div.

```
      if not blank_before_div_fence then
        self.update_rule("EndlineExceptions", function(previous_pattern)
          if previous_pattern == nil then
            previous_pattern = parsers.EndlineExceptions
          end
          return previous_pattern + fencestart
        end)
      end
    end
  }
end
```

3.1.6.8 Header Attributes

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

```
M.extensions.header_attributes = function()
  return {
    name = "built-in header_attributes syntax extension",
```

```
  extend_writer = function()
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local AtxHeading = Cg(parsers.heading_start, "level")
                       * parsers.optionalspace
                       * (C(((parsers.linechar
                             - ((parsers.hash^1
                                 * parsers.optionalspace
                                 * parsers.attributes^-1
                                 + parsers.attributes)
                                * parsers.optionalspace
                                * parsers.newline))
                            * (parsers.linechar
                              - parsers.hash
                              - parsers.lbrace)^0)^1)
                           / self.parser_functions.parse_inlines)
                       * Cg(Ct(parsers.newline
                             + (parsers.hash^1
                               * parsers.optionalspace
                               * parsers.attributes^-1
                               + parsers.attributes)
                             * parsers.optionalspace
                             * parsers.newline), "attributes")
                       * Cb("level")
                       * Cb("attributes")
                       / writer.heading

    local SetextHeading = #(parsers.line * S("=-"))
                          * (C(((parsers.linechar
                                - (parsers.attributes
                                  * parsers.optionalspace
                                  * parsers.newline))
                               * (parsers.linechar
                                 - parsers.lbrace)^0)^1)
                              / self.parser_functions.parse_inlines)
                          * Cg(Ct(parsers.newline
                                + (parsers.attributes
                                  * parsers.optionalspace
                                  * parsers.newline)), "attributes")
                          * parsers.heading_level
                          * Cb("attributes")
                          * parsers.optionalspace
                          * parsers.newline
                          / writer.heading

```

```
      local Heading = AtxHeading + SetextHeading
      self.update_rule("Heading", function() return Heading end)
    end
  }
end
```

3.1.6.9 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line blocks syntax extension.

```
M.extensions.line_blocks = function()
  return {
    name = "built-in line_blocks syntax extension",
    extend_writer = function(self)
```

Define `writer->lineblock` as a function that will transform a line block consisted of `lines` to the output format, with all but the last newline rendered as a line break.

```
      function self.lineblock(lines)
        if not self.is_writing then return "" end
        local buffer = {}
        for i = 1, #lines - 1 do
          buffer[#buffer + 1] = { lines[i], self.linebreak }
        end
        buffer[#buffer + 1] = lines[#lines]

        return {"\\markdownRendererLineBlockBegin\n"
                  ,buffer,
                  "\n\\markdownRendererLineBlockEnd "}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local LineBlock = Ct(
                          (Cs(
                            ( (parsers.pipe * parsers.space)/""
                            * ((parsers.space)/entities.char_entity("nbsp"))^0
                            * parsers.linechar^0 * (parsers.newline/""))
                            * (-parsers.pipe
                              * (parsers.space^1/" ")
                              * parsers.linechar^1
                              * (parsers.newline/"")
                              )^0
                            * (parsers.blankline/"")^0
                          ) / self.parser_functions.parse_inlines)^1) / writer.lineblock

      self.insert_pattern("Block after Blockquote",
                           LineBlock, "LineBlock")
    end
```

```
  }
end
```

3.1.6.10 Notes

The `extensions.notes` function implements the Pandoc note and inline note syntax extensions. When the `note` parameter is `true`, the Pandoc note syntax extension will be enabled. When the `inline_notes` parameter is `true`, the Pandoc inline note syntax extension will be enabled.

```
M.extensions.notes = function(notes, inline_notes)
  assert(notes or inline_notes)
  return {
    name = "built-in notes syntax extension",
    extend_writer = function(self)
```

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

```
      function self.note(s)
        return {"\\markdownRendererNote{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      if inline_notes then
        local InlineNote
                    = parsers.circumflex
                    * (parsers.tag / self.parser_functions.parse_inlines_no_inline_not
                    / writer.note

        self.insert_pattern("Inline after Emph",
                            InlineNote, "InlineNote")
      end
      if notes then
        local function strip_first_char(s)
          return s:sub(2)
        end

        local RawNoteRef
                      = #(parsers.lbracket * parsers.circumflex)
                      * parsers.tag / strip_first_char

        local rawnotes = {}

        -- like indirect_link
        local function lookup_note(ref)
          return writer.defer_call(function()
            local found = rawnotes[self.normalize_tag(ref)]
```

```
            if found then
              return writer.note(
                self.parser_functions.parse_blocks_nested(found))
            else
              return {"[",
                self.parser_functions.parse_inlines("^" .. ref), "]"}
            end
          end)
        end

        local function register_note(ref,rawnote)
          rawnotes[self.normalize_tag(ref)] = rawnote
          return ""
        end

        local NoteRef = RawNoteRef / lookup_note

        local NoteBlock
                    = parsers.leader * RawNoteRef * parsers.colon
                    * parsers.spnl * parsers.indented_blocks(parsers.chunk)
                    / register_note

        local Blank = NoteBlock + parsers.Blank
        self.update_rule("Blank", function() return Blank end)

        self.insert_pattern("Inline after Emph",
                            NoteRef, "NoteRef")
      end

      self.add_special_character("^")
    end
  }
end
```

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

```
M.extensions.pipe_tables = function(table_captions)

  local function make_pipe_table_rectangular(rows)
    local num_columns = #rows[2]
    local rectangular_rows = {}
    for i = 1, #rows do
      local row = rows[i]
      local rectangular_row = {}
```

```
7347      for j = 1, num_columns do
7348        rectangular_row[j] = row[j] or ""
7349      end
7350      table.insert(rectangular_rows, rectangular_row)
7351    end
7352    return rectangular_rows
7353  end
7354
7355  local function pipe_table_row(allow_empty_first_column
7356                              , nonempty_column
7357                              , column_separator
7358                              , column)
7359    local row_beginning
7360    if allow_empty_first_column then
7361      row_beginning = -- empty first column
7362                      #(parsers.spacechar^4
7363                        * column_separator)
7364                    * parsers.optionalspace
7365                    * column
7366                    * parsers.optionalspace
7367                    -- non-empty first column
7368                    + parsers.nonindentspace
7369                    * nonempty_column^-1
7370                    * parsers.optionalspace
7371    else
7372      row_beginning = parsers.nonindentspace
7373                    * nonempty_column^-1
7374                    * parsers.optionalspace
7375    end
7376
7377    return Ct(row_beginning
7378              * (-- single column with no leading pipes
7379                 #(column_separator
7380                   * parsers.optionalspace
7381                   * parsers.newline)
7382                 * column_separator
7383                 * parsers.optionalspace
7384                 -- single column with leading pipes or
7385                 -- more than a single column
7386                 + (column_separator
7387                   * parsers.optionalspace
7388                   * column
7389                   * parsers.optionalspace)^1
7390                 * (column_separator
7391                   * parsers.optionalspace)^-1))
7392  end
7393
```

```
  return {
    name = "built-in pipe_tables syntax extension",
    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.

```
      function self.table(rows, caption)
        if not self.is_writing then return "" end
        local buffer = {"\\markdownRendererTable{",
          caption or "", "}{", #rows - 1, "}{", #rows[1], "}"}
        local temp = rows[2] -- put alignments on the first row
        rows[2] = rows[1]
        rows[1] = temp
        for i, row in ipairs(rows) do
          table.insert(buffer, "{")
          for _, column in ipairs(row) do
            if i > 1 then -- do not use braces for alignments
              table.insert(buffer, "{")
            end
            table.insert(buffer, column)
            if i > 1 then
              table.insert(buffer, "}")
            end
          end
          table.insert(buffer, "}")
        end
        return buffer
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local table_hline_separator = parsers.pipe + parsers.plus

      local table_hline_column = (parsers.dash
                                  - #(parsers.dash
                                     * (parsers.spacechar
                                       + table_hline_separator
                                       + parsers.newline)))^1
                               * (parsers.colon * Cc("r")
                                 + parsers.dash * Cc("d"))
                               + parsers.colon
                               * (parsers.dash
                                 - #(parsers.dash
                                    * (parsers.spacechar
                                      + table_hline_separator
```

```
                                      + parsers.newline)))^1
                                * (parsers.colon * Cc("c")
                                  + parsers.dash * Cc("l"))

      local table_hline = pipe_table_row(false
                                         , table_hline_column
                                         , table_hline_separator
                                         , table_hline_column)

      local table_caption_beginning = parsers.skipblanklines
                                    * parsers.nonindentspace
                                    * (P("Table")^-1 * parsers.colon)
                                    * parsers.optionalspace

      local table_row = pipe_table_row(true
                                       , (C((parsers.linechar - parsers.pipe)^1)
                                         / self.parser_functions.parse_inlines)
                                       , parsers.pipe
                                       , (C((parsers.linechar - parsers.pipe)^0)
                                         / self.parser_functions.parse_inlines))

      local table_caption
      if table_captions then
        table_caption = #table_caption_beginning
                      * table_caption_beginning
                      * Ct(parsers.IndentedInline^1)
                      * parsers.newline
      else
        table_caption = parsers.fail
      end

      local PipeTable = Ct(table_row * parsers.newline
                           * table_hline
                           * (parsers.newline * table_row)^0)
                      / make_pipe_table_rectangular
                      * table_caption^-1
                      / writer.table

      self.insert_pattern("Block after Blockquote",
                          PipeTable, "PipeTable")
    end
  }
end
```

3.1.6.12 Raw Attributes The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
M.extensions.raw_inline = function()
  return {
    name = "built-in raw_inline syntax extension",
    extend_writer = function(self)
      local options = self.options

```

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

```
      function self.rawInline(s, attr)
        if not self.is_writing then return "" end
        local name = util.cache_verbatim(options.cacheDir, s)
        return {"\\markdownRendererInputRawInline{",
                name,"}{", self.string(attr),"}"}
      end
    end, extend_reader = function(self)
      local writer = self.writer

      local RawInline = parsers.inticks
                      * parsers.raw_attribute
                      / writer.rawInline

      self.insert_pattern("Inline before Code",
                          RawInline, "RawInline")
    end
  }
end
```

3.1.6.13 Strike-Through

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

```
M.extensions.strike_through = function()
  return {
    name = "built-in strike_through syntax extension",
    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.

```
      function self.strike_through(s)
        return {"\\markdownRendererStrikeThrough{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local StrikeThrough = (
        parsers.between(parsers.Inline, parsers.doubletildes,
                        parsers.doubletildes)
```

```
7518       ) / writer.strike_through
7519
7520       self.insert_pattern("Inline after Emph",
7521                           StrikeThrough, "StrikeThrough")
7522
7523       self.add_special_character("~")
7524     end
7525   }
7526 end
```

3.1.6.14 Subscripts

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

```
7527 M.extensions.subscripts = function()
7528   return {
7529     name = "built-in subscripts syntax extension",
7530     extend_writer = function(self)
```

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

```
7531       function self.subscript(s)
7532         return {"\\markdownRendererSubscript{",s,"}"}
7533       end
7534     end, extend_reader = function(self)
7535       local parsers = self.parsers
7536       local writer = self.writer
7537
7538       local Subscript = (
7539         parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
7540       ) / writer.subscript
7541
7542       self.insert_pattern("Inline after Emph",
7543                           Subscript, "Subscript")
7544
7545       self.add_special_character("~")
7546     end
7547   }
7548 end
```

3.1.6.15 Superscripts

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

```
7549 M.extensions.superscripts = function()
7550   return {
7551     name = "built-in superscripts syntax extension",
7552     extend_writer = function(self)
```

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

```
      function self.superscript(s)
        return {"\\markdownRendererSuperscript{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Superscript = (
        parsers.between(parsers.Str, parsers.circumflex, parsers.circumflex)
      ) / writer.superscript

      self.insert_pattern("Inline after Emph",
                          Superscript, "Superscript")

      self.add_special_character("^")
    end
  }
end
```

3.1.6.16 YAML Metadata

The `extensions.jekyll_data` function implements the Pandoc `yaml_metadata_block` syntax extension. When the `expect_jekyll_data` parameter is `true`, then a markdown document may begin directly with YAML metadata and may contain nothing but YAML metadata.

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

```
      function self.jekyllData(d, t, p)
        if not self.is_writing then return "" end

        local buf = {}

        local keys = {}
        for k, _ in pairs(d) do
          table.insert(keys, k)
        end
        table.sort(keys)

```

```
7586          if not p then
7587            table.insert(buf, "\\markdownRendererJekyllDataBegin")
7588          end
7589
7590          if #d > 0 then
7591              table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
7592              table.insert(buf, self.uri(p or "null"))
7593              table.insert(buf, "}{")
7594              table.insert(buf, #keys)
7595              table.insert(buf, "}")
7596          else
7597              table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
7598              table.insert(buf, self.uri(p or "null"))
7599              table.insert(buf, "}{")
7600              table.insert(buf, #keys)
7601              table.insert(buf, "}")
7602          end
7603
7604          for _, k in ipairs(keys) do
7605            local v = d[k]
7606            local typ = type(v)
7607            k = tostring(k or "null")
7608            if typ == "table" and next(v) ~= nil then
7609              table.insert(
7610                buf,
7611                self.jekyllData(v, t, k)
7612              )
7613            else
7614              k = self.uri(k)
7615              v = tostring(v)
7616              if typ == "boolean" then
7617                table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
7618                table.insert(buf, k)
7619                table.insert(buf, "}{")
7620                table.insert(buf, v)
7621                table.insert(buf, "}")
7622              elseif typ == "number" then
7623                table.insert(buf, "\\markdownRendererJekyllDataNumber{")
7624                table.insert(buf, k)
7625                table.insert(buf, "}{")
7626                table.insert(buf, v)
7627                table.insert(buf, "}")
7628              elseif typ == "string" then
7629                table.insert(buf, "\\markdownRendererJekyllDataString{")
7630                table.insert(buf, k)
7631                table.insert(buf, "}{")
7632                table.insert(buf, t(v))
```

```
              table.insert(buf, "}")
            elseif typ == "table" then
              table.insert(buf, "\\markdownRendererJekyllDataEmpty{")
              table.insert(buf, k)
              table.insert(buf, "}")
            else
              error(format("Unexpected type %s for value of " ..
                           "YAML key %s", typ, k))
            end
          end
        end

        if #d > 0 then
          table.insert(buf, "\\markdownRendererJekyllDataSequenceEnd")
        else
          table.insert(buf, "\\markdownRendererJekyllDataMappingEnd")
        end

        if not p then
          table.insert(buf, "\\markdownRendererJekyllDataEnd")
        end

        return buf
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local JekyllData
                    = Cmt( C((parsers.line - P("---") - P("..."))^0)
                         , function(s, i, text) -- luacheck: ignore s i
                             local data
                             local ran_ok, _ = pcall(function()
                               local tinyyaml = require("markdown-tinyyaml")
                               data = tinyyaml.parse(text, {timestamps=false})
                             end)
                             if ran_ok and data ~= nil then
                               return true, writer.jekyllData(data, function(s)
                                 return self.parser_functions.parse_blocks_nested(s)
                               end, nil)
                             else
                               return false
                             end
                           end
                         )

      local UnexpectedJekyllData
```

```
                    = P("---")
                    * parsers.blankline / 0
                    * #(-parsers.blankline)  -- if followed by blank, it's thematic br
                    * JekyllData
                    * (P("---") + P("..."))

      local ExpectedJekyllData
                    = ( P("---")
                      * parsers.blankline / 0
                      * #(-parsers.blankline)  -- if followed by blank, it's thematic
                      )^-1
                    * JekyllData
                    * (P("---") + P("..."))^-1

      self.insert_pattern("Block before Blockquote",
                          UnexpectedJekyllData, "UnexpectedJekyllData")
      if expect_jekyll_data then
        self.update_rule("ExpectedJekyllData", function() return ExpectedJekyllData en
      end
    end
  }
end
```

3.1.7 Conversion from Markdown to Plain TEX

The `new` function returns a conversion function that takes a markdown string and turns it into a plain TEX output. See Section 2.1.1.

```
function M.new(options)
```

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

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

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

```
  local extensions = {}

  if options.bracketedSpans then
    local bracketed_spans_extension = M.extensions.bracketed_spans()
    table.insert(extensions, bracketed_spans_extension)
  end

  if options.contentBlocks then
    local content_blocks_extension = M.extensions.content_blocks(
      options.contentBlocksLanguageMap)
    table.insert(extensions, content_blocks_extension)
  end
```

```
7718
7719  if options.definitionLists then
7720    local definition_lists_extension = M.extensions.definition_lists(
7721      options.tightLists)
7722    table.insert(extensions, definition_lists_extension)
7723  end
7724
7725  if options.fencedCode then
7726    local fenced_code_extension = M.extensions.fenced_code(
7727      options.blankBeforeCodeFence,
7728      options.fencedCodeAttributes,
7729      options.rawAttribute)
7730    table.insert(extensions, fenced_code_extension)
7731  end
7732
7733  if options.fencedDivs then
7734    local fenced_div_extension = M.extensions.fenced_divs(
7735      options.blankBeforeDivFence)
7736    table.insert(extensions, fenced_div_extension)
7737  end
7738
7739  if options.headerAttributes then
7740    local header_attributes_extension = M.extensions.header_attributes()
7741    table.insert(extensions, header_attributes_extension)
7742  end
7743
7744  if options.jekyllData then
7745    local jekyll_data_extension = M.extensions.jekyll_data(
7746      options.expectJekyllData)
7747    table.insert(extensions, jekyll_data_extension)
7748  end
7749
7750  if options.pipeTables then
7751    local pipe_tables_extension = M.extensions.pipe_tables(
7752      options.tableCaptions)
7753    table.insert(extensions, pipe_tables_extension)
7754  end
7755
7756  if options.rawAttribute then
7757    local raw_inline_extension = M.extensions.raw_inline()
7758    table.insert(extensions, raw_inline_extension)
7759  end
7760
7761  if options.strikeThrough then
7762    local strike_through_extension = M.extensions.strike_through()
7763    table.insert(extensions, strike_through_extension)
7764  end
```

```
7765
7766  if options.subscripts then
7767    local subscript_extension = M.extensions.subscripts()
7768    table.insert(extensions, subscript_extension)
7769  end
7770
7771  if options.superscripts then
7772    local superscript_extension = M.extensions.superscripts()
7773    table.insert(extensions, superscript_extension)
7774  end
7775
7776  if options.lineBlocks then
7777    local line_block_extension = M.extensions.line_blocks()
7778    table.insert(extensions, line_block_extension)
7779  end
7780
```

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

```
7781  if options.footnotes or options.inlineFootnotes or
7782     options.notes or options.inlineNotes then
7783    local notes_extension = M.extensions.notes(
7784      options.footnotes or options.notes,
7785      options.inlineFootnotes or options.inlineNotes)
7786    table.insert(extensions, notes_extension)
7787  end
7788
7789  if options.citations then
7790    local citations_extension = M.extensions.citations(options.citationNbsps)
7791    table.insert(extensions, citations_extension)
7792  end
7793
7794  if options.fancyLists then
7795    local fancy_lists_extension = M.extensions.fancy_lists()
7796    table.insert(extensions, fancy_lists_extension)
7797  end
```

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

```
7798  for _, user_extension_filename in ipairs(options.extensions) do
7799    local user_extension = (function(filename)
```

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

```
7800      local pathname = util.lookup_files(filename)
7801      local input_file = assert(io.open(pathname, "r"),
7802        [[Could not open user-defined syntax extension "]]
7803        .. pathname .. [[" for reading]])
7804      local input = assert(input_file:read("*a"))
7805      assert(input_file:close())
```

```
7806        local user_extension, err = load([[
7807          local sandbox = {}
7808          setmetatable(sandbox, {__index = _G})
7809          _ENV = sandbox
7810        ]] .. input)()
7811        assert(user_extension,
7812          [[Failed to compile user-defined syntax extension "]]
7813          .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
7814        assert(user_extension.api_version ~= nil,
7815          [[User-defined syntax extension "]] .. pathname
7816          .. [[" does not specify mandatory field "api_version"]])
7817        assert(type(user_extension.api_version) == "number",
7818          [[User-defined syntax extension "]] .. pathname
7819          .. [[" specifies field "api_version" of type "]]
7820          .. type(user_extension.api_version)
7821          .. [[" but "number" was expected]])
7822        assert(user_extension.api_version > 0
7823           and user_extension.api_version <= metadata.user_extension_api_version,
7824          [[User-defined syntax extension "]] .. pathname
7825          .. [[" uses syntax extension API version "]]
7826          .. user_extension.api_version .. [[ but markdown.lua ]]
7827          .. metadata.version .. [[ uses API version ]]
7828          .. metadata.user_extension_api_version
7829          .. [[, which is incompatible]])
7830
7831        assert(user_extension.grammar_version ~= nil,
7832          [[User-defined syntax extension "]] .. pathname
7833          .. [[" does not specify mandatory field "grammar_version"]])
7834        assert(type(user_extension.grammar_version) == "number",
7835          [[User-defined syntax extension "]] .. pathname
7836          .. [[" specifies field "grammar_version" of type "]]
7837          .. type(user_extension.grammar_version)
7838          .. [[" but "number" was expected]])
7839        assert(user_extension.grammar_version == metadata.grammar_version,
7840          [[User-defined syntax extension "]] .. pathname
7841          .. [[" uses grammar version "]] .. user_extension.grammar_version
7842          .. [[ but markdown.lua ]] .. metadata.version
7843          .. [[ uses grammar version ]] .. metadata.grammar_version
7844          .. [[, which is incompatible]])
7845
7846        assert(user_extension.finalize_grammar ~= nil,
7847          [[User-defined syntax extension "]] .. pathname
7848          .. [[" does not specify mandatory "finalize_grammar" field]])
7849        assert(type(user_extension.finalize_grammar) == "function",
7850          [[User-defined syntax extension "]] .. pathname
7851          .. [[" specifies field "finalize_grammar" of type "]]
```

```
        .. type(user_extension.finalize_grammar)
        .. [[" but "function" was expected]])
```

Finally, cast the user-defined syntax extension to the internal format of user extensions used by the Markdown package (see Section 3.1.6.)

```
      local extension = {
        name = [[user-defined "]] .. pathname .. [[" syntax extension]],
        extend_reader = user_extension.finalize_grammar,
        extend_writer = function() end,
      }
      return extension
    end)(user_extension_filename)
    table.insert(extensions, user_extension)
  end
```

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

```
  local writer = M.writer.new(options)
  local reader = M.reader.new(writer, options)
  local convert = reader.finalize_grammar(extensions)

  return convert
end

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.

```

local input
if input_filename then
  local input_file = assert(io.open(input_filename, "r"),
    [[Could not open file "]] .. input_filename .. [[" for reading]])
  input = assert(input_file:read("*a"))
  assert(input_file:close())
else
  input = assert(io.read("*a"))
end

```

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

```
local lfs = require("lfs")
if options.cacheDir and not lfs.isdir(options.cacheDir) then
  assert(lfs.mkdir(options["cacheDir"]))
end

local ran_ok, kpse = pcall(require, "kpse")
```

```
if ran_ok then kpse.set_program_name("luatex") end
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.

```
if metadata.version ~= md.metadata.version then
  warn("markdown-cli.lua " .. metadata.version .. " used with " ..
       "markdown.lua " .. md.metadata.version .. ".")
end
local convert = md.new(options)
local output = convert(input)

if output_filename then
  local output_file = assert(io.open(output_filename, "w"),
    [[Could not open file "]] .. output_filename .. [[" for writing]])
  assert(output_file:write(output))
  assert(output_file:close())
else
  assert(io.write(output))
end
```

3.2 Plain TEX Implementation

The plain TEX implementation provides macros for the interfacing between TEX and Lua and for the buffering of input text. These macros are then used to implement the macros for the conversion from markdown to plain TEX exposed by the plain TEX interface (see Section 2.2).

3.2.1 Logging Facilities

```
\ifx\markdownInfo\undefined
  \def\markdownInfo#1{%
    \immediate\write-1{(l.\the\inputlineno) markdown.tex info: #1.}}%
\fi
\ifx\markdownWarning\undefined
  \def\markdownWarning#1{%
    \immediate\write16{(l.\the\inputlineno) markdown.tex warning: #1}}%
\fi
\ifx\markdownError\undefined
  \def\markdownError#1#2{%
    \errhelp{#2.}%
    \errmessage{(l.\the\inputlineno) markdown.tex error: #1}}%
\fi
```

3.2.2 Token Renderer Prototypes

The following definitions should be considered placeholder.

```
\def\markdownRendererInterblockSeparatorPrototype{\par}%
\def\markdownRendererLineBreakPrototype{\hfil\break}%
\let\markdownRendererEllipsisPrototype\dots
\def\markdownRendererNbspPrototype{~}%
\def\markdownRendererLeftBracePrototype{\char`\{}%
\def\markdownRendererRightBracePrototype{\char`\}}%
\def\markdownRendererDollarSignPrototype{\char`$}%
\def\markdownRendererPercentSignPrototype{\char`\%}%
\def\markdownRendererAmpersandPrototype{\&}%
\def\markdownRendererUnderscorePrototype{\char`_}%
\def\markdownRendererHashPrototype{\char`\#}%
\def\markdownRendererCircumflexPrototype{\char`^}%
\def\markdownRendererBackslashPrototype{\char`\\}%
\def\markdownRendererTildePrototype{\char`~}%
\def\markdownRendererPipePrototype{|}%
\def\markdownRendererCodeSpanPrototype#1{{\tt#1}}%
\def\markdownRendererLinkPrototype#1#2#3#4{#2}%
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \markdownInput{#3}}%
\def\markdownRendererContentBlockOnlineImagePrototype{%
  \markdownRendererImage}%
\def\markdownRendererContentBlockCodePrototype#1#2#3#4#5{%
  \markdownRendererInputFencedCode{#3}{#2}}%
\def\markdownRendererImagePrototype#1#2#3#4{#2}%
\def\markdownRendererUlBeginPrototype{}%
\def\markdownRendererUlBeginTightPrototype{}%
\def\markdownRendererUlItemPrototype{}%
\def\markdownRendererUlItemEndPrototype{}%
\def\markdownRendererUlEndPrototype{}%
\def\markdownRendererUlEndTightPrototype{}%
\def\markdownRendererOlBeginPrototype{}%
\def\markdownRendererOlBeginTightPrototype{}%
\def\markdownRendererFancyOlBeginPrototype#1#2{\markdownRendererOlBegin}%
\def\markdownRendererFancyOlBeginTightPrototype#1#2{\markdownRendererOlBeginTight}%
\def\markdownRendererOlItemPrototype{}%
\def\markdownRendererOlItemWithNumberPrototype#1{}%
\def\markdownRendererOlItemEndPrototype{}%
\def\markdownRendererFancyOlItemPrototype{\markdownRendererOlItem}%
\def\markdownRendererFancyOlItemWithNumberPrototype{\markdownRendererOlItemWithNumber}
\def\markdownRendererFancyOlItemEndPrototype{}%
\def\markdownRendererOlEndPrototype{}%
\def\markdownRendererOlEndTightPrototype{}%
\def\markdownRendererFancyOlEndPrototype{\markdownRendererOlEnd}%
\def\markdownRendererFancyOlEndTightPrototype{\markdownRendererOlEndTight}%
\def\markdownRendererDlBeginPrototype{}%
\def\markdownRendererDlBeginTightPrototype{}%
\def\markdownRendererDlItemPrototype#1{#1}%
```

```
\def\markdownRendererDlItemEndPrototype{}%
\def\markdownRendererDlDefinitionBeginPrototype{}%
\def\markdownRendererDlDefinitionEndPrototype{\par}%
\def\markdownRendererDlEndPrototype{}%
\def\markdownRendererDlEndTightPrototype{}%
\def\markdownRendererEmphasisPrototype#1{{\it#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\begingroup\it}%
\def\markdownRendererBlockQuoteEndPrototype{\endgroup\par}%
\def\markdownRendererLineBlockBeginPrototype{\begingroup\parindent=0pt}%
\def\markdownRendererLineBlockEndPrototype{\endgroup}%
\def\markdownRendererInputVerbatimPrototype#1{%
  \par{\tt\input#1\relax{}}\par}%
\def\markdownRendererInputFencedCodePrototype#1#2{%
  \markdownRendererInputVerbatim{#1}}%
\def\markdownRendererHeadingOnePrototype#1{#1}%
\def\markdownRendererHeadingTwoPrototype#1{#1}%
\def\markdownRendererHeadingThreePrototype#1{#1}%
\def\markdownRendererHeadingFourPrototype#1{#1}%
\def\markdownRendererHeadingFivePrototype#1{#1}%
\def\markdownRendererHeadingSixPrototype#1{#1}%
\def\markdownRendererThematicBreakPrototype{}%
\def\markdownRendererNotePrototype#1{#1}%
\def\markdownRendererCitePrototype#1{}%
\def\markdownRendererTextCitePrototype#1{}%
\def\markdownRendererTickedBoxPrototype{[X]}%
\def\markdownRendererHalfTickedBoxPrototype{[/]}%
\def\markdownRendererUntickedBoxPrototype{[ ]}%
\def\markdownRendererStrikeThroughPrototype#1{#1}%
\def\markdownRendererSuperscriptPrototype#1{#1}%
\def\markdownRendererSubscriptPrototype#1{#1}%
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererHeaderAttributeContextBeginPrototype
  {
    \group_begin:
    \color_group_begin:
  }
\cs_gset:Npn
  \markdownRendererHeaderAttributeContextEndPrototype
  {
    \color_group_end:
    \group_end:
  }
\cs_gset_eq:NN
  \markdownRendererBracketedSpanAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
```

```
\cs_gset_eq:NN
  \markdownRendererBracketedSpanAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset_eq:NN
  \markdownRendererFencedDivAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererFencedDivAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset_eq:NN
  \markdownRendererFencedCodeAttributeContextBeginPrototype
  \markdownRendererHeaderAttributeContextBeginPrototype
\cs_gset_eq:NN
  \markdownRendererFencedCodeAttributeContextEndPrototype
  \markdownRendererHeaderAttributeContextEndPrototype
\cs_gset:Npn
  \markdownRendererReplacementCharacterPrototype
  {
    % TODO: Replace with `\codepoint_generate:nn` in TeX Live 2023
    \sys_if_engine_pdftex:TF
      { ^^ef^^bf^^bd }
      { ^^^^fffd }
  }
\ExplSyntaxOff
```

3.2.2.1 Raw Attribute Renderer Prototypes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
  {
    \str_case:nn
      { #2 }
      {
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} \unskip }
      }
  }
\cs_new:Nn
  \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
  {
    \str_case:nn
      { #2 }
      {
```

```
        { md  } { \markdownInput{#1}  }
        { tex } { \markdownEscape{#1} }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
      { #1 }
      { #2 }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
      { #1 }
      { #2 }
  }
\ExplSyntaxOff
```

3.2.2.2 YAML Metadata Renderer Prototypes

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

`\c_@@_jekyll_data_sequence_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.

```
\ExplSyntaxOn
\seq_new:N   \g_@@_jekyll_data_datatypes_seq
\tl_const:Nn \c_@@_jekyll_data_sequence_tl   { sequence }
\tl_const:Nn \c_@@_jekyll_data_mapping_tl    { mapping  }
\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.

```
\seq_new:N \g_@@_jekyll_data_wildcard_absolute_address_seq
\cs_new:Nn \markdown_jekyll_data_push_address_segment:n
  {
```

```
    \seq_if_empty:NF
      \g_@@_jekyll_data_datatypes_seq
      {
        \seq_get_right:NN
          \g_@@_jekyll_data_datatypes_seq
          \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.

```
        \str_if_eq:NNTF
          \l_tmpa_tl
          \c_@@_jekyll_data_sequence_tl
          {
            \seq_put_right:Nn
              \g_@@_jekyll_data_wildcard_absolute_address_seq
              { *  }
          }
          {
            \seq_put_right:Nn
              \g_@@_jekyll_data_wildcard_absolute_address_seq
              { #1 }
          }
      }
  }
```

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.

```
\tl_new:N  \g_@@_jekyll_data_wildcard_absolute_address_tl
\tl_new:N  \g_@@_jekyll_data_wildcard_relative_address_tl
\cs_new:Nn \markdown_jekyll_data_concatenate_address:NN
  {
    \seq_pop_left:NN #1 \l_tmpa_tl
    \tl_set:Nx #2 { / \seq_use:Nn #1 { / } }
    \seq_put_left:NV #1 \l_tmpa_tl
  }
\cs_new:Nn \markdown_jekyll_data_update_address_tls:
  {
    \markdown_jekyll_data_concatenate_address:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \g_@@_jekyll_data_wildcard_absolute_address_tl
    \seq_get_right:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \g_@@_jekyll_data_wildcard_relative_address_tl
  }
```

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.

```
\cs_new:Nn \markdown_jekyll_data_push:nN
  {
    \markdown_jekyll_data_push_address_segment:n
      { #1 }
    \seq_put_right:NV
     \g_@@_jekyll_data_datatypes_seq
     #2
    \markdown_jekyll_data_update_address_tls:
  }
\cs_new:Nn \markdown_jekyll_data_pop:
  {
    \seq_pop_right:NN
      \g_@@_jekyll_data_wildcard_absolute_address_seq
      \l_tmpa_tl
    \seq_pop_right:NN
      \g_@@_jekyll_data_datatypes_seq
      \l_tmpa_tl
    \markdown_jekyll_data_update_address_tls:
  }
```

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.

```
\cs_new:Nn \markdown_jekyll_data_set_keyval:nn
  {
    \keys_set_known:nn
      { markdown/jekyllData }
      { { #1 } = { #2 } }
  }
\cs_generate_variant:Nn
  \markdown_jekyll_data_set_keyval:nn
  { Vn }
\cs_new:Nn \markdown_jekyll_data_set_keyvals:nn
  {
    \markdown_jekyll_data_push:nN
      { #1 }
      \c_@@_jekyll_data_scalar_tl
    \markdown_jekyll_data_set_keyval:Vn
      \g_@@_jekyll_data_wildcard_absolute_address_tl
      { #2 }
    \markdown_jekyll_data_set_keyval:Vn
      \g_@@_jekyll_data_wildcard_relative_address_tl
      { #2 }
    \markdown_jekyll_data_pop:
  }
```

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

```
\def\markdownRendererJekyllDataSequenceBeginPrototype#1#2{
  \markdown_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_sequence_tl
}
\def\markdownRendererJekyllDataMappingBeginPrototype#1#2{
  \markdown_jekyll_data_push:nN
    { #1 }
    \c_@@_jekyll_data_mapping_tl
}
\def\markdownRendererJekyllDataSequenceEndPrototype{
  \markdown_jekyll_data_pop:
}
\def\markdownRendererJekyllDataMappingEndPrototype{
  \markdown_jekyll_data_pop:
}
\def\markdownRendererJekyllDataBooleanPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
```

```
    { #2 }
}
\def\markdownRendererJekyllDataEmptyPrototype#1{}
\def\markdownRendererJekyllDataNumberPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\def\markdownRendererJekyllDataStringPrototype#1#2{
  \markdown_jekyll_data_set_keyvals:nn
    { #1 }
    { #2 }
}
\ExplSyntaxOff
```

3.2.3 Lua Snippets

After the \markdownPrepareLuaOptions macro has been fully expanded, the \markdownLuaOptions macro will expands 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).

```
\ExplSyntaxOn
\tl_new:N \g_@@_formatted_lua_options_tl
\cs_new:Nn \@@_format_lua_options:
  {
    \tl_gclear:N
      \g_@@_formatted_lua_options_tl
    \seq_map_function:NN
      \g_@@_lua_options_seq
      \@@_format_lua_option:n
  }
\cs_new:Nn \@@_format_lua_option:n
  {
    \@@_typecheck_option:n
      { #1 }
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \bool_case_true:nF
      {
        {
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_boolean_tl ||
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_number_tl ||
```

```
8218            \str_if_eq_p:VV
8219              \l_tmpa_tl
8220              \c_@@_option_type_counter_tl
8221          }
8222            {
8223              \@@_get_option_value:nN
8224                { #1 }
8225                \l_tmpa_tl
8226              \tl_gput_right:Nx
8227                \g_@@_formatted_lua_options_tl
8228                { #1~=~  \l_tmpa_tl   ,~ }
8229            }
8230          {
8231            \str_if_eq_p:VV
8232              \l_tmpa_tl
8233              \c_@@_option_type_clist_tl
8234          }
8235            {
8236              \@@_get_option_value:nN
8237                { #1 }
8238                \l_tmpa_tl
8239              \tl_gput_right:Nx
8240                \g_@@_formatted_lua_options_tl
8241                { #1~=~\c_left_brace_str }
8242              \clist_map_inline:Vn
8243                \l_tmpa_tl
8244                {
8245                  \tl_gput_right:Nx
8246                    \g_@@_formatted_lua_options_tl
8247                    { "##1" ,~ }
8248                }
8249              \tl_gput_right:Nx
8250                \g_@@_formatted_lua_options_tl
8251                { \c_right_brace_str ,~ }
8252            }
8253        }
8254        {
8255          \@@_get_option_value:nN
8256            { #1 }
8257            \l_tmpa_tl
8258          \tl_gput_right:Nx
8259            \g_@@_formatted_lua_options_tl
8260            { #1~=~ " \l_tmpa_tl " ,~ }
8261        }
8262    }
8263  \cs_generate_variant:Nn
8264    \clist_map_inline:nn
```

```
  { Vn }
\let\markdownPrepareLuaOptions=\@@_format_lua_options:
\def\markdownLuaOptions{{ \g_@@_formatted_lua_options_tl }}
\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.

```
\def\markdownPrepare{%
```

First, ensure that the `cacheDir` directory exists.

```
  local lfs = require("lfs")
  local cacheDir = "\markdownOptionCacheDir"
  if not lfs.isdir(cacheDir) then
    assert(lfs.mkdir(cacheDir))
  end
```

Next, load the `markdown` module and create a converter function using the plain TEX options, which were serialized to a Lua table via the `\markdownLuaOptions` macro.

```
  local md = require("markdown")
  local convert = md.new(\markdownLuaOptions)
}%
```

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.

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_if_option:nTF
  {
    \@@_get_option_type:nN
      { #1 }
      \l_tmpa_tl
    \str_if_eq:NNF
      \l_tmpa_tl
      \c_@@_option_type_boolean_tl
      {
        \msg_error:nnxx
          { @@ }
          { expected-boolean-option }
          { #1 }
          { \l_tmpa_tl }
      }
    \@@_get_option_value:nN
      { #1 }
```

```
      \l_tmpa_tl
    \str_if_eq:NNTF
      \l_tmpa_tl
      \c_@@_option_value_true_tl
      { #2 }
      { #3 }
  }
\msg_new:nnn
  { @@ }
  { expected-boolean-option }
  {
    Option~#1~has~type~#2,~
    but~a~boolean~was~expected.
  }
\let\markdownIfOption=\@@_if_option:nTF
\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.

```
\csname newread\endcsname\markdownInputFileStream
\csname newwrite\endcsname\markdownOutputFileStream
```

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

```
\begingroup
  \catcode`\^^I=12%
  \gdef\markdownReadAndConvertTab{^^I}%
\endgroup
```

The `\markdownReadAndConvert` macro is largely a rewrite of the LaTeX 2$_\varepsilon$ `\filecontents` macro to plain TeX.

```
\begingroup
```

Make the newline and tab characters active and swap the character codes of the backslash symbol (`\`) and the pipe symbol (`|`), so that we can use the backslash as an ordinary character inside the macro definition. Likewise, swap the character codes of the percent sign (`%`) and the ampersand (`@`), so that we can remove percent signs from the beginning of lines when `stripPercentSigns` is enabled.

```
  \catcode`\^^M=13%
  \catcode`\^^I=13%
  \catcode`|=0%
  \catcode`\\=12%
  |catcode`@=14%
  |catcode`|%=12@
  |gdef|markdownReadAndConvert#1#2{@
    |begingroup@
```

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

```
8328      |markdownIfOption{frozenCache}{}{@
8329        |immediate|openout|markdownOutputFileStream@
8330          |markdownOptionInputTempFileName|relax@
8331        |markdownInfo{Buffering markdown input into the temporary @
8332          input file "|markdownOptionInputTempFileName" and scanning @
8333          for the closing token sequence "#1"}@
8334      }@
```

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.

```
8335      |def|do##1{|catcode`##1=12}|dospecials@
8336      |catcode`| =12@
8337      |markdownMakeOther@
```

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

```
8338      |def|markdownReadAndConvertStripPercentSign##1{@
8339        |markdownIfOption{stripPercentSigns}{@
8340          |if##1%@
8341            |expandafter|expandafter|expandafter@
8342              |markdownReadAndConvertProcessLine@
8343          |else@
8344            |expandafter|expandafter|expandafter@
8345              |markdownReadAndConvertProcessLine@
8346              |expandafter|expandafter|expandafter##1@
8347          |fi@
8348        }{@
8349          |expandafter@
8350            |markdownReadAndConvertProcessLine@
8351            |expandafter##1@
8352        }@
8353      }@
```

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

```
8354      |def|markdownReadAndConvertProcessLine##1#1##2#1##3|relax{@
```

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

```
      |ifx|relax##3|relax@
        |markdownIfOption{frozenCache}{}{@
          |immediate|write|markdownOutputFileStream{##1}@
        }@
      |else@
```

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

```
        |def^^M{@
          |markdownInfo{The ending token sequence was found}@
          |markdownIfOption{frozenCache}{}{@
            |immediate|closeout|markdownOutputFileStream@
          }@
          |endgroup@
          |markdownInput{@
            |markdownOptionOutputDir@
            /|markdownOptionInputTempFileName@
          }@
          #2}@
      |fi@
```

Repeat with the next line.

```
      ^^M}@
```

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

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

```
    |catcode`|^^M=13@
    |def^^M##1^^M{@
      |def^^M####1^^M{@
        |markdownReadAndConvertStripPercentSign####1#1#1|relax}@
      ^^M}@
    ^^M}@
```

Reset the character categories back to the former state.

```
|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.

```
\ExplSyntaxOn
```

```
\int_compare:nT
  { \markdownMode = 3 }
  {
    \markdownInfo{Using~mode~3:~The~lt3luabridge~package}
    \file_input:n { lt3luabridge.tex }
    \cs_new:Npn
      \markdownLuaExecute
      { \luabridgeExecute }
  }
\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 LuaTEX engine, their TEX distribution contains it, and uses shell access to produce and execute Lua scripts using the TEXLua interpreter [1, Section 4.1.1].

```
\ifnum\markdownMode<2\relax
\ifnum\markdownMode=0\relax
  \markdownWarning{Using mode 0: Shell escape via write18
                   (deprecated, to be removed in Markdown 3.0.0)}%
\else
  \markdownWarning{Using mode 1: Shell escape via os.execute
                   (deprecated, to be removed in Markdown 3.0.0)}%
\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 `\pdfshellescape` (LuaTEX, PdfTEX) or the `\shellescape` (XƎTEX) 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.

```
\ifx\pdfshellescape\undefined
  \ifx\shellescape\undefined
    \ifnum\markdownMode=0\relax
      \def\markdownExecuteShellEscape{1}%
    \else
      \def\markdownExecuteShellEscape{%
        \directlua{tex.sprint(status.shell_escape or "1")}}%
    \fi
  \else
    \let\markdownExecuteShellEscape\shellescape
```

```
  \fi
\else
  \let\markdownExecuteShellEscape\pdfshellescape
\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.

```
\ifnum\markdownMode=0\relax
  \def\markdownExecuteDirect#1{\immediate\write18{#1}}%
\else
  \def\markdownExecuteDirect#1{%
    \directlua{os.execute("\luaescapestring{#1}")}}%
\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.

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

```
\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.

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

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

```
    |immediate|openout|markdownOutputFileStream=%
      |markdownOptionHelperScriptFileName
    |markdownInfo{Writing a helper Lua script to the file
      "|markdownOptionHelperScriptFileName"}%
    |immediate|write|markdownOutputFileStream{%
      local ran_ok, error = pcall(function()
        local ran_ok, kpse = pcall(require, "kpse")
```

```
        if ran_ok then kpse.set_program_name("luatex") end
        #1
      end)
```

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

```
      if not ran_ok then
        local file = io.open("%
          |markdownOptionOutputDir
          /|markdownOptionErrorTempFileName", "w")
        if file then
          file:write(error .. "\n")
          file:close()
        end
        print('\\markdownError{An error was encountered while executing
               Lua code}{For further clues, examine the file
               "|markdownOptionOutputDir
               /|markdownOptionErrorTempFileName"}')
      end}%
    |immediate|closeout|markdownOutputFileStream
```

Execute the generated `helperScriptFileName` Lua script using the TEXLua binary and store the output in the `outputTempFileName` file.

```
    |markdownInfo{Executing a helper Lua script from the file
      "|markdownOptionHelperScriptFileName" and storing the result in the
      file "|markdownOptionOutputTempFileName"}%
    |markdownExecute{texlua "|markdownOptionOutputDir
      /|markdownOptionHelperScriptFileName" > %
      "|markdownOptionOutputDir
      /|markdownOptionOutputTempFileName"}%
```

`\input` the generated `outputTempFileName` file.

```
    |input|markdownOptionOutputTempFileName|relax}%
|endgroup
```

3.2.6 Direct Lua Access

The following TEX code is intended for TEX engines that provide direct access to Lua (LuaTEX). 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.

```
\fi
\ifnum\markdownMode=2\relax
  \markdownWarning{Using mode 2: Direct Lua access
                   (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,

```
\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.

```
  \catcode`|=0%
  \catcode`\\=12%
  |gdef|markdownLuaExecute#1{%
    |directlua{%
      local function print(input)
        local output = {}
        for line in input:gmatch("[^\r\n]+") do
          table.insert(output, line)
        end
        tex.print(output)
      end
      #1
    }%
  }%
|endgroup
\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.

```
\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.

```
  \catcode`|=0%
  \catcode`\\=12%
  \catcode`|&=6%
  |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.

```
    |begingroup
    |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.

```
    |catcode`|#=12
```

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

```
    |markdownIfOption{frozenCache}{%
      |ifnum|markdownOptionFrozenCacheCounter=0|relax
        |markdownInfo{Reading frozen cache from
          "|markdownOptionFrozenCacheFileName"}%
        |input|markdownOptionFrozenCacheFileName|relax
      |fi
      |markdownInfo{Including markdown document number
        "|the|markdownOptionFrozenCacheCounter" from frozen cache}%
      |csname markdownFrozenCache|the|markdownOptionFrozenCacheCounter|endcsname
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }{%
      |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 LaTeXMk to track changes to the markdown document.

```
      |openin|markdownInputFileStream&1
      |closein|markdownInputFileStream
      |markdownPrepareLuaOptions
      |markdownLuaExecute{%
        |markdownPrepare
        local file = assert(io.open("&1", "r"),
          [[Could not open file "&1" for reading]])
        local input = assert(file:read("*a"))
        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.

```
        print(convert(input))}%
```

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

```
      |global|advance|markdownOptionFrozenCacheCounter by 1|relax
    }%
    |endgroup
  }%
|endgroup
```

The `\markdownEscape` macro resets the category codes of the percent sign and the hash sign back to comment and parameter, respectively, before using the `\input` built-in of TeX to execute a TeX document in the middle of a markdown document fragment.

```
\gdef\markdownEscape#1{%
  \catcode`\%=14\relax
  \catcode`\#=6\relax
```

```
  \input #1\relax
  \catcode`\%=12\relax
  \catcode`\#=12\relax
}%
```

3.3 LaTeX Implementation

The LaTeX implemenation makes use of the fact that, apart from some subtle differences, LaTeX implements the majority of the plain TeX format [12, Section 9]. As a consequence, we can directly reuse the existing plain TeX implementation.

```
\def\markdownVersionSpace{ }%
\ProvidesPackage{markdown}[\markdownLastModified\markdownVersionSpace v%
  \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.

```
\ExplSyntaxOn
\@@_latex_define_renderers:
\@@_latex_define_renderer_prototypes:
\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).

```
\let\markdownInputPlainTeX\markdownInput
\renewcommand\markdownInput[2][]{%
  \begingroup
    \markdownSetup{#1}%
    \markdownInputPlainTeX{#2}%
  \endgroup}%
```

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

```
\renewenvironment{markdown}{%
  \markdownReadAndConvert@markdown{}}{%
  \markdownEnd}%
\renewenvironment{markdown*}[1]{%
  \markdownSetup{#1}%
```

```
  \markdownReadAndConvert@markdown*}{%
  \markdownEnd}%
\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*.

```
  \catcode`\|=0\catcode`\<=1\catcode`\>=2%
  \catcode`\\=12|catcode`|{=12|catcode`|}=12%
  |gdef|markdownReadAndConvert@markdown#1<%
    |markdownReadAndConvert<\end{markdown#1}>%
                           <|end<markdown#1>>>%
|endgroup
```

3.3.2.1 LaTeX Themes

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

```
\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.

```
\newcommand\markdownLaTeXThemeName{}
\seq_new:N \g_@@_latex_themes_seq
\seq_gput_right:NV
  \g_@@_latex_themes_seq
  \markdownLaTeXThemeName
\newcommand\markdownLaTeXThemeLoad[2]{
  \def\@tempa{%
    \def\markdownLaTeXThemeName{#2}
    \seq_gput_right:NV
      \g_@@_latex_themes_seq
      \markdownLaTeXThemeName
    \RequirePackage{#1}
    \seq_pop_right:NN
      \g_@@_latex_themes_seq
      \l_tmpa_tl
    \seq_get_right:NN
      \g_@@_latex_themes_seq
      \l_tmpa_tl
    \exp_args:NNV
      \def
      \markdownLaTeXThemeName
      \l_tmpa_tl}
  \ifmarkdownLaTeXLoaded
    \@tempa
  \else
    \exp_args:No
```

```
      \AtEndOfPackage
      { \@tempa }
  \fi}
\ExplSyntaxOff
```

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

```
\markdownSetup{fencedCode}%
```

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

```
\RequirePackage{ifthen,grffile}
```

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

```
\let\markdown@witiko@dot@oldRendererInputFencedCodePrototype
  \markdownRendererInputFencedCodePrototype
```

If the infostring starts with `dot …`, we redefine the fenced code block token renderer prototype, so that it typesets the code block via Graphviz tools if and only if the `frozenCache` plain TEX option is disabled and the code block has not been previously typeset:

```
\renewcommand\markdownRendererInputFencedCodePrototype[2]{%
  \def\next##1 ##2\relax{%
    \ifthenelse{\equal{##1}{dot}}{%
      \markdownIfOption{frozenCache}{}{%
        \immediate\write18{%
          if ! test -e #1.pdf.source || ! diff #1 #1.pdf.source;
          then
            dot -Tpdf -o #1.pdf #1;
            cp #1 #1.pdf.source;
          fi}}%
```

We include the typeset image using the image token renderer:

```
      \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:

```
    }{%
      \markdown@witiko@dot@oldRendererInputFencedCodePrototype{#1}{#2}%
    }%
  }%
  \next#2 \relax}%
```

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

```
\let\markdown@witiko@graphicx@http@oldRendererImagePrototype
  \markdownRendererImagePrototype
```

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

```
\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.

```
\newcount\markdown@witiko@graphicx@http@counter
\markdown@witiko@graphicx@http@counter=0
\newcommand\markdown@witiko@graphicx@http@filename{%
  \markdownOptionCacheDir/witiko_graphicx_http%
  .\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 downloads the online image to the pathname.

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

```
\begingroup
\catcode`\%=12
\catcode`\^^A=14
```

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

```
\global\def\markdownRendererImagePrototype#1#2#3#4{^^A
  \begingroup
    \edef\filename{\markdown@witiko@graphicx@http@filename}^^A
```

The image will be downloaded only if the image URL has the http or https protocols and the frozenCache plain TEX option is disabled:

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

The image will be downloaded to the pathname cacheDir/⟨*the MD5 digest of the image URL*⟩.⟨*the suffix of the image URL*⟩:

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

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

```
          if ! [ -e "$OUTPUT" ];
          then
            \markdown@witiko@graphicx@http@download{'#3'}{"$OUTPUT"};
```

```
            printf '%s' "$OUTPUT" > "\filename";
          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:

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

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

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

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

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

3.3.3 Options

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

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

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

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/latex-options }
  {
    renderers .code:n = {
      \keys_set:nn
        { markdown/latex-options/renderers }
        { #1 }
    },
  }
\@@_with_various_cases:nn
  { rendererPrototypes }
  {
    \keys_define:nn
      { markdown/latex-options }
      {
        #1 .code:n = {
```

```
          \keys_set:nn
            { markdown/latex-options/renderer-prototypes }
            { ##1 }
        },
      }
  }
```

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

```
\keys_define:nn
  { markdown/latex-options }
  {
    code .code:n = { #1 },
  }
```

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

```
\@@_with_various_cases:nn
  { jekyllDataRenderers }
  {
    \keys_define:nn
      { markdown/latex-options }
      {
        #1 .code:n = {
          \tl_set:Nn
            \l_tmpa_tl
            { ##1 }
```

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

```
          \tl_replace_all:NnV
            \l_tmpa_tl
            { / }
            \c_backslash_str
          \keys_set:nV
            { markdown/latex-options/jekyll-data-renderers }
            \l_tmpa_tl
        },
      }
  }
\keys_define:nn
  { markdown/latex-options/jekyll-data-renderers }
  {
```

```
    unknown .code:n = {
      \tl_set_eq:NN
        \l_tmpa_tl
        \l_keys_key_str
      \tl_replace_all:NVn
        \l_tmpa_tl
        \c_backslash_str
        { / }
      \tl_put_right:Nn
        \l_tmpa_tl
        {
          .code:n = { #1 }
        }
      \keys_define:nV
        { markdown/jekyllData }
        \l_tmpa_tl
    }
  }
\cs_generate_variant:Nn
  \keys_define:nn
  { nV }
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NVn }
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NnV }
\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.

```
\markdownIfOption{plain}{\iffalse}{\iftrue}
```

If either the `tightLists` or the `fancyLists` Lua option is enabled and the current document class is not beamer, then load the paralist package.

```
\@ifclassloaded{beamer}{}{%
  \markdownIfOption{tightLists}{\RequirePackage{paralist}}{}%
  \markdownIfOption{fancyLists}{\RequirePackage{paralist}}{}%
}
```

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.

```
\ExplSyntaxOn
\@ifpackageloaded{paralist}{
```

```
\tl_new:N
  \l_@@_latex_fancy_list_item_label_number_style_tl
\tl_new:N
  \l_@@_latex_fancy_list_item_label_delimiter_style_tl
\cs_new:Nn
  \@@_latex_fancy_list_item_label_number:nn
  {
    \str_case:nn
      { #1 }
      {
        { Decimal } { #2 }
        { LowerRoman } { \int_to_roman:n { #2 } }
        { UpperRoman } { \int_to_Roman:n { #2 } }
        { LowerAlpha } { \int_to_alph:n { #2 } }
        { UpperAlpha } { \int_to_alph:n { #2 } }
      }
  }
\cs_new:Nn
  \@@_latex_fancy_list_item_label_delimiter:n
  {
    \str_case:nn
      { #1 }
      {
        { Default } { . }
        { OneParen } { ) }
        { Period } { . }
      }
  }
\cs_new:Nn
  \@@_latex_fancy_list_item_label:nnn
  {
    \@@_latex_fancy_list_item_label_number:nn
      { #1 }
      { #3 }
    \@@_latex_fancy_list_item_label_delimiter:n
      { #2 }
  }
\cs_new:Nn
  \@@_latex_paralist_style:nn
  {
    \str_case:nn
      { #1 }
      {
        { Decimal } { 1 }
        { LowerRoman } { i }
        { UpperRoman } { I }
        { LowerAlpha } { a }
```

```
          { UpperAlpha } { A }
        }
      \@@_latex_fancy_list_item_label_delimiter:n
        { #2 }
    }
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\begin{compactitem}},
    ulEndTight = {\end{compactitem}},
    fancyOlBegin = {
      \group_begin:
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_number_style_tl
        { #1 }
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_delimiter_style_tl
        { #2 }
      \tl_set:Nn
        \l_tmpa_tl
        { \begin{enumerate}[ }
      \tl_put_right:Nx
        \l_tmpa_tl
        { \@@_latex_paralist_style:nn { #1 } { #2 } }
      \tl_put_right:Nn
        \l_tmpa_tl
        { ] }
      \l_tmpa_tl
    },
    fancyOlEnd = {
      \end{enumerate}
      \group_end:
    },
    olBeginTight = {\begin{compactenum}},
    olEndTight = {\end{compactenum}},
    fancyOlBeginTight = {
      \group_begin:
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_number_style_tl
        { #1 }
      \tl_set:Nn
        \l_@@_latex_fancy_list_item_label_delimiter_style_tl
        { #2 }
      \tl_set:Nn
        \l_tmpa_tl
        { \begin{compactenum}[ }
      \tl_put_right:Nx
        \l_tmpa_tl
        { \@@_latex_paralist_style:nn { #1 } { #2 } }
```

```
      \tl_put_right:Nn
        \l_tmpa_tl
        { ] }
      \l_tmpa_tl
    },
    fancyOlEndTight = {
      \end{compactenum}
      \group_end:
    },
    fancyOlItemWithNumber = {
      \item
        [
          \@@_latex_fancy_list_item_label:VVn
            \l_@@_latex_fancy_list_item_label_number_style_tl
            \l_@@_latex_fancy_list_item_label_delimiter_style_tl
            { #1 }
        ]
    },
    dlBeginTight = {\begin{compactdesc}},
    dlEndTight = {\end{compactdesc}}}}
  \cs_generate_variant:Nn
    \@@_latex_fancy_list_item_label:nnn
    { VVn }
}{
  \markdownSetup{rendererPrototypes={
    ulBeginTight = {\markdownRendererUlBegin},
    ulEndTight = {\markdownRendererUlEnd},
    fancyOlBegin = {\markdownRendererOlBegin},
    fancyOlEnd = {\markdownRendererOlEnd},
    olBeginTight = {\markdownRendererOlBegin},
    olEndTight = {\markdownRendererOlEnd},
    fancyOlBeginTight = {\markdownRendererOlBegin},
    fancyOlEndTight = {\markdownRendererOlEnd},
    dlBeginTight = {\markdownRendererDlBegin},
    dlEndTight = {\markdownRendererDlEnd}}}
}
\ExplSyntaxOff
\RequirePackage{amsmath}
```

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

```
\@ifpackageloaded{unicode-math}{
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\mdlgwhtsquare$},
  }}
}{
  \RequirePackage{amssymb}
```

```
  \markdownSetup{rendererPrototypes={
    untickedBox = {$\square$},
  }}
}
\RequirePackage{csvsimple}
\RequirePackage{fancyvrb}
\RequirePackage{graphicx}
\markdownSetup{rendererPrototypes={
  lineBreak = {\\},
  leftBrace = {\textbraceleft},
  rightBrace = {\textbraceright},
  dollarSign = {\textdollar},
  underscore = {\textunderscore},
  circumflex = {\textasciicircum},
  backslash = {\textbackslash},
  tilde = {\textasciitilde},
  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.

```
  codeSpan = {%
    \ifmmode
      \text{#1}%
    \else
      \texttt{#1}%
    \fi
  }}}
\ExplSyntaxOn
\markdownSetup{
  rendererPrototypes = {
    contentBlock = {
      \str_case:nnF
        { #1 }
        {
          { csv }
            {
              \begin{table}
                \begin{center}
                  \csvautotabular{#3}
```

[8] This property may actually be undecidable. Suppose a span of text is a part of a macro definition. Then, whether the span of text is part of a math formula or not depends on where the macro is later used, which may easily be *both* inside and outside a math formula.

```
                \end{center}
                \tl_if_empty:nF
                  { #4 }
                  { \caption{#4} }
              \end{table}
            }
          { tex } { \markdownEscape{#3} }
        }
        { \markdownInput{#3} }
    },
  },
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={
  image = {%
    \begin{figure}%
      \begin{center}%
        \includegraphics{#3}%
      \end{center}%
      \ifx\empty#4\empty\else
        \caption{#4}%
      \fi
    \end{figure}},
  ulBegin = {\begin{itemize}},
  ulEnd = {\end{itemize}},
  olBegin = {\begin{enumerate}},
  olItem = {\item{}},
  olItemWithNumber = {\item[#1.]},
  olEnd = {\end{enumerate}},
  dlBegin = {\begin{description}},
  dlItem = {\item[#1]},
  dlEnd = {\end{description}},
  emphasis = {\emph{#1}},
  tickedBox = {$\boxtimes$},
  halfTickedBox = {$\boxdot$},
```

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

```
  headerAttributeContextBegin = {%
    \markdownSetup{
      rendererPrototypes = {
        attributeIdentifier = {%
          \begingroup
          \def\next####1{%
            \def####1#########1{%
              \endgroup
              ####1{#########1}%
```

```
              \label{##1}%
            }%
          }%
          \next\markdownRendererHeadingOne
          \next\markdownRendererHeadingTwo
          \next\markdownRendererHeadingThree
          \next\markdownRendererHeadingFour
          \next\markdownRendererHeadingFive
          \next\markdownRendererHeadingSix
        },
      },
    }%
  },
  headerAttributeContextEnd = {},
  superscript = {\textsuperscript{#1}},
  subscript = {\textsubscript{#1}},
  blockQuoteBegin = {\begin{quotation}},
  blockQuoteEnd = {\end{quotation}},
  inputVerbatim = {\VerbatimInput{#1}},
  thematicBreak = {\noindent\rule[0.5ex]{\linewidth}{1pt}},
  note = {\footnote{#1}}}}
```

3.3.4.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\RequirePackage{ltxcmds}
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputFencedCodePrototype#1#2
  {
    \tl_if_empty:nTF
      { #2 }
      { \markdownRendererInputVerbatim{#1} }
```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written.

```
      {
        \regex_extract_once:nnN
          { \w* }
          { #2 }
          \l_tmpa_seq
        \seq_pop_left:NN
          \l_tmpa_seq
          \l_tmpa_tl
```

When the minted package is loaded, use it for syntax highlighting.

```
        \ltx@ifpackageloaded
          { minted }
```

```
          {
            \catcode`\#=6\relax
            \exp_args:NV
              \inputminted
              \l_tmpa_tl
              { #1 }
            \catcode`\#=12\relax
          }
          {
```

When the listings package is loaded, use it for syntax highlighting.

```
            \ltx@ifpackageloaded
              { listings }
              { \lstinputlisting[language=\l_tmpa_tl]{#1} }
```

When neither the listings package nor the minted package is loaded, act as though no infostring were given.

```
              { \markdownRendererInputFencedCode{#1}{} }
          }
      }
  }
\ExplSyntaxOff
```

Support the nesting of strong emphasis.

```
\ExplSyntaxOn
\def\markdownLATEXStrongEmphasis#1{%
  \str_if_in:NnTF
    \f@series
    { b }
    { \textnormal{#1} }
    { \textbf{#1} }
}
\ExplSyntaxOff
\markdownSetup{rendererPrototypes={strongEmphasis={%
  \protect\markdownLATEXStrongEmphasis{#1}}}}
```

Support LaTeX document classes that do not provide chapters.

```
\@ifundefined{chapter}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\section{#1}},
    headingTwo = {\subsection{#1}},
    headingThree = {\subsubsection{#1}},
    headingFour = {\paragraph{#1}\leavevmode},
    headingFive = {\subparagraph{#1}\leavevmode}}}
}{%
  \markdownSetup{rendererPrototypes = {
    headingOne = {\chapter{#1}},
    headingTwo = {\section{#1}},
    headingThree = {\subsection{#1}},
```

```
    headingFour = {\subsubsection{#1}},
    headingFive = {\paragraph{#1}\leavevmode},
    headingSix = {\subparagraph{#1}\leavevmode}}}
}%
```

3.3.4.2 Tickboxes

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

```
\markdownSetup{
  rendererPrototypes = {
    ulItem = {%
      \futurelet\markdownLaTeXCheckbox\markdownLaTeXUlItem
    },
  },
}
\def\markdownLaTeXUlItem{%
  \ifx\markdownLaTeXCheckbox\markdownRendererTickedBox
    \item[\markdownLaTeXCheckbox]%
    \expandafter\@gobble
  \else
    \ifx\markdownLaTeXCheckbox\markdownRendererHalfTickedBox
      \item[\markdownLaTeXCheckbox]%
      \expandafter\expandafter\expandafter\@gobble
    \else
      \ifx\markdownLaTeXCheckbox\markdownRendererUntickedBox
        \item[\markdownLaTeXCheckbox]%
        \expandafter\expandafter\expandafter\expandafter
          \expandafter\expandafter\expandafter\@gobble
      \else
        \item{}%
      \fi
    \fi
  \fi
}
```

3.3.4.3 HTML elements

If the `html` option is enabled and we are using TEX4ht[9], we will pass HTML elements to the output HTML document unchanged.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
        \ifvmode
          \IgnorePar
          \EndP
        \fi
```

[9] See https://tug.org/tex4ht/.

```
        \HCode{#1}%
      },
      inputBlockHtmlElement = {%
        \ifvmode
          \IgnorePar
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
  }
}
```

3.3.4.4 Citations

Here is a basic implementation for citations that uses the LaTeX `\cite` macro. There are also implementations that use the natbib `\citep`, and `\citet` macros, and the BibLaTeX `\autocites` and `\textcites` macros. These implementations will be used, when the respective packages are loaded.

```
\newcount\markdownLaTeXCitationsCounter

% Basic implementation
\RequirePackage{gobble}
\def\markdownLaTeXBasicCitations#1#2#3#4#5#6{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#4\relax
    \ifx\relax#5\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \cite{#1#2#6}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobblethree
      \fi
    \else%  Before a postnote (#5), dump the accumulator
      \ifx\relax#1\relax\else
        \cite{#1}%
      \fi
      \cite[#5]{#6}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
      \fi
```

```
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobblethree
    \fi
  \else%  Before a prenote (#4), dump the accumulator
    \ifx\relax#1\relax\else
      \cite{#1}%
    \fi
    \ifnum\markdownLaTeXCitationsCounter>1\relax
      \space  % Insert a space before the prenote in later citations
    \fi
    #4~\expandafter\cite\ifx\relax#5\relax{#6}\else[#5]{#6}\fi
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXBasicCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobblethree
  \fi\markdownLaTeXBasicCitations{#1#2#6},}
\let\markdownLaTeXBasicTextCitations\markdownLaTeXBasicCitations

% Natbib implementation
\def\markdownLaTeXNatbibCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \citep{#1,#5}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  Before a postnote (#4), dump the accumulator
      \ifx\relax#1\relax\else
```

```
        \citep{#1}%
      \fi
      \citep[][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
      \else
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  Before a prenote (#3), dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citep{#1}%
    \fi
    \citep[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \else
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibCitations
    \fi
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibCitations{#1,#5}}
\def\markdownLaTeXNatbibTextCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifx\relax#3\relax
    \ifx\relax#4\relax
      \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
        \citet{#1,#5}%  Without prenotes and postnotes, just accumulate cites
        \expandafter\expandafter\expandafter
        \expandafter\expandafter\expandafter\expandafter
        \@gobbletwo
      \fi
    \else%  After a prenote or a postnote, dump the accumulator
      \ifx\relax#1\relax\else
        \citet{#1}%
```

```
      \fi
      , \citet[#3][#4]{#5}%
      \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
        ,
      \else
        \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
          ,
        \fi
      \fi
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter
      \markdownLaTeXNatbibTextCitations
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter{%
      \expandafter\expandafter\expandafter
      \expandafter\expandafter\expandafter\expandafter}%
      \expandafter\expandafter\expandafter
      \@gobbletwo
    \fi
  \else%  After a prenote or a postnote, dump the accumulator
    \ifx\relax#1\relax\relax\else
      \citet{#1}%
    \fi
    , \citet[#3][#4]{#5}%
    \ifnum\markdownLaTeXCitationsCounter<\markdownLaTeXCitationsTotal\relax
      ,
    \else
      \ifnum\markdownLaTeXCitationsCounter=\markdownLaTeXCitationsTotal\relax
        ,
      \fi
    \fi
    \expandafter\expandafter\expandafter
    \markdownLaTeXNatbibTextCitations
    \expandafter\expandafter\expandafter{%
    \expandafter\expandafter\expandafter}%
    \expandafter
    \@gobbletwo
  \fi\markdownLaTeXNatbibTextCitations{#1,#5}}

% BibLaTeX implementation
\def\markdownLaTeXBibLaTeXCitations#1#2#3#4#5{%
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \autocites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXCitations{#1[#3][#4]{#5}}}
\def\markdownLaTeXBibLaTeXTextCitations#1#2#3#4#5{%
```

```
  \advance\markdownLaTeXCitationsCounter by 1\relax
  \ifnum\markdownLaTeXCitationsCounter>\markdownLaTeXCitationsTotal\relax
    \textcites#1[#3][#4]{#5}%
    \expandafter\@gobbletwo
  \fi\markdownLaTeXBibLaTeXTextCitations{#1[#3][#4]{#5}}}

\markdownSetup{rendererPrototypes = {
  cite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXCitations
      \expandafter{\expandafter}%
    }},
  textCite = {%
    \markdownLaTeXCitationsCounter=1%
    \def\markdownLaTeXCitationsTotal{#1}%
    \@ifundefined{autocites}{%
      \@ifundefined{citep}{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXBasicTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }{%
        \expandafter\expandafter\expandafter
        \markdownLaTeXNatbibTextCitations
        \expandafter\expandafter\expandafter{%
        \expandafter\expandafter\expandafter}%
      }%
    }{%
```

```
      \expandafter\expandafter\expandafter
      \markdownLaTeXBibLaTeXTextCitations
      \expandafter{\expandafter}%
    }}}}
```

3.3.4.5 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.

```
\RequirePackage{url}
\RequirePackage{expl3}
\ExplSyntaxOn
\def\markdownRendererLinkPrototype#1#2#3#4{
  \tl_set:Nn \l_tmpa_tl { #1 }
  \tl_set:Nn \l_tmpb_tl { #2 }
  \bool_set:Nn
    \l_tmpa_bool
    {
      \tl_if_eq_p:NN
        \l_tmpa_tl
        \l_tmpb_tl
    }
  \tl_set:Nn \l_tmpa_tl { #4 }
  \bool_set:Nn
    \l_tmpb_bool
    {
      \tl_if_empty_p:N
        \l_tmpa_tl
    }
```

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.

```
  \bool_if:nTF
    {
      \l_tmpa_bool && \l_tmpb_bool
    }
    {
      \markdownLaTeXRendererAutolink { #2 } { #3 }
    }{
      \markdownLaTeXRendererDirectOrIndirectLink { #1 } { #2 } { #3 } { #4 }
    }
}
\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.

```
  \tl_set:Nn
```

```
    \l_tmpa_tl
    { #2 }
  \tl_trim_spaces:N
    \l_tmpa_tl
  \tl_set:Nx
    \l_tmpb_tl
    {
      \tl_range:Nnn
        \l_tmpa_tl
        { 1 }
        { 1 }
    }
  \str_if_eq:NNTF
    \l_tmpb_tl
    \c_hash_str
    {
      \tl_set:Nx
        \l_tmpb_tl
        {
          \tl_range:Nnn
            \l_tmpa_tl
            { 2 }
            { -1 }
        }
      \exp_args:NV
        \ref
        \l_tmpb_tl
    }{
      \url { #2 }
    }
}
\ExplSyntaxOff
\def\markdownLaTeXRendererDirectOrIndirectLink#1#2#3#4{%
  #1\footnote{\ifx\empty#4\empty\else#4: \fi\url{#3}}}
```

3.3.4.6 Tables

Here is a basic implementation of tables. If the booktabs package is loaded, then it is used to produce horizontal lines.

```
\newcount\markdownLaTeXRowCounter
\newcount\markdownLaTeXRowTotal
\newcount\markdownLaTeXColumnCounter
\newcount\markdownLaTeXColumnTotal
\newtoks\markdownLaTeXTable
\newtoks\markdownLaTeXTableAlignment
\newtoks\markdownLaTeXTableEnd
\AtBeginDocument{%
  \@ifpackageloaded{booktabs}{%
```

```
    \def\markdownLaTeXTopRule{\toprule}%
    \def\markdownLaTeXMidRule{\midrule}%
    \def\markdownLaTeXBottomRule{\bottomrule}%
  }{%
    \def\markdownLaTeXTopRule{\hline}%
    \def\markdownLaTeXMidRule{\hline}%
    \def\markdownLaTeXBottomRule{\hline}%
  }%
}
\markdownSetup{rendererPrototypes={
  table = {%
    \markdownLaTeXTable={}%
    \markdownLaTeXTableAlignment={}%
    \markdownLaTeXTableEnd={%
      \markdownLaTeXBottomRule
      \end{tabular}}%
    \ifx\empty#1\empty\else
      \addto@hook\markdownLaTeXTable{%
        \begin{table}
        \centering}%
      \addto@hook\markdownLaTeXTableEnd{%
        \caption{#1}
        \end{table}}%
    \fi
    \addto@hook\markdownLaTeXTable{\begin{tabular}}%
    \markdownLaTeXRowCounter=0%
    \markdownLaTeXRowTotal=#2%
    \markdownLaTeXColumnTotal=#3%
    \markdownLaTeXRenderTableRow
  }
}}
\def\markdownLaTeXRenderTableRow#1{%
  \markdownLaTeXColumnCounter=0%
  \ifnum\markdownLaTeXRowCounter=0\relax
    \markdownLaTeXReadAlignments#1%
    \markdownLaTeXTable=\expandafter\expandafter\expandafter{%
      \expandafter\the\expandafter\markdownLaTeXTable\expandafter{%
        \the\markdownLaTeXTableAlignment}}%
    \addto@hook\markdownLaTeXTable{\markdownLaTeXTopRule}%
  \else
    \markdownLaTeXRenderTableCell#1%
  \fi
  \ifnum\markdownLaTeXRowCounter=1\relax
    \addto@hook\markdownLaTeXTable\markdownLaTeXMidRule
  \fi
  \advance\markdownLaTeXRowCounter by 1\relax
  \ifnum\markdownLaTeXRowCounter>\markdownLaTeXRowTotal\relax
```

```
    \the\markdownLaTeXTable
    \the\markdownLaTeXTableEnd
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableRow}
\def\markdownLaTeXReadAlignments#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \if#1d%
    \addto@hook\markdownLaTeXTableAlignment{l}%
  \else
    \addto@hook\markdownLaTeXTableAlignment{#1}%
  \fi
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax\else
    \expandafter\@gobble
  \fi\markdownLaTeXReadAlignments}
\def\markdownLaTeXRenderTableCell#1{%
  \advance\markdownLaTeXColumnCounter by 1\relax
  \ifnum\markdownLaTeXColumnCounter<\markdownLaTeXColumnTotal\relax
    \addto@hook\markdownLaTeXTable{#1&}%
  \else
    \addto@hook\markdownLaTeXTable{#1\\}%
    \expandafter\@gobble
  \fi\markdownLaTeXRenderTableCell}
```

3.3.4.7 Line Blocks

Here is a basic implementation of line blocks. If the verse package is loaded, then it is used to produce the verses.

```

\markdownIfOption{lineBlocks}{%
  \RequirePackage{verse}
  \markdownSetup{rendererPrototypes={
    lineBlockBegin = {%
      \begingroup
        \def\markdownRendererLineBreak{\\}%
        \begin{verse}%
    },
    lineBlockEnd = {%
        \end{verse}%
      \endgroup
    },
  }}
}{}

```

3.3.4.8 YAML Metadata

The default setup of YAML metadata will invoke the `\title`, `\author`, and `\date` macros when scalar values for keys that correspond to the `title`, `author`, and `date` relative wildcards are encountered, respectively.

```
\ExplSyntaxOn
\keys_define:nn
  { markdown/jekyllData }
  {
    author  .code:n = { \author{#1} },
    date    .code:n = { \date{#1}   },
    title   .code:n = { \title{#1}  },
  }
```

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.

```
% TODO: Remove the command definition in TeX Live 2021.
\providecommand\IfFormatAtLeastTF{\@ifl@t@r\fmtversion}
\markdownSetup{
  rendererPrototypes = {
    jekyllDataEnd = {
%     TODO: Remove the else branch in TeX Live 2021.
      \IfFormatAtLeastTF
        { 2020-10-01 }
        { \AddToHook{begindocument/end}{\maketitle} }
        {
          \ifx\@onlypreamble\@notprerr
            % We are in the document
            \maketitle
          \else
            % We are in the preamble
            \RequirePackage{etoolbox}
            \AfterEndPreamble{\maketitle}
          \fi
        }
    },
  },
}
\ExplSyntaxOff
```

3.3.4.9 Strike-Through

If the `strikeThrough` option is enabled, we will load the soulutf8 package and use it to implement strike-throughs.

```
\markdownIfOption{strikeThrough}{%
  \RequirePackage{soulutf8}%
  \markdownSetup{
    rendererPrototypes = {
      strikeThrough = {%
        \st{#1}%
      },
```

```
    }
  }
}{}
```

3.3.4.10 Raw Attribute Renderer Prototypes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
```

```
\ExplSyntaxOff
\fi % Closes `\markdownIfOption{Plain}{\iffalse}{iftrue}`
```

3.3.5 Miscellanea

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the inputenc package. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents package.

```
\newcommand\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \ifnum\count0<256\repeat}%
```

3.4 ConTEXt Implementation

The ConTEXt implementation makes use of the fact that, apart from some subtle differences, the Mark II and Mark IV ConTEXt formats *seem* to implement (the documentation is scarce) the majority of the plain TEX format required by the plain TEX implementation. As a consequence, we can directly reuse the existing plain TEX implementation after supplying the missing plain TEX macros.

When buffering user input, we should disable the bytes with the high bit set, since these are made active by the `\enableregime` macro. We will do this by redefining the `\markdownMakeOther` macro accordingly. The code is courtesy of Scott Pakin, the creator of the filecontents LATEX package.

```
\def\markdownMakeOther{%
  \count0=128\relax
  \loop
    \catcode\count0=11\relax
    \advance\count0 by 1\relax
  \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.

```
  \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).

```
\long\def\inputmarkdown{%
  \dosingleempty
  \doinputmarkdown}%
```

```
\long\def\doinputmarkdown[#1]#2{%
  \begingroup
    \iffirstargument
      \setupmarkdown{#1}%
    \fi
    \markdownInput{#2}%
  \endgroup}%
```

The \startmarkdown and \stopmarkdown macros are implemented using the \markdownReadAndConvert macro.

In Knuth's TEX, trailing spaces are removed very early on when a line is being put to the input buffer. [13, sec. 31]. According to Eijkhout [14, sec. 2.2], this is because "these spaces are hard to see in an editor". At the moment, there is no option to suppress this behavior in (Lua)TEX, but ConTEXt MkIV funnels all input through its own input handler. This makes it possible to suppress the removal of trailing spaces in ConTEXt MkIV and therefore to insert hard line breaks into markdown text.

```
\ifx\startluacode\undefined  % MkII
  \begingroup
    \catcode`\|=0%
    \catcode`\\=12%
    |gdef|startmarkdown{%
      |markdownReadAndConvert{\stopmarkdown}%
                             {|stopmarkdown}}%
    |gdef|stopmarkdown{%
      |markdownEnd}%
  |endgroup
\else  % MkIV
  \startluacode
    document.markdown_buffering = false
    local function preserve_trailing_spaces(line)
      if document.markdown_buffering then
        line = line:gsub("[ \t][ \t]$", "\t\t")
      end
      return line
    end
    resolvers.installinputlinehandler(preserve_trailing_spaces)
  \stopluacode
  \begingroup
    \catcode`\|=0%
    \catcode`\\=12%
    |gdef|startmarkdown{%
      |ctxlua{document.markdown_buffering = true}%
      |markdownReadAndConvert{\stopmarkdown}%
                             {|stopmarkdown}}%
    |gdef|stopmarkdown{%
      |ctxlua{document.markdown_buffering = false}%
      |markdownEnd}%
```

```
  |endgroup
\fi
```

3.4.2 Token Renderer Prototypes

The following configuration should be considered placeholder.

```
\def\markdownRendererLineBreakPrototype{\blank}%
\def\markdownRendererLeftBracePrototype{\textbraceleft}%
\def\markdownRendererRightBracePrototype{\textbraceright}%
\def\markdownRendererDollarSignPrototype{\textdollar}%
\def\markdownRendererPercentSignPrototype{\percent}%
\def\markdownRendererUnderscorePrototype{\textunderscore}%
\def\markdownRendererCircumflexPrototype{\textcircumflex}%
\def\markdownRendererBackslashPrototype{\textbackslash}%
\def\markdownRendererTildePrototype{\textasciitilde}%
\def\markdownRendererPipePrototype{\char`|}%
\def\markdownRendererLinkPrototype#1#2#3#4{%
  \useURL[#1][#3][][#4]#1\footnote[#1]{\ifx\empty#4\empty\else#4:
  \fi\tt<\hyphenatedurl{#3}>}}%
\usemodule[database]
\defineseparatedlist
  [MarkdownConTeXtCSV]
  [separator={,},
   before=\bTABLE,after=\eTABLE,
   first=\bTR,last=\eTR,
   left=\bTD,right=\eTD]
\def\markdownConTeXtCSV{csv}
\def\markdownRendererContentBlockPrototype#1#2#3#4{%
  \def\markdownConTeXtCSV@arg{#1}%
  \ifx\markdownConTeXtCSV@arg\markdownConTeXtCSV
    \placetable[][tab:#1]{#4}{%
      \processseparatedfile[MarkdownConTeXtCSV][#3]}%
  \else
    \markdownInput{#3}%
  \fi}%
\def\markdownRendererImagePrototype#1#2#3#4{%
  \placefigure[][]{#4}{\externalfigure[#3]}}%
\def\markdownRendererUlBeginPrototype{\startitemize}%
\def\markdownRendererUlBeginTightPrototype{\startitemize[packed]}%
\def\markdownRendererUlItemPrototype{\item}%
\def\markdownRendererUlEndPrototype{\stopitemize}%
\def\markdownRendererUlEndTightPrototype{\stopitemize}%
\def\markdownRendererOlBeginPrototype{\startitemize[n]}%
\def\markdownRendererOlBeginTightPrototype{\startitemize[packed,n]}%
\def\markdownRendererOlItemPrototype{\item}%
\def\markdownRendererOlItemWithNumberPrototype#1{\sym{#1.}}%
\def\markdownRendererOlEndPrototype{\stopitemize}%
```

```
\def\markdownRendererOlEndTightPrototype{\stopitemize}%
\definedescription
  [MarkdownConTeXtDlItemPrototype]
  [location=hanging,
   margin=standard,
   headstyle=bold]%
\definestartstop
  [MarkdownConTeXtDlPrototype]
  [before=\blank,
   after=\blank]%
\definestartstop
  [MarkdownConTeXtDlTightPrototype]
  [before=\blank\startpacked,
   after=\stoppacked\blank]%
\def\markdownRendererDlBeginPrototype{%
  \startMarkdownConTeXtDlPrototype}%
\def\markdownRendererDlBeginTightPrototype{%
  \startMarkdownConTeXtDlTightPrototype}%
\def\markdownRendererDlItemPrototype#1{%
  \startMarkdownConTeXtDlItemPrototype{#1}}%
\def\markdownRendererDlItemEndPrototype{%
  \stopMarkdownConTeXtDlItemPrototype}%
\def\markdownRendererDlEndPrototype{%
  \stopMarkdownConTeXtDlPrototype}%
\def\markdownRendererDlEndTightPrototype{%
  \stopMarkdownConTeXtDlTightPrototype}%
\def\markdownRendererEmphasisPrototype#1{{\em#1}}%
\def\markdownRendererStrongEmphasisPrototype#1{{\bf#1}}%
\def\markdownRendererBlockQuoteBeginPrototype{\startquotation}%
\def\markdownRendererBlockQuoteEndPrototype{\stopquotation}%
\def\markdownRendererLineBlockBeginPrototype{%
  \begingroup
    \def\markdownRendererLineBreak{
    }%
    \startlines
}%
\def\markdownRendererLineBlockEndPrototype{%
    \stoplines
  \endgroup
}%
\def\markdownRendererInputVerbatimPrototype#1{\typefile{#1}}%
```

3.4.2.1 Fenced Code

When no infostring has been specified, default to the indented code block renderer.

```
\ExplSyntaxOn
\cs_gset:Npn
```

```
  \markdownRendererInputFencedCodePrototype#1#2
  {
    \tl_if_empty:nTF
      { #2 }
      { \markdownRendererInputVerbatim{#1} }
```

Otherwise, extract the first word of the infostring and treat it as the name of the programming language in which the code block is written. This name is then used in the ConTEXt `\definetyping` macro, which allows the user to set up code highlighting mapping as follows:

```
\definetyping [latex]
\setuptyping  [latex] [option=TEX]

\starttext
  \startmarkdown
~~~ latex
\documentclass{article}
\begin{document}
  Hello world!
\end{document}
~~~
  \stopmarkdown
\stoptext
```

```
      {
        \regex_extract_once:nnN
          { \w* }
          { #2 }
          \l_tmpa_seq
        \seq_pop_left:NN
          \l_tmpa_seq
          \l_tmpa_tl
        \typefile[\l_tmpa_tl][]{#1}
      }
  }
\ExplSyntaxOff
\def\markdownRendererHeadingOnePrototype#1{\chapter{#1}}%
\def\markdownRendererHeadingTwoPrototype#1{\section{#1}}%
\def\markdownRendererHeadingThreePrototype#1{\subsection{#1}}%
\def\markdownRendererHeadingFourPrototype#1{\subsubsection{#1}}%
\def\markdownRendererHeadingFivePrototype#1{\subsubsubsection{#1}}%
\def\markdownRendererHeadingSixPrototype#1{\subsubsubsubsection{#1}}%
\def\markdownRendererThematicBreakPrototype{%
  \blackrule[height=1pt, width=\hsize]}%
```

```
\def\markdownRendererNotePrototype#1{\footnote{#1}}%
\def\markdownRendererTickedBoxPrototype{$\boxtimes$}
\def\markdownRendererHalfTickedBoxPrototype{$\boxdot$}
\def\markdownRendererUntickedBoxPrototype{$\square$}
\def\markdownRendererStrikeThroughPrototype#1{\overstrikes{#1}}
\def\markdownRendererSuperscriptPrototype#1{\high{#1}}
\def\markdownRendererSubscriptPrototype#1{\low{#1}}
```

3.4.2.2 Tables

There is a basic implementation of tables.

```
\newcount\markdownConTeXtRowCounter
\newcount\markdownConTeXtRowTotal
\newcount\markdownConTeXtColumnCounter
\newcount\markdownConTeXtColumnTotal
\newtoks\markdownConTeXtTable
\newtoks\markdownConTeXtTableFloat
\def\markdownRendererTablePrototype#1#2#3{%
  \markdownConTeXtTable={}%
  \ifx\empty#1\empty
    \markdownConTeXtTableFloat={%
      \the\markdownConTeXtTable}%
  \else
    \markdownConTeXtTableFloat={%
      \placetable{#1}{\the\markdownConTeXtTable}}%
  \fi
  \begingroup
  \setupTABLE[r][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
  \setupTABLE[c][each][topframe=off, bottomframe=off, leftframe=off, rightframe=off]
  \setupTABLE[r][1][topframe=on, bottomframe=on]
  \setupTABLE[r][#1][bottomframe=on]
  \markdownConTeXtRowCounter=0%
  \markdownConTeXtRowTotal=#2%
  \markdownConTeXtColumnTotal=#3%
  \markdownConTeXtRenderTableRow}
\def\markdownConTeXtRenderTableRow#1{%
  \markdownConTeXtColumnCounter=0%
  \ifnum\markdownConTeXtRowCounter=0\relax
    \markdownConTeXtReadAlignments#1%
    \markdownConTeXtTable={\bTABLE}%
  \else
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\bTR}%
    \markdownConTeXtRenderTableCell#1%
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\eTR}%
  \fi
  \advance\markdownConTeXtRowCounter by 1\relax
```

```
  \ifnum\markdownConTeXtRowCounter>\markdownConTeXtRowTotal\relax
    \markdownConTeXtTable=\expandafter{%
      \the\markdownConTeXtTable\eTABLE}%
    \the\markdownConTeXtTableFloat
    \endgroup
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtRenderTableRow}
\def\markdownConTeXtReadAlignments#1{%
  \advance\markdownConTeXtColumnCounter by 1\relax
  \if#1d%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
  \fi\if#1l%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=right]
  \fi\if#1c%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=middle]
  \fi\if#1r%
    \setupTABLE[c][\the\markdownConTeXtColumnCounter][align=left]
  \fi
  \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtReadAlignments}
\def\markdownConTeXtRenderTableCell#1{%
  \advance\markdownConTeXtColumnCounter by 1\relax
  \markdownConTeXtTable=\expandafter{%
    \the\markdownConTeXtTable\bTD#1\eTD}%
  \ifnum\markdownConTeXtColumnCounter<\markdownConTeXtColumnTotal\relax\else
    \expandafter\gobbleoneargument
  \fi\markdownConTeXtRenderTableCell}
```

3.4.2.3 Raw Attribute Renderer Prototypes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `context` to `tex`.

```
\ExplSyntaxOn
\cs_gset:Npn
  \markdownRendererInputRawInlinePrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { latex }
          {
            \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
              { #1 }
              { context }
          }
      }
```

```
      {
        \@@_plain_tex_default_input_raw_inline_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset:Npn
  \markdownRendererInputRawBlockPrototype#1#2
  {
    \str_case:nnF
      { #2 }
      {
        { context }
          {
            \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
              { #1 }
              { tex }
          }
      }
      {
        \@@_plain_tex_default_input_raw_block_renderer_prototype:nn
          { #1 }
          { #2 }
      }
  }
\cs_gset_eq:NN
  \markdownRendererInputRawBlockPrototype
  \markdownRendererInputRawInlinePrototype
\ExplSyntaxOff
\stopmodule\protect
```

References

[1] LuaTEX development team. *LuaTEX reference manual*. Version 1.10 (stable). July 23, 2021. URL: https://www.pragma-ade.com/general/manuals/luatex.pdf (visited on 09/30/2022).

[2] Vít Novotný. *TeXový interpret jazyka Markdown (markdown.sty)*. 2015. URL: https://www.muni.cz/en/research/projects/32984 (visited on 02/19/2018).

[3] Anton Sotkov. *File transclusion syntax for Markdown*. Jan. 19, 2017. URL: https://github.com/iainc/Markdown-Content-Blocks (visited on 01/08/2018).

[4] John MacFarlane. *Pandoc. a universal document converter*. 2022. URL: https://pandoc.org/ (visited on 10/05/2022).

[5] Bonita Sharif and Jonathan I. Maletic. "An Eye Tracking Study on camelCase and under_score Identifier Styles." In: *2010 IEEE 18th International Conference on Program Comprehension*. 2010, pp. 196–205. DOI: 10.1109/ICPC.2010.41.

[6] Donald Ervin Knuth. *The TEXbook*. 3rd ed. Vol. A. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. ix, 479. ISBN: 0-201-13447-0.

[7] Frank Mittelbach. *The `doc` and `shortvrb` Packages*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/doc.pdf (visited on 02/19/2018).

[8] Till Tantau, Joseph Wright, and Vedran Miletić. *The Beamer class*. Feb. 10, 2021. URL: https://mirrors.ctan.org/macros/latex/contrib/beamer/doc/beameruserguide.pdf (visited on 02/11/2021).

[9] Vít Novotný. *LaTeX 2ε no longer keys packages by pathnames*. Feb. 20, 2021. URL: https://github.com/latex3/latex2e/issues/510 (visited on 02/21/2021).

[10] Geoffrey M. Poore. *The `minted` Package. Highlighted source code in LaTeX*. July 19, 2017. URL: https://mirrors.ctan.org/macros/latex/contrib/minted/minted.pdf (visited on 09/01/2020).

[11] Roberto Ierusalimschy. *Programming in Lua*. 3rd ed. Rio de Janeiro: PUC-Rio, 2013. xviii, 347. ISBN: 978-85-903798-5-0.

[12] Johannes Braams et al. *The LaTeX 2ε Sources*. Apr. 15, 2017. URL: https://mirrors.ctan.org/macros/latex/base/source2e.pdf (visited on 01/08/2018).

[13] Donald Ervin Knuth. *TEX: The Program*. Vol. B. Computers & Typesetting. Reading, MA: Addison-Wesley, 1986. xvi, 594. ISBN: 0-201-13437-7.

[14] Victor Eijkhout. *TEX by Topic. A TEXnician's Reference*. Wokingham, England: Addison-Wesley, Feb. 1, 1992. 307 pp. ISBN: 0-201-56882-0.

Index

`blankBeforeBlockquote` *16*
`blankBeforeCodeFence` *17*
`blankBeforeDivFence` *17*
`blankBeforeHeading` *17*
`bracketedSpans` *18*, 52
`breakableBlockquotes` *18*

`cacheDir` 3, 15, 21, 45, 46, 103, 200, 248, 260
`citationNbsps` *18*
`citations` *19*, 75, 81
`codeSpans` *19*
`contentBlocks` 15, *20*

contentBlocksLanguageMap 15

debugExtensions 8, 16, *20*, 196
debugExtensionsFileName 16, 20
defaultOptions *9*, 41, 233
definitionLists *20*, 57

eagerCache *21*, 21
\endmarkdown *91*
entities.char_entity *159*
entities.dec_entity *159*
entities.hex_entity *159*
errorTempFileName 47, 254
escape_citation *202*, 202
escaped_citation_chars *202*, 202
expandtabs *182*
expectJekyllData *22*
extensions 23, *110*, 201
extensions.bracketed_spans *201*
extensions.citations *202*
extensions.content_blocks *205*
extensions.definition_lists *209*
extensions.fancy_lists *210*
extensions.fenced_code *214*
extensions.fenced_divs *218*
extensions.header_attributes *220*
extensions.jekyll_data *230*
extensions.line_blocks *222*
extensions.notes *223*
extensions.pipe_table *224*
extensions.raw_inline *227*
extensions.strike_through *228*
extensions.subscripts *229*
extensions.superscripts *229*

fancyLists *24*, 70–74, 263
fencedCode *25*, 32, 55, 60, 76, 259
fencedCodeAttributes *25*
fencedDiv 61
fencedDivs *26*
finalizeCache 16, 21, *26*, 27, 45, 46, 102, 200
frozenCache 16, 26, 46, 97, 102, 259, 260
frozenCacheCounter 27, 200, 256

frozenCacheFileName 16, 26, 45, 200

hardLineBreaks *27*
hashEnumerators *27*
headerAttributes *28*, 33, 50, 62
helperScriptFileName 46–48, 253, 254
html *28*, 64, 65, 271
hybrid *28*, 32, 37, 38, 47, 76, 98, 103, 162, 183, 202, 255

inlineNotes *29*
\inputmarkdown *107*, 107, 108, 283
inputTempFileName 250, 251
iterlines *182*

jekyllData 3, 22, *29*, 83–86

languages_json *206*, 206
lineBlocks *30*, 66

\markdown *91*
markdown *91*, 91, 257
markdown* *91*, 91, 92, 257
\markdown_jekyll_data_concatenate_address:NN *244*
\markdown_jekyll_data_pop: *244*
\markdown_jekyll_data_push:nN *244*
\markdown_jekyll_data_push_address_segment:n *242*
\markdown_jekyll_data_set_keyval:Nn *245*
\markdown_jekyll_data_set_keyvals:nn *245*
\markdown_jekyll_data_update_address_tls: *244*
\markdownBegin *43*, 43, 44, 88, 91, 107
\markdownEnd *43*, 43, 44, 88, 91, 107
\markdownError *88*, 88
\markdownEscape *43*, 44, 45, 256
\markdownExecute *253*
\markdownExecuteDirect *253*, 253
\markdownExecuteShellEscape *252*, 253
\markdownIfOption *248*
\markdownIfSnippetExists *92*
\markdownInfo *88*
\markdownInput *43*, 44, 91, 92, 107, 255, 257
\markdownInputFileStream *249*
\markdownInputPlainTeX *257*
\markdownLuaExecute 252, *253*, 254, 255

\markdownLuaOptions *246*, 248
\markdownLuaRegisterIBCallback *90*
\markdownLuaUnregisterIBCallback *90*
\markdownMakeOther *88*, 283
\markdownMode 4, 46, 47, *89*, 89, 251, 252, 254
\markdownOptionErrorTempFileName *47*
\markdownOptionFinalizeCache *45*
\markdownOptionFrozenCache *45*
\markdownOptionHelperScriptFileName *46*
\markdownOptionHybrid *47*
\markdownOptionInputTempFileName *46*
\markdownOptionOutputDir *47*
\markdownOptionOutputTempFileName *46*
\markdownOptionStripPercentSigns *49*
\markdownOutputFileStream *249*
\markdownPrepare *248*
\markdownPrepareLuaOptions *246*
\markdownReadAndConvert *88*, 249, 257, 284
\markdownReadAndConvertProcessLine *250*, 251
\markdownReadAndConvertStripPercentSigns *250*
\markdownReadAndConvertTab *249*
\markdownRendererAttributeClassName *50*
\markdownRendererAttributeIdentifier *50*
\markdownRendererAttributeKeyValue *50*
\markdownRendererBlockHtmlCommentBegin *64*
\markdownRendererBlockHtmlCommentEnd *64*
\markdownRendererBlockQuoteBegin *51*
\markdownRendererBlockQuoteEnd *51*
\markdownRendererBracketedSpanAttributeContextBegin *52*
\markdownRendererBracketedSpanAttributeContextEnd *52*
\markdownRendererCite *75*, 81
\markdownRendererCodeSpan *55*
\markdownRendererContentBlock *55*, 56
\markdownRendererContentBlockCode *56*
\markdownRendererContentBlockOnlineImage *56*
\markdownRendererDlBegin *57*
\markdownRendererDlBeginTight *57*
\markdownRendererDlDefinitionBegin *58*
\markdownRendererDlDefinitionEnd *58*
\markdownRendererDlEnd *59*
\markdownRendererDlEndTight *59*
\markdownRendererDlItem *57*

\markdownRendererDlItemEnd *58*
\markdownRendererDocumentBegin *68*
\markdownRendererDocumentEnd *68*
\markdownRendererEllipsis 34, *59*
\markdownRendererEmphasis *60*, 104
\markdownRendererFancyOlBegin *70*, 71
\markdownRendererFancyOlBeginTight *71*
\markdownRendererFancyOlEnd *74*
\markdownRendererFancyOlEndTight *74*
\markdownRendererFancyOlItem *72*
\markdownRendererFancyOlItemEnd *73*
\markdownRendererFancyOlItemWithNumber *73*
\markdownRendererFencedCodeAttributeContextBegin *60*
\markdownRendererFencedCodeAttributeContextEnd *60*
\markdownRendererFencedDivAttributeContextBegin *61*
\markdownRendererFencedDivAttributeContextEnd *61*
\markdownRendererFootnote *69*, 87
\markdownRendererFootnotePrototype *69*, 87
\markdownRendererHalfTickedBox *82*
\markdownRendererHeaderAttributeContextBegin *62*
\markdownRendererHeaderAttributeContextEnd *62*
\markdownRendererHeadingFive *63*
\markdownRendererHeadingFour *63*
\markdownRendererHeadingOne *62*
\markdownRendererHeadingSix *64*
\markdownRendererHeadingThree *63*
\markdownRendererHeadingTwo *63*
\markdownRendererHorizontalRule *81*, 87
\markdownRendererHorizontalRulePrototype *81*, 87
\markdownRendererImage *66*
\markdownRendererInlineHtmlComment *64*
\markdownRendererInlineHtmlTag *65*
\markdownRendererInputBlockHtmlElement *65*
\markdownRendererInputFencedCode *55*
\markdownRendererInputRawBlock *75*
\markdownRendererInputRawInline *75*
\markdownRendererInputVerbatim *54*
\markdownRendererInterblockSeparator *66*
\markdownRendererJekyllDataBegin *83*
\markdownRendererJekyllDataBoolean *85*
\markdownRendererJekyllDataEmpty *86*
\markdownRendererJekyllDataEnd *83*

\markdownRendererJekyllDataMappingBegin *83*
\markdownRendererJekyllDataMappingEnd *84*
\markdownRendererJekyllDataNumber *85*
\markdownRendererJekyllDataSequenceBegin *84*
\markdownRendererJekyllDataSequenceEnd *85*
\markdownRendererJekyllDataString *86*
\markdownRendererLineBlockBegin *66*
\markdownRendererLineBlockEnd *66*
\markdownRendererLineBreak *67*
\markdownRendererLink *67*, 104
\markdownRendererNbsp *68*
\markdownRendererNote *69*
\markdownRendererOlBegin *70*
\markdownRendererOlBeginTight *70*
\markdownRendererOlEnd *73*
\markdownRendererOlEndTight *74*
\markdownRendererOlItem 34, *71*
\markdownRendererOlItemEnd *72*
\markdownRendererOlItemWithNumber 34, *72*
\markdownRendererReplacementCharacter *76*
\markdownRendererStrikeThrough *79*
\markdownRendererStrongEmphasis *60*
\markdownRendererSubscript *79*
\markdownRendererSuperscript *80*
\markdownRendererTable *80*
\markdownRendererTextCite *80*
\markdownRendererThematicBreak *81*
\markdownRendererTickedBox *82*
\markdownRendererUlBegin *52*
\markdownRendererUlBeginTight *53*
\markdownRendererUlEnd *54*
\markdownRendererUlEndTight *54*
\markdownRendererUlItem *53*
\markdownRendererUlItemEnd *53*
\markdownRendererUntickedBox *82*
\markdownSetup *92*, 92, 257, 261
\markdownSetupSnippet *92*, 92
\markdownWarning *88*

new *7*, 233
notes *30*, 69

outputTempFileName 47, 254

parsers *170*, 181
parsers.commented_line *172*
pipeTables 6, *31*, 36, 80
preserveTabs *32*, 35, 182

rawAttribute *32*, 32, 75, 76
reader *7*, 110, 170, 181, 201
reader->add_special_character *7*, 9, 195
reader->create_parser *182*
reader->finalize_grammar *192*
reader->initialize_named_group *196*
reader->insert_pattern *7*, 8, 193, 197
reader->normalize_tag *181*
reader->options *181*
reader->parser_functions *182*
reader->parser_functions.name *182*
reader->parsers *181*, 181
reader->update_rule 193, 195, 198
reader->writer *181*
reader.new *181*, 181
relativeReferences *33*

\setupmarkdown *108*, 108
shiftHeadings 6, 33
slice 6, 33, 160
smartEllipses *34*, 59, 103
\startmarkdown *107*, 107, 284
startNumber *34*, 71–73
\stopmarkdown *107*, 107, 284
strikeThrough *35*, 79, 281
stripIndent *35*, 182
stripPercentSigns 249, 250
subscripts *35*, 79
superscripts *36*, 80
syntax *194*, 197

tableCaptions 6, *36*
taskLists *37*, 82, 271
texComments *37*, 183
tightLists *38*, 53, 54, 57, 59, 70, 71, 74, 263

underscores *38*
util.cache *111*, 111
util.cache_verbatim *111*
util.encode_json_string *111*
util.err *111*
util.escaper *114*
util.expand_tabs_in_line *112*
util.flatten *113*
util.intersperse *114*
util.lookup_files *112*
util.map *114*
util.pathname *115*
util.rope_last *113*
util.rope_to_string *113*
util.table_copy *111*
util.walk *112*, 113

walkable_syntax *8*, 16, 20, 193, 195–197
writer *110*, 110, 159, 201
writer->active_attributes *167*
writer->active_headings *167*
writer->attributes *166*
writer->block_html_comment *165*
writer->block_html_element *165*
writer->blockquote *166*
writer->bulletitem *164*
writer->bulletlist *163*
writer->citation *202*
writer->citations *202*
writer->code *163*
writer->contentblock *206*
writer->defer_call *170*, 170
writer->definitionlist *209*
writer->div *218*
writer->document *166*
writer->ellipsis *161*
writer->emphasis *165*
writer->escape *162*, 162
writer->escape_minimal *162*, 162, 202
writer->escape_uri *162*, 162
writer->escaped_chars *162*, 162
writer->escaped_minimal_strings *161*, 162

writer->escaped_strings 162
writer->escaped_uri_chars 161, 162
writer->fancyitem 211
writer->fancylist 211
writer->fencedCode 215
writer->get_state 169
writer->heading 167
writer->image 163
writer->inline_html_comment 164
writer->inline_html_tag 165
writer->interblocksep 161
writer->is_writing 160, 160
writer->jekyllData 230
writer->lineblock 222
writer->linebreak 161
writer->link 163
writer->nbsp 161
writer->note 223
writer->options 160
writer->ordereditem 164
writer->orderedlist 164
writer->pack 161, 200
writer->paragraph 161
writer->plain 161
writer->rawBlock 215
writer->rawInline 228
writer->set_state 169
writer->slice_begin 160
writer->slice_end 160
writer->space 160
writer->span 201
writer->strike_through 228
writer->string 162
writer->strong 165
writer->subscript 229
writer->suffix 160
writer->superscript 230
writer->table 226
writer->thematic_break 161
writer->tickbox 165
writer->uri 162
writer->verbatim 166

writer.new 159, *160*, 160