A Markdown Interpreter for TEX

Vít Novotný
witiko@mail.muni.cz

Version 2.22.0-0-g5a3d0fe
2023-04-02

Contents

List of Figures

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",
                 "2016-2023 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 (though not necessarily in the LuaMetaTeX 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 (could be because we are using LuaMetaTeX) 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");
```

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

```
(function()
```

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

```
  local should_initialize = package.loaded.kpse == nil
                         or tex.initialize ~= nil
  local ran_ok
  ran_ok, kpse = pcall(require, "kpse")
  if ran_ok and should_initialize then
    kpse.set_program_name("luatex")
  end
```

If the Kpathsea library is unavailable, we will look up files only in the current working directory.

```
  if not ran_ok then
    kpse = {lookup = function(f, _) return f end}
  end
end)()
```

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

lua-uni-algos A package that implements Unicode case-folding in TEX Live ⩾ 2020.

```
local uni_case
(function()
  local ran_ok
  ran_ok, uni_case = pcall(require, "lua-uni-case")
```

If the lua-uni-algos library is unavailable but the Selene Unicode library is available, we will use its Unicode lower-casing support instead of the more proper case-folding.

```
  if not ran_ok then
    if unicode.utf8.lower then
      uni_case = {casefold = unicode.utf8.lower}
    else
```

If the Selene Unicode library is also unavailable, we will defer to using ASCII lower-casing.

```
      uni_case = {casefold = string.lower}
    end
  end
end)()
```

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. We carry a copy of the library in file `markdown-tinyyaml.lua` distributed together with the Markdown package.

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 $\leqslant$ 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 $\geqslant$ 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,

```
50 \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 2.3.2.3) and will not be loaded if the `plain` package option has been enabled (see Section 2.3.2.2):

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 2.3.2.3).

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 2.3.2.3.

graphicx A package that builds upon the graphics package, which is part of the LaTeX 2_ε kernel. It provides a key-value interface that is used in the default renderer prototypes for image attribute contexts.

grffile A package that extends the name processing of the graphics package to support a larger range of file names in 2006 $\leqslant$ TeX Live $\leqslant$ 2019. Since TeX Live $\geqslant$ 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 2.3.2.3.

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.

```
51 \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.

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

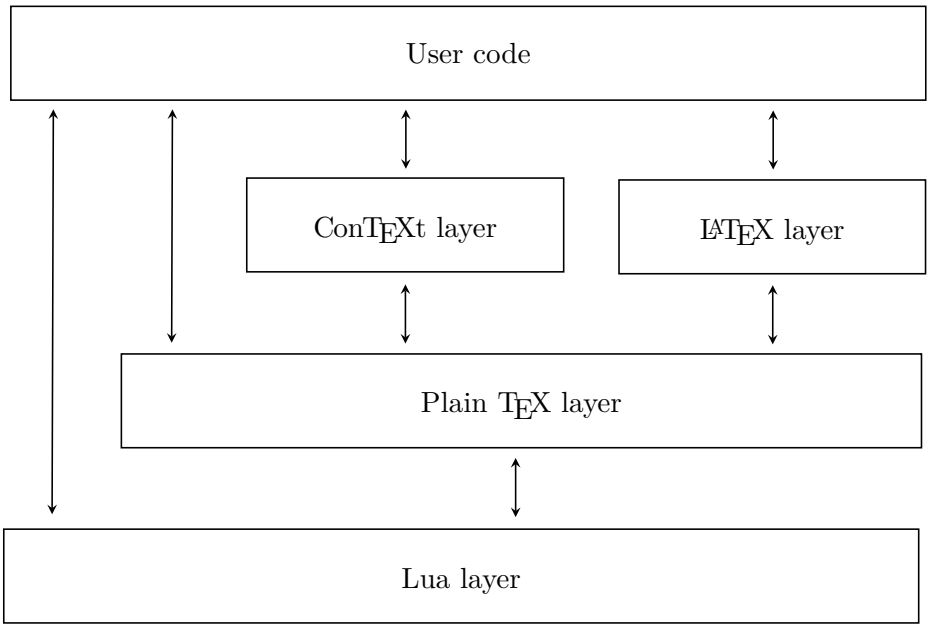

Figure 1: A block diagram of the Markdown package

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 2.2.3.8 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 span syntax extension[6]:

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

`false` Disable the Pandoc bracketed span syntax 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
```

[6]See https://pandoc.org/MANUAL.html#extension-bracketed_spans.

`citations`=true, false default: `false`

`true` Enable the Pandoc citation syntax extension[7]:

```
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
```

[7] See https://pandoc.org/MANUAL.html#extension-citations.

`contentBlocks`=true, `false` default: `false`

true

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

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

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

```
\@@_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 }
```

```
441 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 syntax extension[8]:

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

`false` Disable the Pandoc fancy list syntax extension.

[8] See https://pandoc.org/MANUAL.html#org-fancy-lists.

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

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

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

`true` Enable the Pandoc fenced code attribute syntax extension[9]:

```
~~~~ {#mycode .haskell .numberLines startFrom=100}
qsort []     = []
```

[9]See https://pandoc.org/MANUAL.html#extension-fenced_code_attributes.

```
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++
               qsort (filter (>= x) xs)
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

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

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

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

`true` Enable the Pandoc fenced div syntax extension[10]:

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

And another.
:::::
```

`false` Disable the Pandoc fenced div syntax 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.

[10] See https://pandoc.org/MANUAL.html#extension-fenced_divs.

```
\@@_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 }
```

The hardLineBreaks option has been deprecated and will be removed in Markdown 3.0.0. From then on, all line breaks within a paragraph will be interpreted as soft line breaks.

```
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
```

`inlineCodeAttributes`=true, false default: `false`

`true` Enable the Pandoc inline code span attribute extension[11]:

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

`false` Enable the Pandoc inline code span attribute extension.

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

[11] See https://pandoc.org/MANUAL.html#extension-inline_code_attributes.

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

`true` Enable the Pandoc inline note syntax extension[12]:

```
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[13] 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.

[12] See https://pandoc.org/MANUAL.html#extension-inline_notes.
[13] See https://pandoc.org/MANUAL.html#extension-yaml_metadata_block.

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

`linkAttributes`=true, false default: `false`

`true` Enable the Pandoc link and image attribute syntax extension[14]:

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

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

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

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

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

`true` Enable the Pandoc line block syntax extension[15]:

```
| 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
```

[14] See https://pandoc.org/MANUAL.html#extension-link_attributes.
[15] See https://pandoc.org/MANUAL.html#extension-line_blocks.

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

`true` Enable the Pandoc note syntax extension[16]:

```
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 |
|------:|:-----|---------|:------:|
```

[16] See https://pandoc.org/MANUAL.html#extension-footnotes.

```
|   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[17]:

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

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

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

[17] See https://pandoc.org/MANUAL.html#extension-raw_attribute.

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[18] 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
```

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

`slice`=⟨*the beginning and the end of a slice*⟩ default: `^ $`

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

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

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

```
\@@_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[19]:

```
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
```

[19] See https://pandoc.org/MANUAL.html#extension-strikeout.

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

`true` Enable the Pandoc subscript syntax extension[20]:

```
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[21]:

```
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 caption syntax extension[22] for pipe tables (see the `pipeTables` option).

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

[20] See https://pandoc.org/MANUAL.html#extension-superscript-subscript.
[21] See https://pandoc.org/MANUAL.html#extension-superscript-subscript.
[22] See https://pandoc.org/MANUAL.html#extension-table_captions.
```

```
  : Demonstration of pipe table syntax.
``````
```

`false` Disable the Pandoc table caption syntax extension.

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

`taskLists`=true, false default: `false`

`true` Enable the Pandoc task list syntax extension[23]:

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

`false` Disable the Pandoc task list syntax extension.

```
\@@_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.

[23] See https://pandoc.org/MANUAL.html#extension-task_lists.
```

`false` Do not strip TEX-style comments.

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

`texMathDollars`=`true`, `false` default: `false`

`true` Enable the Pandoc dollar math syntax extension[24]:

```
inline math: $E=mc^2$

display math: $$E=mc^2$$
```

`false` Disable the Pandoc dollar math syntax extension.

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

`texMathDoubleBackslash`=`true`, `false` default: `false`

`true` Enable the Pandoc double backslash math syntax extension[25]:

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

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

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

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

[24] See https://pandoc.org/MANUAL.html#extension-tex_math_dollars.
[25] See https://pandoc.org/MANUAL.html#extension-tex_math_double_backslash.

`texMathSingleBackslash`=`true`, `false` default: `false`

`true` Enable the Pandoc single backslash math syntax extension[26]:

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

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

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

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

`tightLists`=`true`, `false` default: `true`

`true` Unordered and ordered lists whose items do not consist of multiple paragraphs will be considered *tight*. Tight lists will produce tight renderers that may produce different output than lists that are not tight:

```
- This is
- a tight
- unordered list.

- This is

  not a tight

- unordered list.
```

`false` Unordered and ordered lists whose items consist of multiple paragraphs will be treated the same way as lists that consist of multiple paragraphs.

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

[26] See https://pandoc.org/MANUAL.html#extension-tex_math_single_backslash.

`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.

```

local HELP_STRING = [[
Usage: texlua ]] .. arg[0] .. [[ [OPTIONS] -- [INPUT_FILE] [OUTPUT_FILE]
where OPTIONS are documented in the Lua interface section of the
technical Markdown package documentation.

When OUTPUT_FILE is unspecified, the result of the conversion will be
written to the standard output. When INPUT_FILE is also unspecified, the
```

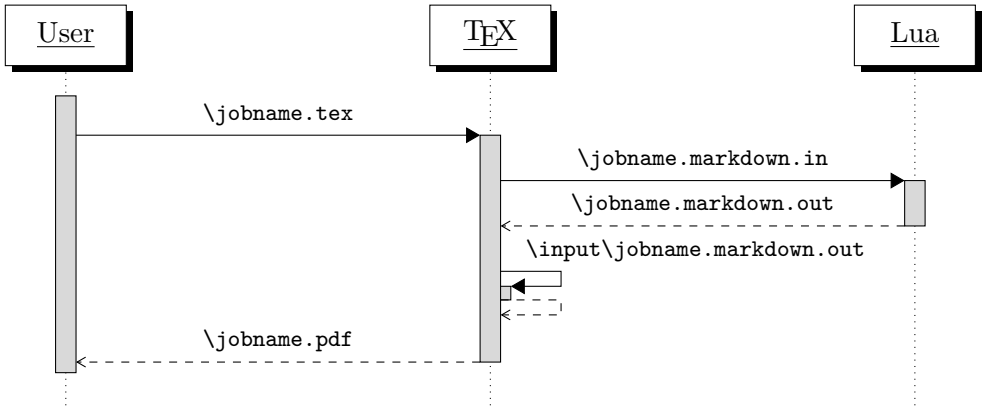

Figure 2: A sequence diagram of the Markdown package typesetting a markdown document using the TEX interface

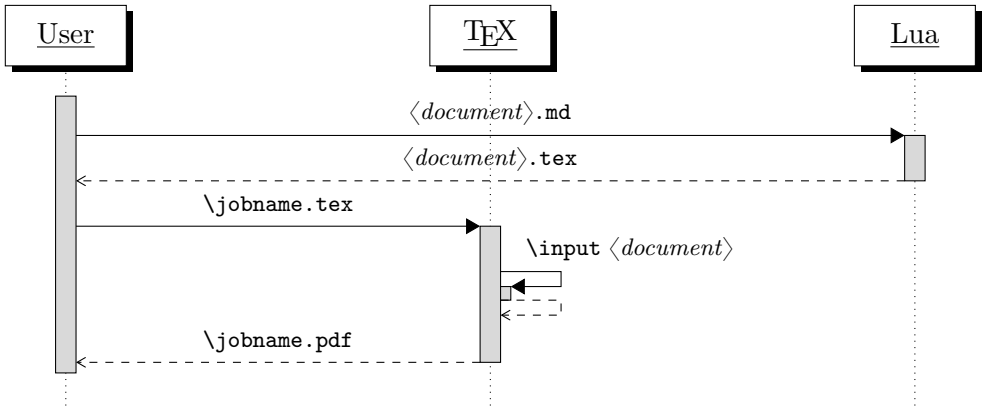

Figure 3: A sequence diagram of the Markdown package typesetting a markdown document using the Lua command-line interface

```
result of the conversion will be read from the standard input.

Report bugs to: witiko@mail.muni.cz
Markdown package home page: <https://github.com/witiko/markdown>]]

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 and
         various_case_options[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
```

```
739         options[key] = value
740       end
741       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.

```
742     elseif arg[i] == "--help" or arg[i] == "-h" then
743       print(HELP_STRING)
744       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.

```
745     elseif arg[i] == "--version" or arg[i] == "-v" then
746       print(VERSION_STRING)
747       os.exit()
748     end
749   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.

```
750   if input_filename == nil then
751     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.

```
752   elseif output_filename == nil then
753     output_filename = arg[i]
754   else
755     error('Unexpected argument: "' .. arg[i] .. '".')
756   end
757   ::continue::
758 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.

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

```
\let\markdownBegin\relax
\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.

```
763 \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.

```
764 \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 }
  { 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.

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

```
\str_new:N
  \g_luabridge_helper_script_filename_str
\tl_gset:Nn
  \g_luabridge_helper_script_filename_str
  { \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.

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

```
\@@_add_plain_tex_option:nnn
  { 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 Code Span Attribute Context Renderers

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

The `\markdownRendererCodeSpanAttributeContextBegin` and `\markdownRendererCodeSpanA` macros represent the beginning and the end of an inline code span in which the attributes of the inline code span apply. The macros receive no arguments.

```
\def\markdownRendererCodeSpanAttributeContextBegin{%
  \markdownRendererCodeSpanAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { codeSpanAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { codeSpanAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererCodeSpanAttributeContextEnd{%
  \markdownRendererCodeSpanAttributeContextEndPrototype}%
\ExplSyntaxOn
```

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

2.2.3.8 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[27] 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
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { contentBlockCode }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { contentBlockCode }
  { 5 }
\ExplSyntaxOff
```

2.2.3.9 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
```

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

The \markdownRendererDlBeginTight macro represents the beginning of a definition list that contains 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
```

2.2.3.10 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
```

2.2.3.11 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
```

2.2.3.12 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
```

2.2.3.13 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
```

2.2.3.14 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.

These semantics have been deprecated and will be changed in Markdown 3.0.0. From then on, header attribute contexts will only span headings, not the surrounding sections.

```
\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
```

2.2.3.15 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
```

2.2.3.16 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
```

2.2.3.17 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
```

2.2.3.18 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
```

2.2.3.19 Image Attribute Context Renderers

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

The `\markdownRendererImageAttributeContextBegin` and `\markdownRendererImageAttribu` macros represent the beginning and the end of an image in which the attributes of the image apply. The macros receive no arguments.

```
\def\markdownRendererImageAttributeContextBegin{%
  \markdownRendererImageAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { imageAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { imageAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererImageAttributeContextEnd{%
  \markdownRendererImageAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { imageAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { imageAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.20 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
```

2.2.3.21 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
```

2.2.3.22 Line Break Renderer

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

The `\markdownRendererLineBreak` and `\markdownRendererLineBreakPrototype` macros have been deprecated and will be removed in Markdown 3.0.0.

```
\ExplSyntaxOn
\cs_new:Npn
  \markdownRendererHardLineBreak
  {
    \cs_if_exist:NTF
      \markdownRendererLineBreak
      {
        \markdownWarning
          {
            Line~break~renderer~has~been~deprecated,~
            to~be~removed~in~Markdown~3.0.0
          }
        \markdownRendererLineBreak
```

```
      }
      {
        \cs_if_exist:NTF
          \markdownRendererLineBreakPrototype
          {
            \markdownWarning
              {
                Line~break~renderer~prototype~has~been~deprecated,~
                to~be~removed~in~Markdown~3.0.0
              }
            \markdownRendererLineBreakPrototype
          }
          {
            \markdownRendererHardLineBreakPrototype
          }
      }
  }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { lineBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { lineBreak }
  { 0 }
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { hardLineBreak }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { hardLineBreak }
  { 0 }
\ExplSyntaxOff
```

2.2.3.23 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
```

2.2.3.24 Link Attribute Context Renderers

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

The `\markdownRendererLinkAttributeContextBegin` and `\markdownRendererLinkAttributeC` macros represent the beginning and the end of a hyperlink in which the attributes of the hyperlink apply. The macros receive no arguments.

```
\def\markdownRendererLinkAttributeContextBegin{%
  \markdownRendererLinkAttributeContextBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { linkAttributeContextBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { linkAttributeContextBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererLinkAttributeContextEnd{%
  \markdownRendererLinkAttributeContextEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { linkAttributeContextEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { linkAttributeContextEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.25 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
```

2.2.3.26 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
```

2.2.3.27 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
```

2.2.3.28 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
```

2.2.3.29 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
```

2.2.3.30 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
```

2.2.3.31 Section Renderers

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

```
\def\markdownRendererSectionBegin{%
  \markdownRendererSectionBeginPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
```

```
  \g_@@_renderers_seq
  { sectionBegin }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { sectionBegin }
  { 0 }
\ExplSyntaxOff
\def\markdownRendererSectionEnd{%
  \markdownRendererSectionEndPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { sectionEnd }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { sectionEnd }
  { 0 }
\ExplSyntaxOff
```

2.2.3.32 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
```

2.2.3.33 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
```

2.2.3.34 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
```

2.2.3.35 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 }
```

2.2.3.36 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
```

2.2.3.37 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
```

2.2.3.38 Tex Math Renderers

The `\markdownRendererInlineMath` and `\markdownRendererDisplayMath` macros represent inline and display TEX math. Both macros receive a single argument that corresponds to the tex math content. These macros will only be produced, when the `texMathDollars`, `texMathSingleBackslash`, or `texMathDoubleBackslash` option are enabled.

```
\def\markdownRendererInlineMath{%
  \markdownRendererInlineMathPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { inlineMath }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { inlineMath }
  { 1 }
\ExplSyntaxOff
\def\markdownRendererDisplayMath{%
  \markdownRendererDisplayMathPrototype}%
\ExplSyntaxOn
\seq_gput_right:Nn
  \g_@@_renderers_seq
  { displayMath }
\prop_gput:Nnn
  \g_@@_renderer_arities_prop
  { displayMath }
  { 1 }
\ExplSyntaxOff
```

2.2.3.39 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
```

2.2.3.40 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
```

2.2.3.41 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
```

2.2.3.42 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 2.3.2.6) and `markdownRendererPrototypes` (see Section 2.3.2.7) keys. Furthermore, although the base variant of the `import` key that loads a single LaTeX theme (see Section 2.3.2.3) can be used, the extended variant that can load multiple themes and import snippets from them (see Section 2.3.2.4). 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.

```
\newenvironment{markdown}\relax\relax
\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 2.3.2.2, and the LaTeX themes described in Section 2.3.2.3, and the LaTeX snippets described in Section 2.3.2.1, 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
```

2.3.2.1 LaTeX snippets

We may also set up LaTeX options as *snippets* using the `\markdownSetupSnippet` macro and invoke them later. The `\markdownSetupSnippet` macro receives two arguments: the name of the snippet and the options to store:

```
\ExplSyntaxOn
\cs_new:Nn
  \@@_latex_setup_snippet:nn
  {
    \markdownIfSnippetExists
      { #1 }
      {
        \markdownWarning
          {Redefined~snippet~\markdownLaTeXThemeName#1}
        \csname markdownLaTeXSetupSnippet%
          \markdownLaTeXThemeName#1\endcsname={#2}
      }
      {
        \newtoks\next
          \next={#2}
        \expandafter\let\csname markdownLaTeXSetupSnippet%
          \markdownLaTeXThemeName#1\endcsname=\next
      }
  }
\let\markdownSetupSnippet=\@@_latex_setup_snippet:nn
\ExplSyntaxOff
```

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

```
\newcommand\markdownIfSnippetExists[3]{%
  \@ifundefined
    {markdownLaTeXSetupSnippet\markdownLaTeXThemeName#1}%
    {#3}{#2}}%
```

See Section 2.3.2.3 for information on interactions between snippets and LaTeX themes. See Section 2.3.2.4 for information about invoking the set-up 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 }
  }
```

2.3.2.2 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
```

2.3.2.3 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 `import`=⟨*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[
  import=witiko/beamer/MU,
  import=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 }
  {
    import .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 input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```
      \tl_replace_all:NnV
        \l_tmpa_tl
        { / }
        \c_backslash_str
      \keys_set:nV
        { markdown/latex-options/import }
        \l_tmpa_tl
    },
  }
\cs_generate_variant:Nn
```

```
  \tl_replace_all:Nnn
  { NnV }
```

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

```
\keys_define:nn
  { markdown/latex-options }
  {
    theme .code:n = { \@@_set_latex_theme:n { #1 } },
  }
\ExplSyntaxOff
```

The LaTeX themes have a useful synergy with snippets (see Section 2.3.2.1): To make it less likely that different themes will set up 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 sets up the `product` snippet, the 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[import=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.

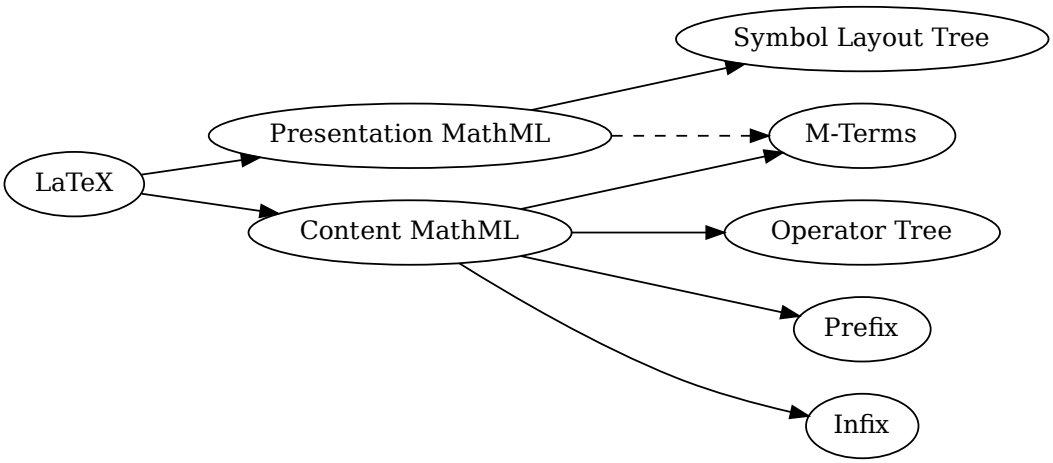


**Figure 4: Various formats of mathemathical formulae**

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.

```
2500 \ProvidesPackage{markdownthemewitiko_dot}[2021/03/09]%
```
```

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

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

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

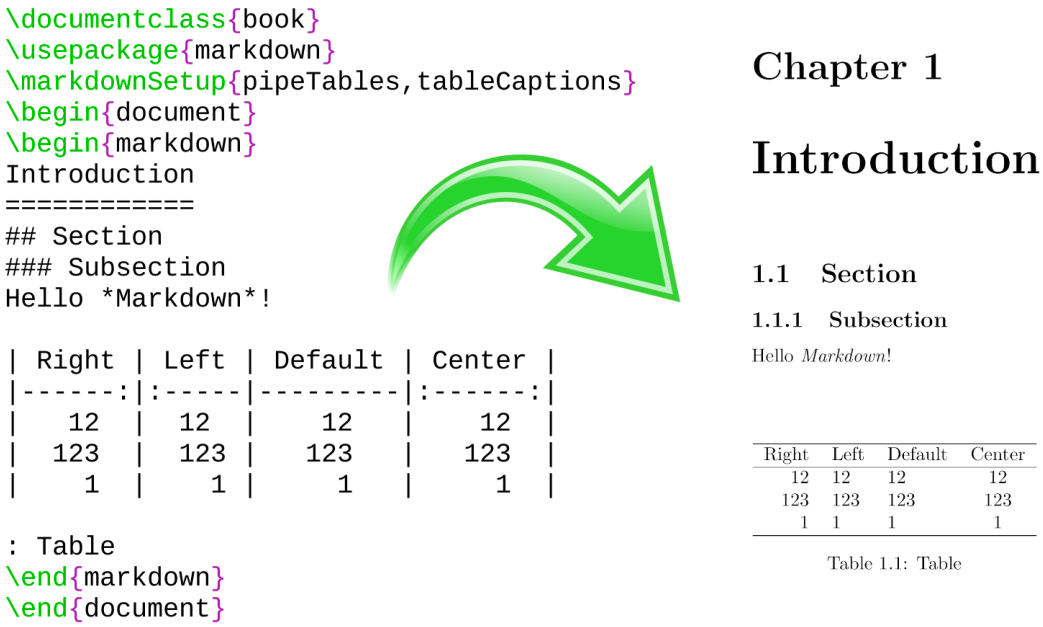

Figure 5: The banner of the Markdown package

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

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

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

```
\documentclass{article}
\usepackage[import=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.

2.3.2.4 LaTeX 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 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*}
```

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

```
\markdownSetup{import=jdoe/lists}
\begin{markdown*}{snippet=jdoe/lists/romanNumerals}

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown*}
```

Alternatively, we can use the extended variant of the `import` LaTeX option that allows us to import the `romanNumerals` snippet to the current namespace, so that we can invoke the snippet with less typing:

```
\markdownSetup{
  import = {
    jdoe/lists = romanNumerals,
  },
}
\begin{markdown*}{snippet=romanNumerals}

The following ordered list will be preceded by roman numerals:
```

```
3. tres
4. quattuor

\end{markdown*}
```

Furthermore, we can also specify the name of the snippet in the current namespace, which is different from the name of the snippet in the `jdoe/lists` theme:

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

The following ordered list will be preceded by roman numerals:

3. tres
4. quattuor

\end{markdown*}
```

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

```
\ExplSyntaxOn
\tl_new:N
  \l_@@_latex_import_current_theme_tl
\keys_define:nn
  { markdown/latex-options/import }
  {
```

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

```
    unknown .default:n = {},
    unknown .code:n = {
```

To ensure that keys containing forward slashes get passed correctly, we replace all forward slashes in the input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```
      \tl_set_eq:NN
```

```
      \l_@@_latex_import_current_theme_tl
      \l_keys_key_str
    \tl_replace_all:NVn
      \l_@@_latex_import_current_theme_tl
      \c_backslash_str
      { / }
```

Here, we load the LaTeX theme.

```
    \@@_set_latex_theme:V
      \l_@@_latex_import_current_theme_tl
```

Here, we import the LaTeX snippets.

```
    \clist_map_inline:nn
      { #1 }
      {
        \regex_extract_once:nnNTF
          { ^(.*?)\s+as\s+(.*?)$ }
          { ##1 }
          \l_tmpa_seq
          {
            \seq_pop:NN
              \l_tmpa_seq
              \l_tmpa_tl
            \seq_pop:NN
              \l_tmpa_seq
              \l_tmpa_tl
            \seq_pop:NN
              \l_tmpa_seq
              \l_tmpb_tl
          }
          {
            \tl_set:Nn
              \l_tmpa_tl
              { ##1 }
            \tl_set:Nn
              \l_tmpb_tl
              { ##1 }
          }
        \tl_put_left:Nn
          \l_tmpa_tl
          { / }
        \tl_put_left:NV
          \l_tmpa_tl
          \l_@@_latex_import_current_theme_tl
        \@@_latex_setup_snippet:Vx
          \l_tmpb_tl
          { snippet = { \l_tmpa_tl } }
      }
```

```
    },
  }
\cs_generate_variant:Nn
  \tl_replace_all:Nnn
  { NVn }
\cs_generate_variant:Nn
  \@@_set_latex_theme:n
  { V }
\cs_generate_variant:Nn
  \@@_latex_setup_snippet:nn
  { Vx }
\ExplSyntaxOff
```

2.3.2.5 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,
}
```

2.3.2.6 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`.
  }
}
```

2.3.2.7 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, format, length =
  string.upper, 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.

```
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.

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

```
function util.cache(dir, string, salt, transform, suffix)
  local digest = md5.sumhexa(string .. (salt or ""))
  local name = util.pathname(dir, digest .. suffix)
  local file = io.open(name, "r")
  if file == nil then -- If no cache entry exists, then create a new one.
    file = assert(io.open(name, "w"),
      [[Could not open file "]] .. name .. [[" for writing]])
    local result = string
    if transform ~= nil then
      result = transform(result)
    end
    assert(file:write(result))
    assert(file:close())
  end
  return name
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.

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

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

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

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

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

The `util.lookup_files` method looks up files with filename `f` and returns their paths. Further options for the Kpathsea library can be specified in table `options`. [1, Section 10.7.4]

```
function util.lookup_files(f, options)
  return kpse.lookup(f, options)
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
    local n
    n = t[i]
    while n do
      util.walk(n, f)
      i = i + 1
      n = t[i]
    end
  elseif typ == "function" then
    local ok, val = pcall(t)
    if ok then
      util.walk(val,f)
    end
  else
    f(tostring(t))
  end
end
```

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

```
function util.flatten(ary)
  local new = {}
  for _,v in ipairs(ary) do
    if type(v) == "table" then
      for _,w in ipairs(util.flatten(v)) do
        new[#new + 1] = w
      end
    else
      new[#new + 1] = v
    end
  end
  return new
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.

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

```
function util.rope_last(rope)
  if #rope == 0 then
    return nil
  else
    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.

```
3030   local char_escapes_list = ""
3031   for i,_ in pairs(char_escapes) do
3032     char_escapes_list = char_escapes_list .. i
3033   end
```

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

```
3034   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.

```
3035   if string_escapes then
3036     for k,v in pairs(string_escapes) do
3037       escapable = P(k) / v + escapable
3038     end
3039   end
```

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

```
3040   local escape_string = Cs((escapable + any)^0)
```

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

```
3041   return function(s)
3042     return lpeg.match(escape_string, s)
3043   end
3044 end
```

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

```
3045 function util.pathname(dir, file)
3046   if #dir == 0 then
3047     return file
3048   else
3049     return dir .. "/" .. file
3050   end
3051 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,
  ["iexcl"] = 161,
  ["cent"] = 162,
  ["pound"] = 163,
  ["curren"] = 164,
  ["yen"] = 165,
  ["brvbar"] = 166,
  ["sect"] = 167,
  ["Dot"] = 168,
  ["die"] = 168,
  ["DoubleDot"] = 168,
  ["uml"] = 168,
  ["copy"] = 169,
  ["COPY"] = 169,
  ["ordf"] = 170,
  ["laquo"] = 171,
  ["not"] = 172,
  ["shy"] = 173,
  ["reg"] = 174,
  ["circledR"] = 174,
  ["REG"] = 174,
  ["macr"] = 175,
  ["OverBar"] = 175,
  ["strns"] = 175,
  ["deg"] = 176,
  ["plusmn"] = 177,
  ["pm"] = 177,
  ["PlusMinus"] = 177,
  ["sup2"] = 178,
  ["sup3"] = 179,
  ["acute"] = 180,
  ["DiacriticalAcute"] = 180,
  ["micro"] = 181,
  ["para"] = 182,
  ["middot"] = 183,
  ["centerdot"] = 183,
  ["CenterDot"] = 183,
  ["cedil"] = 184,
  ["Cedilla"] = 184,
  ["sup1"] = 185,
  ["ordm"] = 186,
```

```
3141    ["raquo"] = 187,
3142    ["frac14"] = 188,
3143    ["frac12"] = 189,
3144    ["half"] = 189,
3145    ["frac34"] = 190,
3146    ["iquest"] = 191,
3147    ["Agrave"] = 192,
3148    ["Aacute"] = 193,
3149    ["Acirc"] = 194,
3150    ["Atilde"] = 195,
3151    ["Auml"] = 196,
3152    ["Aring"] = 197,
3153    ["AElig"] = 198,
3154    ["Ccedil"] = 199,
3155    ["Egrave"] = 200,
3156    ["Eacute"] = 201,
3157    ["Ecirc"] = 202,
3158    ["Euml"] = 203,
3159    ["Igrave"] = 204,
3160    ["Iacute"] = 205,
3161    ["Icirc"] = 206,
3162    ["Iuml"] = 207,
3163    ["ETH"] = 208,
3164    ["Ntilde"] = 209,
3165    ["Ograve"] = 210,
3166    ["Oacute"] = 211,
3167    ["Ocirc"] = 212,
3168    ["Otilde"] = 213,
3169    ["Ouml"] = 214,
3170    ["times"] = 215,
3171    ["Oslash"] = 216,
3172    ["Ugrave"] = 217,
3173    ["Uacute"] = 218,
3174    ["Ucirc"] = 219,
3175    ["Uuml"] = 220,
3176    ["Yacute"] = 221,
3177    ["THORN"] = 222,
3178    ["szlig"] = 223,
3179    ["agrave"] = 224,
3180    ["aacute"] = 225,
3181    ["acirc"] = 226,
3182    ["atilde"] = 227,
3183    ["auml"] = 228,
3184    ["aring"] = 229,
3185    ["aelig"] = 230,
3186    ["ccedil"] = 231,
3187    ["egrave"] = 232,
```

```
3188    ["eacute"] = 233,
3189    ["ecirc"] = 234,
3190    ["euml"] = 235,
3191    ["igrave"] = 236,
3192    ["iacute"] = 237,
3193    ["icirc"] = 238,
3194    ["iuml"] = 239,
3195    ["eth"] = 240,
3196    ["ntilde"] = 241,
3197    ["ograve"] = 242,
3198    ["oacute"] = 243,
3199    ["ocirc"] = 244,
3200    ["otilde"] = 245,
3201    ["ouml"] = 246,
3202    ["divide"] = 247,
3203    ["div"] = 247,
3204    ["oslash"] = 248,
3205    ["ugrave"] = 249,
3206    ["uacute"] = 250,
3207    ["ucirc"] = 251,
3208    ["uuml"] = 252,
3209    ["yacute"] = 253,
3210    ["thorn"] = 254,
3211    ["yuml"] = 255,
3212    ["Amacr"] = 256,
3213    ["amacr"] = 257,
3214    ["Abreve"] = 258,
3215    ["abreve"] = 259,
3216    ["Aogon"] = 260,
3217    ["aogon"] = 261,
3218    ["Cacute"] = 262,
3219    ["cacute"] = 263,
3220    ["Ccirc"] = 264,
3221    ["ccirc"] = 265,
3222    ["Cdot"] = 266,
3223    ["cdot"] = 267,
3224    ["Ccaron"] = 268,
3225    ["ccaron"] = 269,
3226    ["Dcaron"] = 270,
3227    ["dcaron"] = 271,
3228    ["Dstrok"] = 272,
3229    ["dstrok"] = 273,
3230    ["Emacr"] = 274,
3231    ["emacr"] = 275,
3232    ["Edot"] = 278,
3233    ["edot"] = 279,
3234    ["Eogon"] = 280,
```

```
3235    ["eogon"] = 281,
3236    ["Ecaron"] = 282,
3237    ["ecaron"] = 283,
3238    ["Gcirc"] = 284,
3239    ["gcirc"] = 285,
3240    ["Gbreve"] = 286,
3241    ["gbreve"] = 287,
3242    ["Gdot"] = 288,
3243    ["gdot"] = 289,
3244    ["Gcedil"] = 290,
3245    ["Hcirc"] = 292,
3246    ["hcirc"] = 293,
3247    ["Hstrok"] = 294,
3248    ["hstrok"] = 295,
3249    ["Itilde"] = 296,
3250    ["itilde"] = 297,
3251    ["Imacr"] = 298,
3252    ["imacr"] = 299,
3253    ["Iogon"] = 302,
3254    ["iogon"] = 303,
3255    ["Idot"] = 304,
3256    ["imath"] = 305,
3257    ["inodot"] = 305,
3258    ["IJlig"] = 306,
3259    ["ijlig"] = 307,
3260    ["Jcirc"] = 308,
3261    ["jcirc"] = 309,
3262    ["Kcedil"] = 310,
3263    ["kcedil"] = 311,
3264    ["kgreen"] = 312,
3265    ["Lacute"] = 313,
3266    ["lacute"] = 314,
3267    ["Lcedil"] = 315,
3268    ["lcedil"] = 316,
3269    ["Lcaron"] = 317,
3270    ["lcaron"] = 318,
3271    ["Lmidot"] = 319,
3272    ["lmidot"] = 320,
3273    ["Lstrok"] = 321,
3274    ["lstrok"] = 322,
3275    ["Nacute"] = 323,
3276    ["nacute"] = 324,
3277    ["Ncedil"] = 325,
3278    ["ncedil"] = 326,
3279    ["Ncaron"] = 327,
3280    ["ncaron"] = 328,
3281    ["napos"] = 329,
```

```
3282   ["ENG"] = 330,
3283   ["eng"] = 331,
3284   ["Omacr"] = 332,
3285   ["omacr"] = 333,
3286   ["Odblac"] = 336,
3287   ["odblac"] = 337,
3288   ["OElig"] = 338,
3289   ["oelig"] = 339,
3290   ["Racute"] = 340,
3291   ["racute"] = 341,
3292   ["Rcedil"] = 342,
3293   ["rcedil"] = 343,
3294   ["Rcaron"] = 344,
3295   ["rcaron"] = 345,
3296   ["Sacute"] = 346,
3297   ["sacute"] = 347,
3298   ["Scirc"] = 348,
3299   ["scirc"] = 349,
3300   ["Scedil"] = 350,
3301   ["scedil"] = 351,
3302   ["Scaron"] = 352,
3303   ["scaron"] = 353,
3304   ["Tcedil"] = 354,
3305   ["tcedil"] = 355,
3306   ["Tcaron"] = 356,
3307   ["tcaron"] = 357,
3308   ["Tstrok"] = 358,
3309   ["tstrok"] = 359,
3310   ["Utilde"] = 360,
3311   ["utilde"] = 361,
3312   ["Umacr"] = 362,
3313   ["umacr"] = 363,
3314   ["Ubreve"] = 364,
3315   ["ubreve"] = 365,
3316   ["Uring"] = 366,
3317   ["uring"] = 367,
3318   ["Udblac"] = 368,
3319   ["udblac"] = 369,
3320   ["Uogon"] = 370,
3321   ["uogon"] = 371,
3322   ["Wcirc"] = 372,
3323   ["wcirc"] = 373,
3324   ["Ycirc"] = 374,
3325   ["ycirc"] = 375,
3326   ["Yuml"] = 376,
3327   ["Zacute"] = 377,
3328   ["zacute"] = 378,
```

```
  ["Zdot"] = 379,
  ["zdot"] = 380,
  ["Zcaron"] = 381,
  ["zcaron"] = 382,
  ["fnof"] = 402,
  ["imped"] = 437,
  ["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,
```

```
3376   ["alpha"] = 945,
3377   ["beta"] = 946,
3378   ["gamma"] = 947,
3379   ["delta"] = 948,
3380   ["epsiv"] = 949,
3381   ["varepsilon"] = 949,
3382   ["epsilon"] = 949,
3383   ["zeta"] = 950,
3384   ["eta"] = 951,
3385   ["theta"] = 952,
3386   ["iota"] = 953,
3387   ["kappa"] = 954,
3388   ["lambda"] = 955,
3389   ["mu"] = 956,
3390   ["nu"] = 957,
3391   ["xi"] = 958,
3392   ["omicron"] = 959,
3393   ["pi"] = 960,
3394   ["rho"] = 961,
3395   ["sigmav"] = 962,
3396   ["varsigma"] = 962,
3397   ["sigmaf"] = 962,
3398   ["sigma"] = 963,
3399   ["tau"] = 964,
3400   ["upsi"] = 965,
3401   ["upsilon"] = 965,
3402   ["phi"] = 966,
3403   ["phiv"] = 966,
3404   ["varphi"] = 966,
3405   ["chi"] = 967,
3406   ["psi"] = 968,
3407   ["omega"] = 969,
3408   ["thetav"] = 977,
3409   ["vartheta"] = 977,
3410   ["thetasym"] = 977,
3411   ["Upsi"] = 978,
3412   ["upsih"] = 978,
3413   ["straightphi"] = 981,
3414   ["piv"] = 982,
3415   ["varpi"] = 982,
3416   ["Gammad"] = 988,
3417   ["gammad"] = 989,
3418   ["digamma"] = 989,
3419   ["kappav"] = 1008,
3420   ["varkappa"] = 1008,
3421   ["rhov"] = 1009,
3422   ["varrho"] = 1009,
```

```
3423   ["epsi"] = 1013,
3424   ["straightepsilon"] = 1013,
3425   ["bepsi"] = 1014,
3426   ["backepsilon"] = 1014,
3427   ["IOcy"] = 1025,
3428   ["DJcy"] = 1026,
3429   ["GJcy"] = 1027,
3430   ["Jukcy"] = 1028,
3431   ["DScy"] = 1029,
3432   ["Iukcy"] = 1030,
3433   ["YIcy"] = 1031,
3434   ["Jsercy"] = 1032,
3435   ["LJcy"] = 1033,
3436   ["NJcy"] = 1034,
3437   ["TSHcy"] = 1035,
3438   ["KJcy"] = 1036,
3439   ["Ubrcy"] = 1038,
3440   ["DZcy"] = 1039,
3441   ["Acy"] = 1040,
3442   ["Bcy"] = 1041,
3443   ["Vcy"] = 1042,
3444   ["Gcy"] = 1043,
3445   ["Dcy"] = 1044,
3446   ["IEcy"] = 1045,
3447   ["ZHcy"] = 1046,
3448   ["Zcy"] = 1047,
3449   ["Icy"] = 1048,
3450   ["Jcy"] = 1049,
3451   ["Kcy"] = 1050,
3452   ["Lcy"] = 1051,
3453   ["Mcy"] = 1052,
3454   ["Ncy"] = 1053,
3455   ["Ocy"] = 1054,
3456   ["Pcy"] = 1055,
3457   ["Rcy"] = 1056,
3458   ["Scy"] = 1057,
3459   ["Tcy"] = 1058,
3460   ["Ucy"] = 1059,
3461   ["Fcy"] = 1060,
3462   ["KHcy"] = 1061,
3463   ["TScy"] = 1062,
3464   ["CHcy"] = 1063,
3465   ["SHcy"] = 1064,
3466   ["SHCHcy"] = 1065,
3467   ["HARDcy"] = 1066,
3468   ["Ycy"] = 1067,
3469   ["SOFTcy"] = 1068,
```

```
3470    ["Ecy"] = 1069,
3471    ["YUcy"] = 1070,
3472    ["YAcy"] = 1071,
3473    ["acy"] = 1072,
3474    ["bcy"] = 1073,
3475    ["vcy"] = 1074,
3476    ["gcy"] = 1075,
3477    ["dcy"] = 1076,
3478    ["iecy"] = 1077,
3479    ["zhcy"] = 1078,
3480    ["zcy"] = 1079,
3481    ["icy"] = 1080,
3482    ["jcy"] = 1081,
3483    ["kcy"] = 1082,
3484    ["lcy"] = 1083,
3485    ["mcy"] = 1084,
3486    ["ncy"] = 1085,
3487    ["ocy"] = 1086,
3488    ["pcy"] = 1087,
3489    ["rcy"] = 1088,
3490    ["scy"] = 1089,
3491    ["tcy"] = 1090,
3492    ["ucy"] = 1091,
3493    ["fcy"] = 1092,
3494    ["khcy"] = 1093,
3495    ["tscy"] = 1094,
3496    ["chcy"] = 1095,
3497    ["shcy"] = 1096,
3498    ["shchcy"] = 1097,
3499    ["hardcy"] = 1098,
3500    ["ycy"] = 1099,
3501    ["softcy"] = 1100,
3502    ["ecy"] = 1101,
3503    ["yucy"] = 1102,
3504    ["yacy"] = 1103,
3505    ["iocy"] = 1105,
3506    ["djcy"] = 1106,
3507    ["gjcy"] = 1107,
3508    ["jukcy"] = 1108,
3509    ["dscy"] = 1109,
3510    ["iukcy"] = 1110,
3511    ["yicy"] = 1111,
3512    ["jsercy"] = 1112,
3513    ["ljcy"] = 1113,
3514    ["njcy"] = 1114,
3515    ["tshcy"] = 1115,
3516    ["kjcy"] = 1116,
```

```
3517   ["ubrcy"] = 1118,
3518   ["dzcy"] = 1119,
3519   ["ensp"] = 8194,
3520   ["emsp"] = 8195,
3521   ["emsp13"] = 8196,
3522   ["emsp14"] = 8197,
3523   ["numsp"] = 8199,
3524   ["puncsp"] = 8200,
3525   ["thinsp"] = 8201,
3526   ["ThinSpace"] = 8201,
3527   ["hairsp"] = 8202,
3528   ["VeryThinSpace"] = 8202,
3529   ["ZeroWidthSpace"] = 8203,
3530   ["NegativeVeryThinSpace"] = 8203,
3531   ["NegativeThinSpace"] = 8203,
3532   ["NegativeMediumSpace"] = 8203,
3533   ["NegativeThickSpace"] = 8203,
3534   ["zwnj"] = 8204,
3535   ["zwj"] = 8205,
3536   ["lrm"] = 8206,
3537   ["rlm"] = 8207,
3538   ["hyphen"] = 8208,
3539   ["dash"] = 8208,
3540   ["ndash"] = 8211,
3541   ["mdash"] = 8212,
3542   ["horbar"] = 8213,
3543   ["Verbar"] = 8214,
3544   ["Vert"] = 8214,
3545   ["lsquo"] = 8216,
3546   ["OpenCurlyQuote"] = 8216,
3547   ["rsquo"] = 8217,
3548   ["rsquor"] = 8217,
3549   ["CloseCurlyQuote"] = 8217,
3550   ["lsquor"] = 8218,
3551   ["sbquo"] = 8218,
3552   ["ldquo"] = 8220,
3553   ["OpenCurlyDoubleQuote"] = 8220,
3554   ["rdquo"] = 8221,
3555   ["rdquor"] = 8221,
3556   ["CloseCurlyDoubleQuote"] = 8221,
3557   ["ldquor"] = 8222,
3558   ["bdquo"] = 8222,
3559   ["dagger"] = 8224,
3560   ["Dagger"] = 8225,
3561   ["ddagger"] = 8225,
3562   ["bull"] = 8226,
3563   ["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,
["Laplacetrf"] = 8466,
["ell"] = 8467,
["Nopf"] = 8469,
["naturals"] = 8469,
["numero"] = 8470,
["copysr"] = 8471,
["weierp"] = 8472,
["wp"] = 8472,
["Popf"] = 8473,
["primes"] = 8473,
["rationals"] = 8474,
["Qopf"] = 8474,
["Rscr"] = 8475,
["realine"] = 8475,
["real"] = 8476,
["Re"] = 8476,
["realpart"] = 8476,
["Rfr"] = 8476,
["reals"] = 8477,
["Ropf"] = 8477,
["rx"] = 8478,
["trade"] = 8482,
["TRADE"] = 8482,
["integers"] = 8484,
["Zopf"] = 8484,
["ohm"] = 8486,
["mho"] = 8487,
["Zfr"] = 8488,
["zeetrf"] = 8488,
["iiota"] = 8489,
["angst"] = 8491,
["bernou"] = 8492,
["Bernoullis"] = 8492,
["Bscr"] = 8492,
["Cfr"] = 8493,
["Cayleys"] = 8493,
["escr"] = 8495,
["Escr"] = 8496,
["expectation"] = 8496,
["Fscr"] = 8497,
```

```
3658   ["Fouriertrf"] = 8497,
3659   ["phmmat"] = 8499,
3660   ["Mellintrf"] = 8499,
3661   ["Mscr"] = 8499,
3662   ["order"] = 8500,
3663   ["orderof"] = 8500,
3664   ["oscr"] = 8500,
3665   ["alefsym"] = 8501,
3666   ["aleph"] = 8501,
3667   ["beth"] = 8502,
3668   ["gimel"] = 8503,
3669   ["daleth"] = 8504,
3670   ["CapitalDifferentialD"] = 8517,
3671   ["DD"] = 8517,
3672   ["DifferentialD"] = 8518,
3673   ["dd"] = 8518,
3674   ["ExponentialE"] = 8519,
3675   ["exponentiale"] = 8519,
3676   ["ee"] = 8519,
3677   ["ImaginaryI"] = 8520,
3678   ["ii"] = 8520,
3679   ["frac13"] = 8531,
3680   ["frac23"] = 8532,
3681   ["frac15"] = 8533,
3682   ["frac25"] = 8534,
3683   ["frac35"] = 8535,
3684   ["frac45"] = 8536,
3685   ["frac16"] = 8537,
3686   ["frac56"] = 8538,
3687   ["frac18"] = 8539,
3688   ["frac38"] = 8540,
3689   ["frac58"] = 8541,
3690   ["frac78"] = 8542,
3691   ["larr"] = 8592,
3692   ["leftarrow"] = 8592,
3693   ["LeftArrow"] = 8592,
3694   ["slarr"] = 8592,
3695   ["ShortLeftArrow"] = 8592,
3696   ["uarr"] = 8593,
3697   ["uparrow"] = 8593,
3698   ["UpArrow"] = 8593,
3699   ["ShortUpArrow"] = 8593,
3700   ["rarr"] = 8594,
3701   ["rightarrow"] = 8594,
3702   ["RightArrow"] = 8594,
3703   ["srarr"] = 8594,
3704   ["ShortRightArrow"] = 8594,
```

```
3705   ["darr"] = 8595,
3706   ["downarrow"] = 8595,
3707   ["DownArrow"] = 8595,
3708   ["ShortDownArrow"] = 8595,
3709   ["harr"] = 8596,
3710   ["leftrightarrow"] = 8596,
3711   ["LeftRightArrow"] = 8596,
3712   ["varr"] = 8597,
3713   ["updownarrow"] = 8597,
3714   ["UpDownArrow"] = 8597,
3715   ["nwarr"] = 8598,
3716   ["UpperLeftArrow"] = 8598,
3717   ["nwarrow"] = 8598,
3718   ["nearr"] = 8599,
3719   ["UpperRightArrow"] = 8599,
3720   ["nearrow"] = 8599,
3721   ["searr"] = 8600,
3722   ["searrow"] = 8600,
3723   ["LowerRightArrow"] = 8600,
3724   ["swarr"] = 8601,
3725   ["swarrow"] = 8601,
3726   ["LowerLeftArrow"] = 8601,
3727   ["nlarr"] = 8602,
3728   ["nleftarrow"] = 8602,
3729   ["nrarr"] = 8603,
3730   ["nrightarrow"] = 8603,
3731   ["rarrw"] = 8605,
3732   ["rightsquigarrow"] = 8605,
3733   ["Larr"] = 8606,
3734   ["twoheadleftarrow"] = 8606,
3735   ["Uarr"] = 8607,
3736   ["Rarr"] = 8608,
3737   ["twoheadrightarrow"] = 8608,
3738   ["Darr"] = 8609,
3739   ["larrtl"] = 8610,
3740   ["leftarrowtail"] = 8610,
3741   ["rarrtl"] = 8611,
3742   ["rightarrowtail"] = 8611,
3743   ["LeftTeeArrow"] = 8612,
3744   ["mapstoleft"] = 8612,
3745   ["UpTeeArrow"] = 8613,
3746   ["mapstoup"] = 8613,
3747   ["map"] = 8614,
3748   ["RightTeeArrow"] = 8614,
3749   ["mapsto"] = 8614,
3750   ["DownTeeArrow"] = 8615,
3751   ["mapstodown"] = 8615,
```

```
3752   ["larrhk"] = 8617,
3753   ["hookleftarrow"] = 8617,
3754   ["rarrhk"] = 8618,
3755   ["hookrightarrow"] = 8618,
3756   ["larrlp"] = 8619,
3757   ["looparrowleft"] = 8619,
3758   ["rarrlp"] = 8620,
3759   ["looparrowright"] = 8620,
3760   ["harrw"] = 8621,
3761   ["leftrightsquigarrow"] = 8621,
3762   ["nharr"] = 8622,
3763   ["nleftrightarrow"] = 8622,
3764   ["lsh"] = 8624,
3765   ["Lsh"] = 8624,
3766   ["rsh"] = 8625,
3767   ["Rsh"] = 8625,
3768   ["ldsh"] = 8626,
3769   ["rdsh"] = 8627,
3770   ["crarr"] = 8629,
3771   ["cularr"] = 8630,
3772   ["curvearrowleft"] = 8630,
3773   ["curarr"] = 8631,
3774   ["curvearrowright"] = 8631,
3775   ["olarr"] = 8634,
3776   ["circlearrowleft"] = 8634,
3777   ["orarr"] = 8635,
3778   ["circlearrowright"] = 8635,
3779   ["lharu"] = 8636,
3780   ["LeftVector"] = 8636,
3781   ["leftharpoonup"] = 8636,
3782   ["lhard"] = 8637,
3783   ["leftharpoondown"] = 8637,
3784   ["DownLeftVector"] = 8637,
3785   ["uharr"] = 8638,
3786   ["upharpoonright"] = 8638,
3787   ["RightUpVector"] = 8638,
3788   ["uharl"] = 8639,
3789   ["upharpoonleft"] = 8639,
3790   ["LeftUpVector"] = 8639,
3791   ["rharu"] = 8640,
3792   ["RightVector"] = 8640,
3793   ["rightharpoonup"] = 8640,
3794   ["rhard"] = 8641,
3795   ["rightharpoondown"] = 8641,
3796   ["DownRightVector"] = 8641,
3797   ["dharr"] = 8642,
3798   ["RightDownVector"] = 8642,
```

```
["downharpoonright"] = 8642,
["dharl"] = 8643,
["LeftDownVector"] = 8643,
["downharpoonleft"] = 8643,
["rlarr"] = 8644,
["rightleftarrows"] = 8644,
["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,
```

```
3846   ["DoubleLeftRightArrow"] = 8660,
3847   ["iff"] = 8660,
3848   ["vArr"] = 8661,
3849   ["Updownarrow"] = 8661,
3850   ["DoubleUpDownArrow"] = 8661,
3851   ["nwArr"] = 8662,
3852   ["neArr"] = 8663,
3853   ["seArr"] = 8664,
3854   ["swArr"] = 8665,
3855   ["lAarr"] = 8666,
3856   ["Lleftarrow"] = 8666,
3857   ["rAarr"] = 8667,
3858   ["Rrightarrow"] = 8667,
3859   ["zigrarr"] = 8669,
3860   ["larrb"] = 8676,
3861   ["LeftArrowBar"] = 8676,
3862   ["rarrb"] = 8677,
3863   ["RightArrowBar"] = 8677,
3864   ["duarr"] = 8693,
3865   ["DownArrowUpArrow"] = 8693,
3866   ["loarr"] = 8701,
3867   ["roarr"] = 8702,
3868   ["hoarr"] = 8703,
3869   ["forall"] = 8704,
3870   ["ForAll"] = 8704,
3871   ["comp"] = 8705,
3872   ["complement"] = 8705,
3873   ["part"] = 8706,
3874   ["PartialD"] = 8706,
3875   ["exist"] = 8707,
3876   ["Exists"] = 8707,
3877   ["nexist"] = 8708,
3878   ["NotExists"] = 8708,
3879   ["nexists"] = 8708,
3880   ["empty"] = 8709,
3881   ["emptyset"] = 8709,
3882   ["emptyv"] = 8709,
3883   ["varnothing"] = 8709,
3884   ["nabla"] = 8711,
3885   ["Del"] = 8711,
3886   ["isin"] = 8712,
3887   ["isinv"] = 8712,
3888   ["Element"] = 8712,
3889   ["in"] = 8712,
3890   ["notin"] = 8713,
3891   ["NotElement"] = 8713,
3892   ["notinva"] = 8713,
```

```
["niv"] = 8715,
["ReverseElement"] = 8715,
["ni"] = 8715,
["SuchThat"] = 8715,
["notni"] = 8716,
["notniva"] = 8716,
["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,
["mstpos"] = 8766,
["acd"] = 8767,
["wreath"] = 8768,
["VerticalTilde"] = 8768,
["wr"] = 8768,
["nsim"] = 8769,
["NotTilde"] = 8769,
["esim"] = 8770,
["EqualTilde"] = 8770,
["eqsim"] = 8770,
["sime"] = 8771,
["TildeEqual"] = 8771,
["simeq"] = 8771,
["nsime"] = 8772,
["nsimeq"] = 8772,
["NotTildeEqual"] = 8772,
["cong"] = 8773,
["TildeFullEqual"] = 8773,
["simne"] = 8774,
["ncong"] = 8775,
["NotTildeFullEqual"] = 8775,
["asymp"] = 8776,
["ap"] = 8776,
["TildeTilde"] = 8776,
["approx"] = 8776,
["thkap"] = 8776,
["thickapprox"] = 8776,
["nap"] = 8777,
["NotTildeTilde"] = 8777,
["napprox"] = 8777,
["ape"] = 8778,
["approxeq"] = 8778,
["apid"] = 8779,
["bcong"] = 8780,
["backcong"] = 8780,
["asympeq"] = 8781,
["CupCap"] = 8781,
["bump"] = 8782,
["HumpDownHump"] = 8782,
["Bumpeq"] = 8782,
```

```
4034    ["bumpe"] = 8783,
4035    ["HumpEqual"] = 8783,
4036    ["bumpeq"] = 8783,
4037    ["esdot"] = 8784,
4038    ["DotEqual"] = 8784,
4039    ["doteq"] = 8784,
4040    ["eDot"] = 8785,
4041    ["doteqdot"] = 8785,
4042    ["efDot"] = 8786,
4043    ["fallingdotseq"] = 8786,
4044    ["erDot"] = 8787,
4045    ["risingdotseq"] = 8787,
4046    ["colone"] = 8788,
4047    ["coloneq"] = 8788,
4048    ["Assign"] = 8788,
4049    ["ecolon"] = 8789,
4050    ["eqcolon"] = 8789,
4051    ["ecir"] = 8790,
4052    ["eqcirc"] = 8790,
4053    ["cire"] = 8791,
4054    ["circeq"] = 8791,
4055    ["wedgeq"] = 8793,
4056    ["veeeq"] = 8794,
4057    ["trie"] = 8796,
4058    ["triangleq"] = 8796,
4059    ["equest"] = 8799,
4060    ["questeq"] = 8799,
4061    ["ne"] = 8800,
4062    ["NotEqual"] = 8800,
4063    ["equiv"] = 8801,
4064    ["Congruent"] = 8801,
4065    ["nequiv"] = 8802,
4066    ["NotCongruent"] = 8802,
4067    ["le"] = 8804,
4068    ["leq"] = 8804,
4069    ["ge"] = 8805,
4070    ["GreaterEqual"] = 8805,
4071    ["geq"] = 8805,
4072    ["lE"] = 8806,
4073    ["LessFullEqual"] = 8806,
4074    ["leqq"] = 8806,
4075    ["gE"] = 8807,
4076    ["GreaterFullEqual"] = 8807,
4077    ["geqq"] = 8807,
4078    ["lnE"] = 8808,
4079    ["lneqq"] = 8808,
4080    ["gnE"] = 8809,
```

```
4081    ["gneqq"] = 8809,
4082    ["Lt"] = 8810,
4083    ["NestedLessLess"] = 8810,
4084    ["ll"] = 8810,
4085    ["Gt"] = 8811,
4086    ["NestedGreaterGreater"] = 8811,
4087    ["gg"] = 8811,
4088    ["twixt"] = 8812,
4089    ["between"] = 8812,
4090    ["NotCupCap"] = 8813,
4091    ["nlt"] = 8814,
4092    ["NotLess"] = 8814,
4093    ["nless"] = 8814,
4094    ["ngt"] = 8815,
4095    ["NotGreater"] = 8815,
4096    ["ngtr"] = 8815,
4097    ["nle"] = 8816,
4098    ["NotLessEqual"] = 8816,
4099    ["nleq"] = 8816,
4100    ["nge"] = 8817,
4101    ["NotGreaterEqual"] = 8817,
4102    ["ngeq"] = 8817,
4103    ["lsim"] = 8818,
4104    ["LessTilde"] = 8818,
4105    ["lesssim"] = 8818,
4106    ["gsim"] = 8819,
4107    ["gtrsim"] = 8819,
4108    ["GreaterTilde"] = 8819,
4109    ["nlsim"] = 8820,
4110    ["NotLessTilde"] = 8820,
4111    ["ngsim"] = 8821,
4112    ["NotGreaterTilde"] = 8821,
4113    ["lg"] = 8822,
4114    ["lessgtr"] = 8822,
4115    ["LessGreater"] = 8822,
4116    ["gl"] = 8823,
4117    ["gtrless"] = 8823,
4118    ["GreaterLess"] = 8823,
4119    ["ntlg"] = 8824,
4120    ["NotLessGreater"] = 8824,
4121    ["ntgl"] = 8825,
4122    ["NotGreaterLess"] = 8825,
4123    ["pr"] = 8826,
4124    ["Precedes"] = 8826,
4125    ["prec"] = 8826,
4126    ["sc"] = 8827,
4127    ["Succeeds"] = 8827,
```

```
4128  ["succ"] = 8827,
4129  ["prcue"] = 8828,
4130  ["PrecedesSlantEqual"] = 8828,
4131  ["preccurlyeq"] = 8828,
4132  ["sccue"] = 8829,
4133  ["SucceedsSlantEqual"] = 8829,
4134  ["succcurlyeq"] = 8829,
4135  ["prsim"] = 8830,
4136  ["precsim"] = 8830,
4137  ["PrecedesTilde"] = 8830,
4138  ["scsim"] = 8831,
4139  ["succsim"] = 8831,
4140  ["SucceedsTilde"] = 8831,
4141  ["npr"] = 8832,
4142  ["nprec"] = 8832,
4143  ["NotPrecedes"] = 8832,
4144  ["nsc"] = 8833,
4145  ["nsucc"] = 8833,
4146  ["NotSucceeds"] = 8833,
4147  ["sub"] = 8834,
4148  ["subset"] = 8834,
4149  ["sup"] = 8835,
4150  ["supset"] = 8835,
4151  ["Superset"] = 8835,
4152  ["nsub"] = 8836,
4153  ["nsup"] = 8837,
4154  ["sube"] = 8838,
4155  ["SubsetEqual"] = 8838,
4156  ["subseteq"] = 8838,
4157  ["supe"] = 8839,
4158  ["supseteq"] = 8839,
4159  ["SupersetEqual"] = 8839,
4160  ["nsube"] = 8840,
4161  ["nsubseteq"] = 8840,
4162  ["NotSubsetEqual"] = 8840,
4163  ["nsupe"] = 8841,
4164  ["nsupseteq"] = 8841,
4165  ["NotSupersetEqual"] = 8841,
4166  ["subne"] = 8842,
4167  ["subsetneq"] = 8842,
4168  ["supne"] = 8843,
4169  ["supsetneq"] = 8843,
4170  ["cupdot"] = 8845,
4171  ["uplus"] = 8846,
4172  ["UnionPlus"] = 8846,
4173  ["sqsub"] = 8847,
4174  ["SquareSubset"] = 8847,
```

```
4175    ["sqsubset"] = 8847,
4176    ["sqsup"] = 8848,
4177    ["SquareSuperset"] = 8848,
4178    ["sqsupset"] = 8848,
4179    ["sqsube"] = 8849,
4180    ["SquareSubsetEqual"] = 8849,
4181    ["sqsubseteq"] = 8849,
4182    ["sqsupe"] = 8850,
4183    ["SquareSupersetEqual"] = 8850,
4184    ["sqsupseteq"] = 8850,
4185    ["sqcap"] = 8851,
4186    ["SquareIntersection"] = 8851,
4187    ["sqcup"] = 8852,
4188    ["SquareUnion"] = 8852,
4189    ["oplus"] = 8853,
4190    ["CirclePlus"] = 8853,
4191    ["ominus"] = 8854,
4192    ["CircleMinus"] = 8854,
4193    ["otimes"] = 8855,
4194    ["CircleTimes"] = 8855,
4195    ["osol"] = 8856,
4196    ["odot"] = 8857,
4197    ["CircleDot"] = 8857,
4198    ["ocir"] = 8858,
4199    ["circledcirc"] = 8858,
4200    ["oast"] = 8859,
4201    ["circledast"] = 8859,
4202    ["odash"] = 8861,
4203    ["circleddash"] = 8861,
4204    ["plusb"] = 8862,
4205    ["boxplus"] = 8862,
4206    ["minusb"] = 8863,
4207    ["boxminus"] = 8863,
4208    ["timesb"] = 8864,
4209    ["boxtimes"] = 8864,
4210    ["sdotb"] = 8865,
4211    ["dotsquare"] = 8865,
4212    ["vdash"] = 8866,
4213    ["RightTee"] = 8866,
4214    ["dashv"] = 8867,
4215    ["LeftTee"] = 8867,
4216    ["top"] = 8868,
4217    ["DownTee"] = 8868,
4218    ["bottom"] = 8869,
4219    ["bot"] = 8869,
4220    ["perp"] = 8869,
4221    ["UpTee"] = 8869,
```

```
4222  ["models"] = 8871,
4223  ["vDash"] = 8872,
4224  ["DoubleRightTee"] = 8872,
4225  ["Vdash"] = 8873,
4226  ["Vvdash"] = 8874,
4227  ["VDash"] = 8875,
4228  ["nvdash"] = 8876,
4229  ["nvDash"] = 8877,
4230  ["nVdash"] = 8878,
4231  ["nVDash"] = 8879,
4232  ["prurel"] = 8880,
4233  ["vltri"] = 8882,
4234  ["vartriangleleft"] = 8882,
4235  ["LeftTriangle"] = 8882,
4236  ["vrtri"] = 8883,
4237  ["vartriangleright"] = 8883,
4238  ["RightTriangle"] = 8883,
4239  ["ltrie"] = 8884,
4240  ["trianglelefteq"] = 8884,
4241  ["LeftTriangleEqual"] = 8884,
4242  ["rtrie"] = 8885,
4243  ["trianglerighteq"] = 8885,
4244  ["RightTriangleEqual"] = 8885,
4245  ["origof"] = 8886,
4246  ["imof"] = 8887,
4247  ["mumap"] = 8888,
4248  ["multimap"] = 8888,
4249  ["hercon"] = 8889,
4250  ["intcal"] = 8890,
4251  ["intercal"] = 8890,
4252  ["veebar"] = 8891,
4253  ["barvee"] = 8893,
4254  ["angrtvb"] = 8894,
4255  ["lrtri"] = 8895,
4256  ["xwedge"] = 8896,
4257  ["Wedge"] = 8896,
4258  ["bigwedge"] = 8896,
4259  ["xvee"] = 8897,
4260  ["Vee"] = 8897,
4261  ["bigvee"] = 8897,
4262  ["xcap"] = 8898,
4263  ["Intersection"] = 8898,
4264  ["bigcap"] = 8898,
4265  ["xcup"] = 8899,
4266  ["Union"] = 8899,
4267  ["bigcup"] = 8899,
4268  ["diam"] = 8900,
```

```
4269  ["diamond"] = 8900,
4270  ["Diamond"] = 8900,
4271  ["sdot"] = 8901,
4272  ["sstarf"] = 8902,
4273  ["Star"] = 8902,
4274  ["divonx"] = 8903,
4275  ["divideontimes"] = 8903,
4276  ["bowtie"] = 8904,
4277  ["ltimes"] = 8905,
4278  ["rtimes"] = 8906,
4279  ["lthree"] = 8907,
4280  ["leftthreetimes"] = 8907,
4281  ["rthree"] = 8908,
4282  ["rightthreetimes"] = 8908,
4283  ["bsime"] = 8909,
4284  ["backsimeq"] = 8909,
4285  ["cuvee"] = 8910,
4286  ["curlyvee"] = 8910,
4287  ["cuwed"] = 8911,
4288  ["curlywedge"] = 8911,
4289  ["Sub"] = 8912,
4290  ["Subset"] = 8912,
4291  ["Sup"] = 8913,
4292  ["Supset"] = 8913,
4293  ["Cap"] = 8914,
4294  ["Cup"] = 8915,
4295  ["fork"] = 8916,
4296  ["pitchfork"] = 8916,
4297  ["epar"] = 8917,
4298  ["ltdot"] = 8918,
4299  ["lessdot"] = 8918,
4300  ["gtdot"] = 8919,
4301  ["gtrdot"] = 8919,
4302  ["Ll"] = 8920,
4303  ["Gg"] = 8921,
4304  ["ggg"] = 8921,
4305  ["leg"] = 8922,
4306  ["LessEqualGreater"] = 8922,
4307  ["lesseqgtr"] = 8922,
4308  ["gel"] = 8923,
4309  ["gtreqless"] = 8923,
4310  ["GreaterEqualLess"] = 8923,
4311  ["cuepr"] = 8926,
4312  ["curlyeqprec"] = 8926,
4313  ["cuesc"] = 8927,
4314  ["curlyeqsucc"] = 8927,
4315  ["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,
```

```
4363   ["rceil"] = 8969,
4364   ["RightCeiling"] = 8969,
4365   ["lfloor"] = 8970,
4366   ["LeftFloor"] = 8970,
4367   ["rfloor"] = 8971,
4368   ["RightFloor"] = 8971,
4369   ["drcrop"] = 8972,
4370   ["dlcrop"] = 8973,
4371   ["urcrop"] = 8974,
4372   ["ulcrop"] = 8975,
4373   ["bnot"] = 8976,
4374   ["profline"] = 8978,
4375   ["profsurf"] = 8979,
4376   ["telrec"] = 8981,
4377   ["target"] = 8982,
4378   ["ulcorn"] = 8988,
4379   ["ulcorner"] = 8988,
4380   ["urcorn"] = 8989,
4381   ["urcorner"] = 8989,
4382   ["dlcorn"] = 8990,
4383   ["llcorner"] = 8990,
4384   ["drcorn"] = 8991,
4385   ["lrcorner"] = 8991,
4386   ["frown"] = 8994,
4387   ["sfrown"] = 8994,
4388   ["smile"] = 8995,
4389   ["ssmile"] = 8995,
4390   ["cylcty"] = 9005,
4391   ["profalar"] = 9006,
4392   ["topbot"] = 9014,
4393   ["ovbar"] = 9021,
4394   ["solbar"] = 9023,
4395   ["angzarr"] = 9084,
4396   ["lmoust"] = 9136,
4397   ["lmoustache"] = 9136,
4398   ["rmoust"] = 9137,
4399   ["rmoustache"] = 9137,
4400   ["tbrk"] = 9140,
4401   ["OverBracket"] = 9140,
4402   ["bbrk"] = 9141,
4403   ["UnderBracket"] = 9141,
4404   ["bbrktbrk"] = 9142,
4405   ["OverParenthesis"] = 9180,
4406   ["UnderParenthesis"] = 9181,
4407   ["OverBrace"] = 9182,
4408   ["UnderBrace"] = 9183,
4409   ["trpezium"] = 9186,
```

```
4410    ["elinters"] = 9191,
4411    ["blank"] = 9251,
4412    ["oS"] = 9416,
4413    ["circledS"] = 9416,
4414    ["boxh"] = 9472,
4415    ["HorizontalLine"] = 9472,
4416    ["boxv"] = 9474,
4417    ["boxdr"] = 9484,
4418    ["boxdl"] = 9488,
4419    ["boxur"] = 9492,
4420    ["boxul"] = 9496,
4421    ["boxvr"] = 9500,
4422    ["boxvl"] = 9508,
4423    ["boxhd"] = 9516,
4424    ["boxhu"] = 9524,
4425    ["boxvh"] = 9532,
4426    ["boxH"] = 9552,
4427    ["boxV"] = 9553,
4428    ["boxdR"] = 9554,
4429    ["boxDr"] = 9555,
4430    ["boxDR"] = 9556,
4431    ["boxdL"] = 9557,
4432    ["boxDl"] = 9558,
4433    ["boxDL"] = 9559,
4434    ["boxuR"] = 9560,
4435    ["boxUr"] = 9561,
4436    ["boxUR"] = 9562,
4437    ["boxuL"] = 9563,
4438    ["boxUl"] = 9564,
4439    ["boxUL"] = 9565,
4440    ["boxvR"] = 9566,
4441    ["boxVr"] = 9567,
4442    ["boxVR"] = 9568,
4443    ["boxvL"] = 9569,
4444    ["boxVl"] = 9570,
4445    ["boxVL"] = 9571,
4446    ["boxHd"] = 9572,
4447    ["boxhD"] = 9573,
4448    ["boxHD"] = 9574,
4449    ["boxHu"] = 9575,
4450    ["boxhU"] = 9576,
4451    ["boxHU"] = 9577,
4452    ["boxvH"] = 9578,
4453    ["boxVh"] = 9579,
4454    ["boxVH"] = 9580,
4455    ["uhblk"] = 9600,
4456    ["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,
```

```
4504   ["bigstar"] = 9733,
4505   ["star"] = 9734,
4506   ["phone"] = 9742,
4507   ["female"] = 9792,
4508   ["male"] = 9794,
4509   ["spades"] = 9824,
4510   ["spadesuit"] = 9824,
4511   ["clubs"] = 9827,
4512   ["clubsuit"] = 9827,
4513   ["hearts"] = 9829,
4514   ["heartsuit"] = 9829,
4515   ["diams"] = 9830,
4516   ["diamondsuit"] = 9830,
4517   ["sung"] = 9834,
4518   ["flat"] = 9837,
4519   ["natur"] = 9838,
4520   ["natural"] = 9838,
4521   ["sharp"] = 9839,
4522   ["check"] = 10003,
4523   ["checkmark"] = 10003,
4524   ["cross"] = 10007,
4525   ["malt"] = 10016,
4526   ["maltese"] = 10016,
4527   ["sext"] = 10038,
4528   ["VerticalSeparator"] = 10072,
4529   ["lbbrk"] = 10098,
4530   ["rbbrk"] = 10099,
4531   ["lobrk"] = 10214,
4532   ["LeftDoubleBracket"] = 10214,
4533   ["robrk"] = 10215,
4534   ["RightDoubleBracket"] = 10215,
4535   ["lang"] = 10216,
4536   ["LeftAngleBracket"] = 10216,
4537   ["langle"] = 10216,
4538   ["rang"] = 10217,
4539   ["RightAngleBracket"] = 10217,
4540   ["rangle"] = 10217,
4541   ["Lang"] = 10218,
4542   ["Rang"] = 10219,
4543   ["loang"] = 10220,
4544   ["roang"] = 10221,
4545   ["xlarr"] = 10229,
4546   ["longleftarrow"] = 10229,
4547   ["LongLeftArrow"] = 10229,
4548   ["xrarr"] = 10230,
4549   ["longrightarrow"] = 10230,
4550   ["LongRightArrow"] = 10230,
```

```
  ["xharr"] = 10231,
  ["longleftrightarrow"] = 10231,
  ["LongLeftRightArrow"] = 10231,
  ["xlArr"] = 10232,
  ["Longleftarrow"] = 10232,
  ["DoubleLongLeftArrow"] = 10232,
  ["xrArr"] = 10233,
  ["Longrightarrow"] = 10233,
  ["DoubleLongRightArrow"] = 10233,
  ["xhArr"] = 10234,
  ["Longleftrightarrow"] = 10234,
  ["DoubleLongLeftRightArrow"] = 10234,
  ["xmap"] = 10236,
  ["longmapsto"] = 10236,
  ["dzigrarr"] = 10239,
  ["nvlArr"] = 10498,
  ["nvrArr"] = 10499,
  ["nvHarr"] = 10500,
  ["Map"] = 10501,
  ["lbarr"] = 10508,
  ["rbarr"] = 10509,
  ["bkarow"] = 10509,
  ["lBarr"] = 10510,
  ["rBarr"] = 10511,
  ["dbkarow"] = 10511,
  ["RBarr"] = 10512,
  ["drbkarow"] = 10512,
  ["DDotrahd"] = 10513,
  ["UpArrowBar"] = 10514,
  ["DownArrowBar"] = 10515,
  ["Rarrtl"] = 10518,
  ["latail"] = 10521,
  ["ratail"] = 10522,
  ["lAtail"] = 10523,
  ["rAtail"] = 10524,
  ["larrfs"] = 10525,
  ["rarrfs"] = 10526,
  ["larrbfs"] = 10527,
  ["rarrbfs"] = 10528,
  ["nwarhk"] = 10531,
  ["nearhk"] = 10532,
  ["searhk"] = 10533,
  ["hksearow"] = 10533,
  ["swarhk"] = 10534,
  ["hkswarow"] = 10534,
  ["nwnear"] = 10535,
  ["nesear"] = 10536,
```

```
4598    ["toea"] = 10536,
4599    ["seswar"] = 10537,
4600    ["tosa"] = 10537,
4601    ["swnwar"] = 10538,
4602    ["rarrc"] = 10547,
4603    ["cudarrr"] = 10549,
4604    ["ldca"] = 10550,
4605    ["rdca"] = 10551,
4606    ["cudarrl"] = 10552,
4607    ["larrpl"] = 10553,
4608    ["curarrm"] = 10556,
4609    ["cularrp"] = 10557,
4610    ["rarrpl"] = 10565,
4611    ["harrcir"] = 10568,
4612    ["Uarrocir"] = 10569,
4613    ["lurdshar"] = 10570,
4614    ["ldrushar"] = 10571,
4615    ["LeftRightVector"] = 10574,
4616    ["RightUpDownVector"] = 10575,
4617    ["DownLeftRightVector"] = 10576,
4618    ["LeftUpDownVector"] = 10577,
4619    ["LeftVectorBar"] = 10578,
4620    ["RightVectorBar"] = 10579,
4621    ["RightUpVectorBar"] = 10580,
4622    ["RightDownVectorBar"] = 10581,
4623    ["DownLeftVectorBar"] = 10582,
4624    ["DownRightVectorBar"] = 10583,
4625    ["LeftUpVectorBar"] = 10584,
4626    ["LeftDownVectorBar"] = 10585,
4627    ["LeftTeeVector"] = 10586,
4628    ["RightTeeVector"] = 10587,
4629    ["RightUpTeeVector"] = 10588,
4630    ["RightDownTeeVector"] = 10589,
4631    ["DownLeftTeeVector"] = 10590,
4632    ["DownRightTeeVector"] = 10591,
4633    ["LeftUpTeeVector"] = 10592,
4634    ["LeftDownTeeVector"] = 10593,
4635    ["lHar"] = 10594,
4636    ["uHar"] = 10595,
4637    ["rHar"] = 10596,
4638    ["dHar"] = 10597,
4639    ["luruhar"] = 10598,
4640    ["ldrdhar"] = 10599,
4641    ["ruluhar"] = 10600,
4642    ["rdldhar"] = 10601,
4643    ["lharul"] = 10602,
4644    ["llhard"] = 10603,
```

```
4645    ["rharul"] = 10604,
4646    ["lrhard"] = 10605,
4647    ["udhar"] = 10606,
4648    ["UpEquilibrium"] = 10606,
4649    ["duhar"] = 10607,
4650    ["ReverseUpEquilibrium"] = 10607,
4651    ["RoundImplies"] = 10608,
4652    ["erarr"] = 10609,
4653    ["simrarr"] = 10610,
4654    ["larrsim"] = 10611,
4655    ["rarrsim"] = 10612,
4656    ["rarrap"] = 10613,
4657    ["ltlarr"] = 10614,
4658    ["gtrarr"] = 10616,
4659    ["subrarr"] = 10617,
4660    ["suplarr"] = 10619,
4661    ["lfisht"] = 10620,
4662    ["rfisht"] = 10621,
4663    ["ufisht"] = 10622,
4664    ["dfisht"] = 10623,
4665    ["lopar"] = 10629,
4666    ["ropar"] = 10630,
4667    ["lbrke"] = 10635,
4668    ["rbrke"] = 10636,
4669    ["lbrkslu"] = 10637,
4670    ["rbrksld"] = 10638,
4671    ["lbrksld"] = 10639,
4672    ["rbrkslu"] = 10640,
4673    ["langd"] = 10641,
4674    ["rangd"] = 10642,
4675    ["lparlt"] = 10643,
4676    ["rpargt"] = 10644,
4677    ["gtlPar"] = 10645,
4678    ["ltrPar"] = 10646,
4679    ["vzigzag"] = 10650,
4680    ["vangrt"] = 10652,
4681    ["angrtvbd"] = 10653,
4682    ["ange"] = 10660,
4683    ["range"] = 10661,
4684    ["dwangle"] = 10662,
4685    ["uwangle"] = 10663,
4686    ["angmsdaa"] = 10664,
4687    ["angmsdab"] = 10665,
4688    ["angmsdac"] = 10666,
4689    ["angmsdad"] = 10667,
4690    ["angmsdae"] = 10668,
4691    ["angmsdaf"] = 10669,
```

```
4692  ["angmsdag"] = 10670,
4693  ["angmsdah"] = 10671,
4694  ["bemptyv"] = 10672,
4695  ["demptyv"] = 10673,
4696  ["cemptyv"] = 10674,
4697  ["raemptyv"] = 10675,
4698  ["laemptyv"] = 10676,
4699  ["ohbar"] = 10677,
4700  ["omid"] = 10678,
4701  ["opar"] = 10679,
4702  ["operp"] = 10681,
4703  ["olcross"] = 10683,
4704  ["odsold"] = 10684,
4705  ["olcir"] = 10686,
4706  ["ofcir"] = 10687,
4707  ["olt"] = 10688,
4708  ["ogt"] = 10689,
4709  ["cirscir"] = 10690,
4710  ["cirE"] = 10691,
4711  ["solb"] = 10692,
4712  ["bsolb"] = 10693,
4713  ["boxbox"] = 10697,
4714  ["trisb"] = 10701,
4715  ["rtriltri"] = 10702,
4716  ["LeftTriangleBar"] = 10703,
4717  ["RightTriangleBar"] = 10704,
4718  ["race"] = 10714,
4719  ["iinfin"] = 10716,
4720  ["infintie"] = 10717,
4721  ["nvinfin"] = 10718,
4722  ["eparsl"] = 10723,
4723  ["smeparsl"] = 10724,
4724  ["eqvparsl"] = 10725,
4725  ["lozf"] = 10731,
4726  ["blacklozenge"] = 10731,
4727  ["RuleDelayed"] = 10740,
4728  ["dsol"] = 10742,
4729  ["xodot"] = 10752,
4730  ["bigodot"] = 10752,
4731  ["xoplus"] = 10753,
4732  ["bigoplus"] = 10753,
4733  ["xotime"] = 10754,
4734  ["bigotimes"] = 10754,
4735  ["xuplus"] = 10756,
4736  ["biguplus"] = 10756,
4737  ["xsqcup"] = 10758,
4738  ["bigsqcup"] = 10758,
```

```
4739  ["qint"] = 10764,
4740  ["iiiint"] = 10764,
4741  ["fpartint"] = 10765,
4742  ["cirfnint"] = 10768,
4743  ["awint"] = 10769,
4744  ["rppolint"] = 10770,
4745  ["scpolint"] = 10771,
4746  ["npolint"] = 10772,
4747  ["pointint"] = 10773,
4748  ["quatint"] = 10774,
4749  ["intlarhk"] = 10775,
4750  ["pluscir"] = 10786,
4751  ["plusacir"] = 10787,
4752  ["simplus"] = 10788,
4753  ["plusdu"] = 10789,
4754  ["plussim"] = 10790,
4755  ["plustwo"] = 10791,
4756  ["mcomma"] = 10793,
4757  ["minusdu"] = 10794,
4758  ["loplus"] = 10797,
4759  ["roplus"] = 10798,
4760  ["Cross"] = 10799,
4761  ["timesd"] = 10800,
4762  ["timesbar"] = 10801,
4763  ["smashp"] = 10803,
4764  ["lotimes"] = 10804,
4765  ["rotimes"] = 10805,
4766  ["otimesas"] = 10806,
4767  ["Otimes"] = 10807,
4768  ["odiv"] = 10808,
4769  ["triplus"] = 10809,
4770  ["triminus"] = 10810,
4771  ["tritime"] = 10811,
4772  ["iprod"] = 10812,
4773  ["intprod"] = 10812,
4774  ["amalg"] = 10815,
4775  ["capdot"] = 10816,
4776  ["ncup"] = 10818,
4777  ["ncap"] = 10819,
4778  ["capand"] = 10820,
4779  ["cupor"] = 10821,
4780  ["cupcap"] = 10822,
4781  ["capcup"] = 10823,
4782  ["cupbrcap"] = 10824,
4783  ["capbrcup"] = 10825,
4784  ["cupcup"] = 10826,
4785  ["capcap"] = 10827,
```

```
4786  ["ccups"] = 10828,
4787  ["ccaps"] = 10829,
4788  ["ccupssm"] = 10832,
4789  ["And"] = 10835,
4790  ["Or"] = 10836,
4791  ["andand"] = 10837,
4792  ["oror"] = 10838,
4793  ["orslope"] = 10839,
4794  ["andslope"] = 10840,
4795  ["andv"] = 10842,
4796  ["orv"] = 10843,
4797  ["andd"] = 10844,
4798  ["ord"] = 10845,
4799  ["wedbar"] = 10847,
4800  ["sdote"] = 10854,
4801  ["simdot"] = 10858,
4802  ["congdot"] = 10861,
4803  ["easter"] = 10862,
4804  ["apacir"] = 10863,
4805  ["apE"] = 10864,
4806  ["eplus"] = 10865,
4807  ["pluse"] = 10866,
4808  ["Esim"] = 10867,
4809  ["Colone"] = 10868,
4810  ["Equal"] = 10869,
4811  ["eDDot"] = 10871,
4812  ["ddotseq"] = 10871,
4813  ["equivDD"] = 10872,
4814  ["ltcir"] = 10873,
4815  ["gtcir"] = 10874,
4816  ["ltquest"] = 10875,
4817  ["gtquest"] = 10876,
4818  ["les"] = 10877,
4819  ["LessSlantEqual"] = 10877,
4820  ["leqslant"] = 10877,
4821  ["ges"] = 10878,
4822  ["GreaterSlantEqual"] = 10878,
4823  ["geqslant"] = 10878,
4824  ["lesdot"] = 10879,
4825  ["gesdot"] = 10880,
4826  ["lesdoto"] = 10881,
4827  ["gesdoto"] = 10882,
4828  ["lesdotor"] = 10883,
4829  ["gesdotol"] = 10884,
4830  ["lap"] = 10885,
4831  ["lessapprox"] = 10885,
4832  ["gap"] = 10886,
```

```
4833    ["gtrapprox"] = 10886,
4834    ["lne"] = 10887,
4835    ["lneq"] = 10887,
4836    ["gne"] = 10888,
4837    ["gneq"] = 10888,
4838    ["lnap"] = 10889,
4839    ["lnapprox"] = 10889,
4840    ["gnap"] = 10890,
4841    ["gnapprox"] = 10890,
4842    ["lEg"] = 10891,
4843    ["lesseqqgtr"] = 10891,
4844    ["gEl"] = 10892,
4845    ["gtreqqless"] = 10892,
4846    ["lsime"] = 10893,
4847    ["gsime"] = 10894,
4848    ["lsimg"] = 10895,
4849    ["gsiml"] = 10896,
4850    ["lgE"] = 10897,
4851    ["glE"] = 10898,
4852    ["lesges"] = 10899,
4853    ["gesles"] = 10900,
4854    ["els"] = 10901,
4855    ["eqslantless"] = 10901,
4856    ["egs"] = 10902,
4857    ["eqslantgtr"] = 10902,
4858    ["elsdot"] = 10903,
4859    ["egsdot"] = 10904,
4860    ["el"] = 10905,
4861    ["eg"] = 10906,
4862    ["siml"] = 10909,
4863    ["simg"] = 10910,
4864    ["simlE"] = 10911,
4865    ["simgE"] = 10912,
4866    ["LessLess"] = 10913,
4867    ["GreaterGreater"] = 10914,
4868    ["glj"] = 10916,
4869    ["gla"] = 10917,
4870    ["ltcc"] = 10918,
4871    ["gtcc"] = 10919,
4872    ["lescc"] = 10920,
4873    ["gescc"] = 10921,
4874    ["smt"] = 10922,
4875    ["lat"] = 10923,
4876    ["smte"] = 10924,
4877    ["late"] = 10925,
4878    ["bumpE"] = 10926,
4879    ["pre"] = 10927,
```

```
4880    ["preceq"] = 10927,
4881    ["PrecedesEqual"] = 10927,
4882    ["sce"] = 10928,
4883    ["succeq"] = 10928,
4884    ["SucceedsEqual"] = 10928,
4885    ["prE"] = 10931,
4886    ["scE"] = 10932,
4887    ["prnE"] = 10933,
4888    ["precneqq"] = 10933,
4889    ["scnE"] = 10934,
4890    ["succneqq"] = 10934,
4891    ["prap"] = 10935,
4892    ["precapprox"] = 10935,
4893    ["scap"] = 10936,
4894    ["succapprox"] = 10936,
4895    ["prnap"] = 10937,
4896    ["precnapprox"] = 10937,
4897    ["scnap"] = 10938,
4898    ["succnapprox"] = 10938,
4899    ["Pr"] = 10939,
4900    ["Sc"] = 10940,
4901    ["subdot"] = 10941,
4902    ["supdot"] = 10942,
4903    ["subplus"] = 10943,
4904    ["supplus"] = 10944,
4905    ["submult"] = 10945,
4906    ["supmult"] = 10946,
4907    ["subedot"] = 10947,
4908    ["supedot"] = 10948,
4909    ["subE"] = 10949,
4910    ["subseteqq"] = 10949,
4911    ["supE"] = 10950,
4912    ["supseteqq"] = 10950,
4913    ["subsim"] = 10951,
4914    ["supsim"] = 10952,
4915    ["subnE"] = 10955,
4916    ["subsetneqq"] = 10955,
4917    ["supnE"] = 10956,
4918    ["supsetneqq"] = 10956,
4919    ["csub"] = 10959,
4920    ["csup"] = 10960,
4921    ["csube"] = 10961,
4922    ["csupe"] = 10962,
4923    ["subsup"] = 10963,
4924    ["supsub"] = 10964,
4925    ["subsub"] = 10965,
4926    ["supsup"] = 10966,
```

```
4927   ["suphsub"] = 10967,
4928   ["supdsub"] = 10968,
4929   ["forkv"] = 10969,
4930   ["topfork"] = 10970,
4931   ["mlcp"] = 10971,
4932   ["Dashv"] = 10980,
4933   ["DoubleLeftTee"] = 10980,
4934   ["Vdashl"] = 10982,
4935   ["Barv"] = 10983,
4936   ["vBar"] = 10984,
4937   ["vBarv"] = 10985,
4938   ["Vbar"] = 10987,
4939   ["Not"] = 10988,
4940   ["bNot"] = 10989,
4941   ["rnmid"] = 10990,
4942   ["cirmid"] = 10991,
4943   ["midcir"] = 10992,
4944   ["topcir"] = 10993,
4945   ["nhpar"] = 10994,
4946   ["parsim"] = 10995,
4947   ["parsl"] = 11005,
4948   ["fflig"] = 64256,
4949   ["filig"] = 64257,
4950   ["fllig"] = 64258,
4951   ["ffilig"] = 64259,
4952   ["ffllig"] = 64260,
4953   ["Ascr"] = 119964,
4954   ["Cscr"] = 119966,
4955   ["Dscr"] = 119967,
4956   ["Gscr"] = 119970,
4957   ["Jscr"] = 119973,
4958   ["Kscr"] = 119974,
4959   ["Nscr"] = 119977,
4960   ["Oscr"] = 119978,
4961   ["Pscr"] = 119979,
4962   ["Qscr"] = 119980,
4963   ["Sscr"] = 119982,
4964   ["Tscr"] = 119983,
4965   ["Uscr"] = 119984,
4966   ["Vscr"] = 119985,
4967   ["Wscr"] = 119986,
4968   ["Xscr"] = 119987,
4969   ["Yscr"] = 119988,
4970   ["Zscr"] = 119989,
4971   ["ascr"] = 119990,
4972   ["bscr"] = 119991,
4973   ["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,
```

```
5021    ["gfr"] = 120100,
5022    ["hfr"] = 120101,
5023    ["ifr"] = 120102,
5024    ["jfr"] = 120103,
5025    ["kfr"] = 120104,
5026    ["lfr"] = 120105,
5027    ["mfr"] = 120106,
5028    ["nfr"] = 120107,
5029    ["ofr"] = 120108,
5030    ["pfr"] = 120109,
5031    ["qfr"] = 120110,
5032    ["rfr"] = 120111,
5033    ["sfr"] = 120112,
5034    ["tfr"] = 120113,
5035    ["ufr"] = 120114,
5036    ["vfr"] = 120115,
5037    ["wfr"] = 120116,
5038    ["xfr"] = 120117,
5039    ["yfr"] = 120118,
5040    ["zfr"] = 120119,
5041    ["Aopf"] = 120120,
5042    ["Bopf"] = 120121,
5043    ["Dopf"] = 120123,
5044    ["Eopf"] = 120124,
5045    ["Fopf"] = 120125,
5046    ["Gopf"] = 120126,
5047    ["Iopf"] = 120128,
5048    ["Jopf"] = 120129,
5049    ["Kopf"] = 120130,
5050    ["Lopf"] = 120131,
5051    ["Mopf"] = 120132,
5052    ["Oopf"] = 120134,
5053    ["Sopf"] = 120138,
5054    ["Topf"] = 120139,
5055    ["Uopf"] = 120140,
5056    ["Vopf"] = 120141,
5057    ["Wopf"] = 120142,
5058    ["Xopf"] = 120143,
5059    ["Yopf"] = 120144,
5060    ["aopf"] = 120146,
5061    ["bopf"] = 120147,
5062    ["copf"] = 120148,
5063    ["dopf"] = 120149,
5064    ["eopf"] = 120150,
5065    ["fopf"] = 120151,
5066    ["gopf"] = 120152,
5067    ["hopf"] = 120153,
```

```
  ["iopf"] = 120154,
  ["jopf"] = 120155,
  ["kopf"] = 120156,
  ["lopf"] = 120157,
  ["mopf"] = 120158,
  ["nopf"] = 120159,
  ["oopf"] = 120160,
  ["popf"] = 120161,
  ["qopf"] = 120162,
  ["ropf"] = 120163,
  ["sopf"] = 120164,
  ["topf"] = 120165,
  ["uopf"] = 120166,
  ["vopf"] = 120167,
  ["wopf"] = 120168,
  ["xopf"] = 120169,
  ["yopf"] = 120170,
  ["zopf"] = 120171,
}
```

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

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

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

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

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

```
function entities.char_entity(s)
  local n = character_entities[s]
  if n == nil then
    return "&" .. s .. ";"
  end
  return unicode.utf8.char(n)
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.

```
5100 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.

```
5101 function M.writer.new(options)
5102   local self = {}
```

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

```
5103   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.

```
5104   local slice_specifiers = {}
5105   for specifier in options.slice:gmatch("[^%s]+") do
5106     table.insert(slice_specifiers, specifier)
5107   end
5108
5109   if #slice_specifiers == 2 then
5110     self.slice_begin, self.slice_end = table.unpack(slice_specifiers)
5111     local slice_begin_type = self.slice_begin:sub(1, 1)
5112     if slice_begin_type ~= "^" and slice_begin_type ~= "$" then
5113       self.slice_begin = "^" .. self.slice_begin
5114     end
5115     local slice_end_type = self.slice_end:sub(1, 1)
5116     if slice_end_type ~= "^" and slice_end_type ~= "$" then
5117       self.slice_end = "$" .. self.slice_end
5118     end
5119   elseif #slice_specifiers == 1 then
5120     self.slice_begin = "^" .. slice_specifiers[1]
5121     self.slice_end = "$" .. slice_specifiers[1]
5122   end
5123
5124   self.slice_begin_type = self.slice_begin:sub(1, 1)
5125   self.slice_begin_identifier = self.slice_begin:sub(2) or ""
```

```
5126    self.slice_end_type = self.slice_end:sub(1, 1)
5127    self.slice_end_identifier = self.slice_end:sub(2) or ""
5128
5129    if self.slice_begin == "^" and self.slice_end ~= "^" then
5130      self.is_writing = true
5131    else
5132      self.is_writing = false
5133    end
```

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

```
5134    self.suffix = ".tex"
```

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

```
5135    self.space = " "
```

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

```
5136    self.nbsp = "\\markdownRendererNbsp{}"
```

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

```
5137    function self.plain(s)
5138      return s
5139    end
```

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

```
5140    function self.paragraph(s)
5141      if not self.is_writing then return "" end
5142      return s
5143    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.

```
5144    function self.pack(name)
5145      return [[\input{]] .. name .. [[}\relax]]
5146    end
```

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

```
5147    function self.interblocksep()
5148      if not self.is_writing then return "" end
5149      return "\\markdownRendererInterblockSeparator\n{}"
5150    end
```

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

```
5151    self.hard_line_break = "\\markdownRendererHardLineBreak\n{}"
```

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

```
5152    self.ellipsis = "\\markdownRendererEllipsis{}"
```

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

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

```
self.escaped_uri_chars = {
  ["{"] = "\\markdownRendererLeftBrace{}",
  ["}"] = "\\markdownRendererRightBrace{}",
  ["\\"] = "\\markdownRendererBackslash{}",
}
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_typographic_text`, `writer->escape_programmatic_text`, and `writer->escape_minimal` escaper functions.

```
  local escape_typographic_text = util.escaper(
    self.escaped_chars, self.escaped_strings)
  local escape_programmatic_text = util.escaper(
    self.escaped_uri_chars, self.escaped_minimal_strings)
  local escape_minimal = util.escaper(
    {}, self.escaped_minimal_strings)
```

Define the following semantic aliases for the escaper functions:

- `writer->escape` transforms a text string that should always be made printable.
- `writer->string` transforms a text string that should be made printable only when the `hybrid` Lua option is disabled. When `hybrid` is enabled, the text string should be kept as-is.
- `writer->math` transforms a math span.
- `writer->identifier` transforms an input programmatic identifier.
- `writer->uri` transforms an input URI.

```
  self.escape = escape_typographic_text
  self.math = escape_minimal
  if options.hybrid then
    self.identifier = escape_minimal
    self.string = escape_minimal
    self.uri = escape_minimal
  else
    self.identifier = escape_programmatic_text
    self.string = escape_typographic_text
    self.uri = escape_programmatic_text
  end
```

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

```
  function self.code(s, attributes)
    local buf = {}
    if attributes ~= nil then
      table.insert(buf,
                   "\\markdownRendererCodeSpanAttributeContextBegin\n")
      table.insert(buf, self.attributes(attributes))
    end
    table.insert(buf,
                 {"\\markdownRendererCodeSpan{", self.escape(s), "}"})
    if attributes ~= nil then
      table.insert(buf,
                   "\n\\markdownRendererCodeSpanAttributeContextEnd ")
    end
    return buf
  end
```

Define `writer->link` as a function that will transform an input hyperlink to the output format, where `lab` corresponds to the label, `src` to URI, `tit` to the title of the link, and `attributes` to optional attributes.

```
function self.link(lab, src, tit, attributes)
  local buf = {}
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererLinkAttributeContextBegin\n")
    table.insert(buf, self.attributes(attributes))
  end
  table.insert(buf, {"\\markdownRendererLink{",lab,"}",
                     "{",self.escape(src),"}",
                     "{",self.uri(src),"}",
                     "{",self.string(tit or ""),"}"})
  if attributes ~= nil then
    table.insert(buf,
                 "\n\\markdownRendererLinkAttributeContextEnd ")
  end
  return buf
end
```

Define `writer->image` as a function that will transform an input image to the output format, where `lab` corresponds to the label, `src` to the URL, `tit` to the title of the image, and `attributes` to optional attributes.

```
function self.image(lab, src, tit, attributes)
  local buf = {}
  if attributes ~= nil then
    table.insert(buf,
                 "\\markdownRendererImageAttributeContextBegin\n")
    table.insert(buf, self.attributes(attributes))
  end
  table.insert(buf, {"\\markdownRendererImage{",lab,"}",
                     "{",self.string(src),"}",
                     "{",self.uri(src),"}",
                     "{",self.string(tit or ""),"}"})
  if attributes ~= nil then
    table.insert(buf,
                 "\n\\markdownRendererImageAttributeContextEnd ")
  end
  return buf
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.

```
function self.bulletlist(items,tight)
  if not self.is_writing then return "" end
```

```
5253      local buffer = {}
5254      for _,item in ipairs(items) do
5255        buffer[#buffer + 1] = self.bulletitem(item)
5256      end
5257      local contents = util.intersperse(buffer,"\n")
5258      if tight and options.tightLists then
5259        return {"\\markdownRendererUlBeginTight\n",contents,
5260          "\n\\markdownRendererUlEndTight "}
5261      else
5262        return {"\\markdownRendererUlBegin\n",contents,
5263          "\n\\markdownRendererUlEnd "}
5264      end
5265    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.

```
5266    function self.bulletitem(s)
5267      return {"\\markdownRendererUlItem ",s,
5268              "\\markdownRendererUlItemEnd "}
5269    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.

```
5270    function self.orderedlist(items,tight,startnum)
5271      if not self.is_writing then return "" end
5272      local buffer = {}
5273      local num = startnum
5274      for _,item in ipairs(items) do
5275        buffer[#buffer + 1] = self.ordereditem(item,num)
5276        if num ~= nil then
5277          num = num + 1
5278        end
5279      end
5280      local contents = util.intersperse(buffer,"\n")
5281      if tight and options.tightLists then
5282        return {"\\markdownRendererOlBeginTight\n",contents,
5283                "\n\\markdownRendererOlEndTight "}
5284      else
5285        return {"\\markdownRendererOlBegin\n",contents,
5286                "\n\\markdownRendererOlEnd "}
5287      end
5288    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.

```
function self.ordereditem(s,num)
  if num ~= nil then
    return {"\\markdownRendererOlItemWithNumber{",num,"}",s,
            "\\markdownRendererOlItemEnd "}
  else
    return {"\\markdownRendererOlItem ",s,
            "\\markdownRendererOlItemEnd "}
  end
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.

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

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

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

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

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

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

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

```
function self.tickbox(f)
  if f == 1.0 then
    return "☒ "
  elseif f == 0.0 then
    return "▫ "
  else
    return "⊡ "
  end
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 buf = {"\\markdownRendererDocumentBegin\n", d}

  -- pop all attributes
  table.insert(buf, self.pop_attributes())

  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 seen = {}
  local key, value
  for i = 1, #attr do
    if seen[attr[i]] ~= nil then
      goto continue  -- prevent duplicate attributes
    else
      seen[attr[i]] = true
    end
    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
    ::continue::
  end

  return buf
end
```

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

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

Define `writer->push_attributes` and `writer->pop_attributes` as functions that will add a new set of active block-level attributes or remove the most current attributes from `writer->active_attributes`.

```
local function apply_attributes()
  local buf = {}
  for i = 1, #self.active_attributes do
    local start_output = self.active_attributes[i][3]
    if start_output ~= nil then
      table.insert(buf, start_output)
    end
  end
  return buf
```

```
5388  end
5389
5390  local function tear_down_attributes()
5391    local buf = {}
5392    for i = #self.active_attributes, 1, -1 do
5393      local end_output = self.active_attributes[i][4]
5394      if end_output ~= nil then
5395        table.insert(buf, end_output)
5396      end
5397    end
5398    return buf
5399  end
```

The `writer->push_attributes` method adds `attributes` of type `attribute_type` to `writer->active_attributes`. The `start_output` string is used to construct a rope that will be returned by this function, together with output produced as a result of slicing (see `slice`). The `end_output` string is stored together with `attributes` and is used to construct the return value of the `writer->pop_attributes` method.

```
5400  function self.push_attributes(attribute_type, attributes,
5401                                start_output, end_output)
5402    -- index attributes in a hash table for easy lookup
5403    attributes = attributes or {}
5404    for i = 1, #attributes do
5405      attributes[attributes[i]] = true
5406    end
5407
5408    local buf = {}
5409    -- handle slicing
5410    if attributes["#" .. self.slice_end_identifier] ~= nil and
5411       self.slice_end_type == "^" then
5412      if self.is_writing then
5413        table.insert(buf, tear_down_attributes())
5414      end
5415      self.is_writing = false
5416    end
5417    if attributes["#" .. self.slice_begin_identifier] ~= nil and
5418       self.slice_begin_type == "^" then
5419      self.is_writing = true
5420      table.insert(buf, apply_attributes())
5421      self.is_writing = true
5422    end
5423    if self.is_writing and start_output ~= nil then
5424      table.insert(buf, start_output)
5425    end
5426    table.insert(self.active_attributes,
5427                 {attribute_type, attributes,
5428                  start_output, end_output})
```

```
5429      return buf
5430    end
5431
```

The writer->pop_attributes method removes the most current of active block-level attributes from writer->active_attributes until attributes of type attribute_type have been removed. The method returns a rope constructed from the end_output string specified in the calls of writer->push_attributes that produced the most current attributes, and also from output produced as a result of slicing (see slice).

```
5432    function self.pop_attributes(attribute_type)
5433      local buf = {}
5434      -- pop attributes until we find attributes of correct type
5435      -- or until no attributes remain
5436      local current_attribute_type = false
5437      while current_attribute_type ~= attribute_type and
5438            #self.active_attributes > 0 do
5439        local attributes, _, end_output
5440        current_attribute_type, attributes, _, end_output = table.unpack(
5441          self.active_attributes[#self.active_attributes])
5442        if self.is_writing and end_output ~= nil then
5443          table.insert(buf, end_output)
5444        end
5445        table.remove(self.active_attributes, #self.active_attributes)
5446        -- handle slicing
5447        if attributes["#" .. self.slice_end_identifier] ~= nil
5448           and self.slice_end_type == "$" then
5449          if self.is_writing then
5450            table.insert(buf, tear_down_attributes())
5451          end
5452          self.is_writing = false
5453        end
5454        if attributes["#" .. self.slice_begin_identifier] ~= nil and
5455           self.slice_begin_type == "$" then
5456          self.is_writing = true
5457          table.insert(buf, apply_attributes())
5458        end
5459      end
5460      return buf
5461    end
```

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

```
5462    local current_heading_level = 0
5463    function self.heading(s, level, attributes)
5464      local buf = {}
5465
```

```
5466    -- push empty attributes for implied sections
5467    while current_heading_level < level - 1 do
5468      table.insert(buf,
5469                   self.push_attributes("heading",
5470                                        nil,
5471                                        "\\markdownRendererSectionBegin\n",
5472                                        "\n\\markdownRendererSectionEnd "))
5473      current_heading_level = current_heading_level + 1
5474    end
5475
5476    -- pop attributes for sections that have ended
5477    while current_heading_level >= level do
5478      table.insert(buf, self.pop_attributes("heading"))
5479      current_heading_level = current_heading_level - 1
5480    end
5481
5482    -- push attributes for the new section
5483    local start_output = {}
5484    local end_output = {}
5485    table.insert(start_output, "\\markdownRendererSectionBegin\n")
5486    if options.headerAttributes and attributes ~= nil and #attributes > 0 then
5487      table.insert(start_output,
5488                   "\\markdownRendererHeaderAttributeContextBegin\n")
5489      table.insert(start_output, self.attributes(attributes))
5490      table.insert(end_output,
5491                   "\n\\markdownRendererHeaderAttributeContextEnd ")
5492    end
5493    table.insert(end_output, "\n\\markdownRendererSectionEnd ")
5494
5495    table.insert(buf, self.push_attributes("heading",
5496                                           attributes,
5497                                           start_output,
5498                                           end_output))
5499    current_heading_level = current_heading_level + 1
5500    assert(current_heading_level == level)
5501
5502    -- produce the renderer
5503    local cmd
5504    level = level + options.shiftHeadings
5505    if level <= 1 then
5506      cmd = "\\markdownRendererHeadingOne"
5507    elseif level == 2 then
5508      cmd = "\\markdownRendererHeadingTwo"
5509    elseif level == 3 then
5510      cmd = "\\markdownRendererHeadingThree"
5511    elseif level == 4 then
5512      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
      self[key] = value
    end
    return previous_state
  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`.

```
  function self.defer_call(f)
    local previous_state = self.get_state()
    return function(...)
      local state = self.set_state(previous_state)
      local return_value = f(...)
      self.set_state(state)
      return return_value
    end
  end

  return self
end
```

3.1.4 Parsers

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

```
local parsers                 = {}
```

3.1.4.1 Basic Parsers

```
parsers.percent               = P("%")
parsers.at                    = P("@")
parsers.comma                 = P(",")
parsers.asterisk              = P("*")
parsers.dash                  = P("-")
parsers.plus                  = P("+")
parsers.underscore            = P("_")
parsers.period                = P(".")
parsers.hash                  = P("#")
parsers.dollar                = P("$")
parsers.ampersand             = P("&")
parsers.backtick              = P("`")
parsers.less                  = P("<")
parsers.more                  = P(">")
parsers.space                 = P(" ")
parsers.squote                = P("'")
parsers.dquote                = P('"')
parsers.lparent               = P("(")
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")
                                     * 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 spac
                                 + #(parsers.newline)
                                 * Cb("backslashes")
                                 * C(parsers.newline))

parsers.chunk                  = parsers.line * (parsers.optionallyindentedline
```

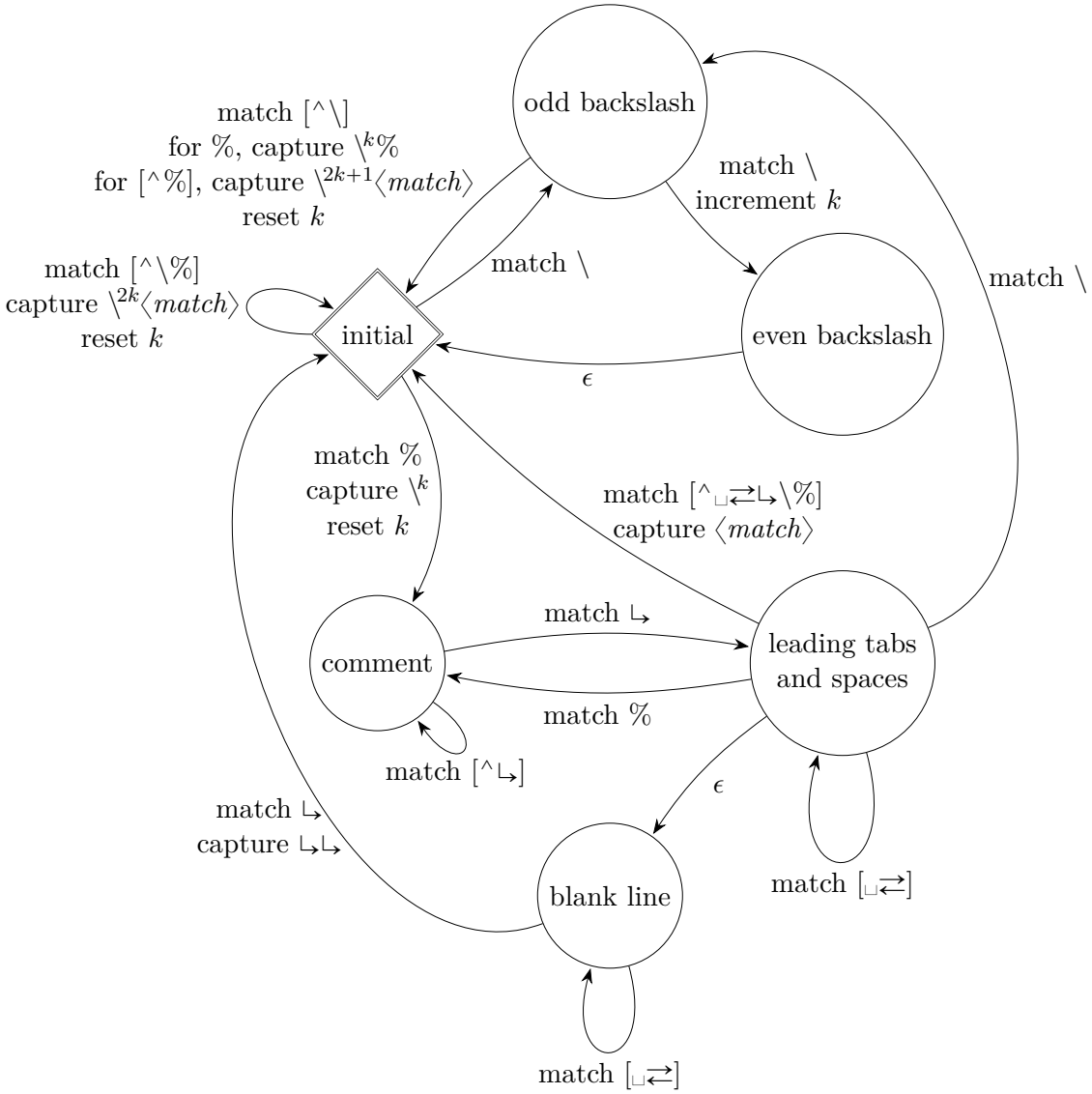

Figure 6: A pushdown automaton that recognizes TEX comments

```
                                                - 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.rbra
                                   - 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

parsers.closeticks  = parsers.space^-1
                    * Cmt(C(parsers.backtick^1)
                        * Cb("ticks"), captures_equal_length)

parsers.intickschar = (parsers.any - S(" \n\r`"))
                    + (parsers.newline * -parsers.blankline)
                    + (parsers.space - parsers.closeticks)
                    + (parsers.backtick^1 - parsers.closeticks)

parsers.inticks     = parsers.openticks * parsers.space^-1
                    * C(parsers.intickschar^0) * parsers.closeticks
```

3.1.4.4 Parsers Used for Markdown Tags and Links

```
parsers.leader      = parsers.space^-3

-- content in balanced brackets, parentheses, or quotes:
parsers.bracketed   = P{ parsers.lbracket
                       * (( parsers.backslash / "" * parsers.rbracket
```

```
                          + parsers.any - (parsers.lbracket
                                          + parsers.rbracket
                                          + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.rbracket }

parsers.inparens    = P{ parsers.lparent
                       * ((parsers.anyescaped - (parsers.lparent
                                                 + parsers.rparent
                                                 + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.rparent }

parsers.squoted     = P{ parsers.squote * parsers.alphanumeric
                       * ((parsers.anyescaped - (parsers.squote
                                                 + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.squote }

parsers.dquoted     = P{ parsers.dquote * parsers.alphanumeric
                       * ((parsers.anyescaped - (parsers.dquote
                                                 + parsers.blankline^2)
                          ) + V(1))^0
                       * parsers.dquote }

-- bracketed tag for markdown links, allowing nested brackets:
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
```

```
5811
5812 parsers.title_d     = parsers.dquote * Cs(((parsers.anyescaped-parsers.dquote)
5813                                          + parsers.dquoted)^0)
5814                                       * parsers.dquote
5815
5816 parsers.title_p     = parsers.lparent
5817                     * Cs((parsers.inparens + (parsers.anyescaped-parsers.rparent))^0)
5818                     * parsers.rparent
5819
5820 parsers.title       = parsers.title_d + parsers.title_s + parsers.title_p
5821
5822 parsers.optionaltitle
5823                     = parsers.spnl * parsers.title * parsers.spacechar^0
5824                     + Cc("")
5825
5826 parsers.indirect_link
5827                     = parsers.tag
5828                     * ( C(parsers.spnl) * parsers.tag
5829                       + Cc(nil) * Cc(nil)  -- always produce exactly two captures
5830                       )
5831
5832 parsers.indirect_image
5833                     = parsers.exclamation * parsers.indirect_link
```

3.1.4.5 Parsers Used for HTML

```
5834 -- case-insensitive match (we assume s is lowercase). must be single byte encoding
5835 parsers.keyword_exact = function(s)
5836   local parser = P(0)
5837   for i=1,#s do
5838     local c = s:sub(i,i)
5839     local m = c .. upper(c)
5840     parser = parser * S(m)
5841   end
5842   return parser
5843 end
5844
5845 parsers.block_keyword =
5846     parsers.keyword_exact("address") + parsers.keyword_exact("blockquote") +
5847     parsers.keyword_exact("center") + parsers.keyword_exact("del") +
5848     parsers.keyword_exact("dir") + parsers.keyword_exact("div") +
5849     parsers.keyword_exact("p") + parsers.keyword_exact("pre") +
5850     parsers.keyword_exact("li") + parsers.keyword_exact("ol") +
5851     parsers.keyword_exact("ul") + parsers.keyword_exact("dl") +
5852     parsers.keyword_exact("dd") + parsers.keyword_exact("form") +
5853     parsers.keyword_exact("fieldset") + parsers.keyword_exact("isindex") +
5854     parsers.keyword_exact("ins") + parsers.keyword_exact("menu") +
```

```
    parsers.keyword_exact("noframes") + parsers.keyword_exact("frameset") +
    parsers.keyword_exact("h1") + parsers.keyword_exact("h2") +
    parsers.keyword_exact("h3") + parsers.keyword_exact("h4") +
    parsers.keyword_exact("h5") + parsers.keyword_exact("h6") +
    parsers.keyword_exact("hr") + parsers.keyword_exact("script") +
    parsers.keyword_exact("noscript") + parsers.keyword_exact("table") +
    parsers.keyword_exact("tbody") + parsers.keyword_exact("tfoot") +
    parsers.keyword_exact("thead") + parsers.keyword_exact("th") +
    parsers.keyword_exact("td") + parsers.keyword_exact("tr")

-- There is no reason to support bad html, so we expect quoted attributes
parsers.htmlattributevalue
                          = parsers.squote * (parsers.any - (parsers.blankline
                                                           + parsers.squote))^0
                                           * parsers.squote
                          + parsers.dquote * (parsers.any - (parsers.blankline
                                                           + parsers.dquote))^0
                                           * parsers.dquote

parsers.htmlattribute     = parsers.spacing^1
                          * (parsers.alphanumeric + S("_-"))^1
                          * parsers.sp * parsers.equal * parsers.sp
                          * parsers.htmlattributevalue

parsers.htmlcomment       = P("<!--")
                          * parsers.optionalspace
                          * Cs((parsers.any - parsers.optionalspace * P("-->"))^0)
                          * parsers.optionalspace
                          * P("-->")

parsers.htmlinstruction   = P("<?") * (parsers.any - P("?>"))^0 * P("?>")

parsers.openelt_any = parsers.less * parsers.keyword * parsers.htmlattribute^0
                    * parsers.sp * parsers.more

parsers.openelt_exact = function(s)
  return parsers.less * parsers.sp * parsers.keyword_exact(s)
       * parsers.htmlattribute^0 * parsers.sp * parsers.more
end

parsers.openelt_block = parsers.sp * parsers.block_keyword
                      * parsers.htmlattribute^0 * parsers.sp * parsers.more

parsers.closeelt_any = parsers.less * parsers.sp * parsers.slash
                     * parsers.keyword * parsers.sp * parsers.more

parsers.closeelt_exact = function(s)
```

```
  return parsers.less * parsers.sp * parsers.slash * parsers.keyword_exact(s)
       * parsers.sp * parsers.more
end

parsers.emptyelt_any = parsers.less * parsers.sp * parsers.keyword
                     * parsers.htmlattribute^0 * parsers.sp * parsers.slash
                     * parsers.more

parsers.emptyelt_block = parsers.less * parsers.sp * parsers.block_keyword
                       * parsers.htmlattribute^0 * parsers.sp * parsers.slash
                       * parsers.more

parsers.displaytext = (parsers.any - parsers.less)^1

-- return content between two matched HTML tags
parsers.in_matched = function(s)
  return { parsers.openelt_exact(s)
         * (V(1) + parsers.displaytext
           + (parsers.less - parsers.closeelt_exact(s)))^0
         * parsers.closeelt_exact(s) }
end

local function parse_matched_tags(s,pos)
  local t = string.lower(lpeg.match(C(parsers.keyword),s,pos))
  return lpeg.match(parsers.in_matched(t),s,pos-1)
end

parsers.in_matched_block_tags = parsers.less
                              * Cmt(#parsers.openelt_block, parse_matched_tags)

```

3.1.4.6 Parsers Used for HTML Entities

```
parsers.hexentity = parsers.ampersand * parsers.hash * S("Xx")
                  * C(parsers.hexdigit^1) * parsers.semicolon
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 Link Reference Definitions

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

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

parsers.auto_link_url = parsers.less
                      * C( parsers.alphanumeric^1 * P("://")
                         * parsers.urlchar^1)
                      * parsers.more

parsers.auto_link_email
                      = parsers.less
                      * C((parsers.alphanumeric + S("-._+"))^1
                      * P("@") * parsers.urlchar^1)
                      * parsers.more

parsers.auto_link_relative_reference
                      = parsers.less
                      * C(parsers.urlchar^1)
                      * 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*⟩.

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

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

```
  self.writer = writer
  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.

```
  self.parsers = {}
  (function(parsers)
    setmetatable(self.parsers, {
      __index = function (_, key)
        return parsers[key]
      end
    })
  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.

```
  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.

```
  function self.normalize_tag(tag)
```

```
6001      tag = util.rope_to_string(tag)
6002      tag = tag:gsub("[ \n\r\t]+", " ")
6003      tag = tag:gsub("^ ", ""):gsub(" $", "")
6004      tag = uni_case.casefold(tag, true, false)
6005      return tag
6006    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.

```
6007    local function iterlines(s, f)
6008      local rope = lpeg.match(Ct((parsers.line / f)^1), s)
6009      return util.rope_to_string(rope)
6010    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.

```
6011    if options.preserveTabs then
6012      self.expandtabs = function(s) return s end
6013    else
6014      self.expandtabs = function(s)
6015                          if s:find("\t") then
6016                            return iterlines(s, util.expand_tabs_in_line)
6017                          else
6018                            return s
6019                          end
6020                        end
6021    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.

```
6022    self.parser_functions = {}
6023    self.create_parser = function(name, grammar, toplevel)
6024      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.

```
6025        if toplevel and options.stripIndent then
6026            local min_prefix_length, min_prefix = nil, ''
6027            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
            is_shorter = is_shorter or prefix_length < min_prefix_length
            if is_shorter then
                min_prefix_length, min_prefix = prefix_length, prefix
            end
            return line
        end)
        str = str:gsub('^' .. min_prefix, '')
    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.

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

self.create_parser("parse_blocks",
                   function()
                     return parsers.blocks
                   end, true)

self.create_parser("parse_blocks_nested",
                   function()
                     return parsers.blocks_nested
                   end, false)

self.create_parser("parse_inlines",
                   function()
                     return parsers.inlines
                   end, false)

```

```
6071   self.create_parser("parse_inlines_no_link",
6072                      function()
6073                        return parsers.inlines_no_link
6074                      end, false)
6075
6076   self.create_parser("parse_inlines_no_inline_note",
6077                      function()
6078                        return parsers.inlines_no_inline_note
6079                      end, false)
6080
6081   self.create_parser("parse_inlines_no_html",
6082                      function()
6083                        return parsers.inlines_no_html
6084                      end, false)
6085
6086   self.create_parser("parse_inlines_nbsp",
6087                      function()
6088                        return parsers.inlines_nbsp
6089                      end, false)
```

3.1.5.3 Parsers Used for Markdown Lists (local)

```
6090   if options.hashEnumerators then
6091     parsers.dig = parsers.digit + parsers.hash
6092   else
6093     parsers.dig = parsers.digit
6094   end
6095
6096   parsers.enumerator = C(parsers.dig^3 * parsers.period) * #parsers.spacing
6097                      + C(parsers.dig^2 * parsers.period) * #parsers.spacing
6098                                          * (parsers.tab + parsers.space^1)
6099                      + C(parsers.dig * parsers.period) * #parsers.spacing
6100                                         * (parsers.tab + parsers.space^-2)
6101                      + parsers.space * C(parsers.dig^2 * parsers.period)
6102                                         * #parsers.spacing
6103                      + parsers.space * C(parsers.dig * parsers.period)
6104                                         * #parsers.spacing
6105                                         * (parsers.tab + parsers.space^-1)
6106                      + parsers.space * parsers.space * C(parsers.dig^1
6107                                         * parsers.period) * #parsers.spacing
```

3.1.5.4 Parsers Used for Blockquotes (local)

```
6108   -- strip off leading > and indents, and run through blocks
6109   parsers.blockquote_body = ((parsers.leader * parsers.more * parsers.space^-
    1)/""
6110                               * parsers.linechar^0 * parsers.newline)^1
6111                             * (-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 Link Reference Definitions (local)

```
  -- List of references defined in the document
  local references
```

The `reader->register_link` method registers a link reference, where `tag` is the link label, `url` is the link destination, `title` is the optional link title, and `attributes` are the optional attributes.

```
  function self.register_link(tag, url, title,
                              attributes)
    tag = self.normalize_tag(tag)
    references[tag] = {
      url = url,
      title = title,
      attributes = attributes,
    }
    return ""
  end
```

The `reader->lookup_reference` method looks up a reference with link label `tag`. When the reference exists the method returns a link. The attributes of a link are produced by merging the attributes of the link reference and the optional `attributes`. Otherwise, the method returns a two-tuple of `nil` and fallback text constructed from the link text `label` and the optional spaces `sps` between the link text and the link label.

```
  function self.lookup_reference(label, sps, tag,
                                 attributes)
    local tagpart
    if not tag then
      tag = label
      tagpart = ""
    elseif tag == "" then
      tag = label
      tagpart = "[]"
    else
      tagpart = {
        "[",
        self.parser_functions.parse_inlines(tag),
```

```
        "]"
      }
    end
    if sps then
      tagpart = {sps, tagpart}
    end
    tag = self.normalize_tag(tag)
    local r = references[tag]
    if r then
      local merged_attributes = {}
      for _, attribute in ipairs(r.attributes or {}) do
        table.insert(merged_attributes, attribute)
      end
      for _, attribute in ipairs(attributes or {}) do
        table.insert(merged_attributes, attribute)
      end
      if #merged_attributes == 0 then
        merged_attributes = nil
      end
      return {
        url = r.url,
        title = r.title,
        attributes = merged_attributes,
      }
    else
      return nil, {
        "[",
        self.parser_functions.parse_inlines(label),
        "]",
        tagpart
      }
    end
  end

  -- lookup link reference and return a link, if the reference is found,
  -- or a bracketed label otherwise.
  local function indirect_link(label, sps, tag)
    return writer.defer_call(function()
      local r,fallback = self.lookup_reference(label, sps, tag)
      if r then
        return writer.link(
          self.parser_functions.parse_inlines_no_link(label),
          r.url, r.title)
      else
        return fallback
      end
    end)
```

```
6192  end
6193
6194  -- lookup image reference and return an image, if the reference is found,
6195  -- or a bracketed label otherwise.
6196  local function indirect_image(label, sps, tag)
6197    return writer.defer_call(function()
6198      local r,fallback = self.lookup_reference(label, sps, tag)
6199      if r then
6200        return writer.image(writer.string(label), r.url, r.title)
6201      else
6202        return {"!", fallback}
6203      end
6204    end)
6205  end
6206
6207  parsers.direct_link_tail = parsers.spnl
6208                           * parsers.lparent
6209                           * (parsers.url + Cc(""))  -- link can be empty [foo]()
6210                           * parsers.optionaltitle
6211                           * parsers.rparent
6212
6213  parsers.direct_link = (parsers.tag / self.parser_functions.parse_inlines_no_link)
6214                      * parsers.direct_link_tail
6215
6216  parsers.direct_image = parsers.exclamation
6217                       * (parsers.tag / self.parser_functions.parse_inlines)
6218                       * parsers.direct_link_tail
```

3.1.5.6 Inline Elements (local)

```
6219  parsers.Str      = (parsers.normalchar * (parsers.normalchar + parsers.at)^0)
6220                   / writer.string
6221
6222  parsers.Symbol   = (V("SpecialChar") - parsers.tightblocksep)
6223                   / writer.string
6224
6225  parsers.Ellipsis = P("...") / writer.ellipsis
6226
6227  parsers.Smart    = parsers.Ellipsis
6228
6229  parsers.Code     = parsers.inticks / writer.code
6230
6231  if options.blankBeforeBlockquote then
6232    parsers.bqstart = parsers.fail
6233  else
6234    parsers.bqstart = parsers.more
6235  end
```

```
6236
6237  if options.blankBeforeHeading then
6238    parsers.headerstart = parsers.fail
6239  else
6240    parsers.headerstart = parsers.hash
6241                        + (parsers.line * (parsers.equal^1 + parsers.dash^1)
6242                        * parsers.optionalspace * parsers.newline)
6243  end
6244
6245  parsers.EndlineExceptions
6246                     = parsers.blankline -- paragraph break
6247                     + parsers.tightblocksep  -- nested list
6248                     + parsers.eof       -- end of document
6249                     + parsers.bqstart
6250                     + parsers.headerstart
6251
6252  parsers.Endline   = parsers.newline
6253                    * -V("EndlineExceptions")
6254                    * parsers.spacechar^0
6255                    / (options.hardLineBreaks and writer.hard_line_break
6256                                              or writer.space)
6257
6258  parsers.OptionalIndent
6259                     = parsers.spacechar^1 / writer.space
6260
6261  parsers.Space      = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6262                     + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6263                     + parsers.spacechar^1 * parsers.Endline
6264                                           * parsers.optionalspace
6265                                           / (options.hardLineBreaks
6266                                              and writer.hard_line_break
6267                                               or writer.space)
6268                     + parsers.spacechar^1 * parsers.optionalspace
6269                                           / writer.space
6270
6271  parsers.NonbreakingEndline
6272                    = parsers.newline
6273                    * -V("EndlineExceptions")
6274                    * parsers.spacechar^0
6275                    / (options.hardLineBreaks and writer.hard_line_break
6276                                              or writer.nbsp)
6277
6278  parsers.NonbreakingSpace
6279                  = parsers.spacechar^2 * parsers.Endline / writer.hard_line_break
6280                  + parsers.spacechar^1 * parsers.Endline^-1 * parsers.eof / ""
6281                  + parsers.spacechar^1 * parsers.Endline
6282                                        * parsers.optionalspace
```

```
                                        / (options.hardLineBreaks
                                           and writer.hard_line_break
                                            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
```

The `reader->auto_link_url` method produces an autolink to a URL or a relative reference in the output format, where `url` is the link destination and `attributes` are the optional attributes.

```
function self.auto_link_url(url, attributes)
  return writer.link(writer.escape(url),
                     url, nil, attributes)
end
```

The `reader->auto_link_email` method produces an autolink to an e-mail in the output format, where `email` is the email address destination and `attributes` are the optional attributes.

```
function self.auto_link_email(email, attributes)
  return writer.link(writer.escape(email),
                     "mailto:"..email,
                     nil, attributes)
end
```

```
6322  parsers.AutoLinkUrl = parsers.auto_link_url
6323                      / self.auto_link_url
6324
6325  parsers.AutoLinkEmail
6326                      = parsers.auto_link_email
6327                      / self.auto_link_email
6328
6329  parsers.AutoLinkRelativeReference
6330                      = parsers.auto_link_relative_reference
6331                      / self.auto_link_url
6332
6333  parsers.DirectLink    = parsers.direct_link
6334                        / writer.link
6335
6336  parsers.IndirectLink  = parsers.indirect_link
6337                        / indirect_link
6338
6339  -- parse a link or image (direct or indirect)
6340  parsers.Link          = parsers.DirectLink + parsers.IndirectLink
6341
6342  parsers.DirectImage   = parsers.direct_image
6343                        / writer.image
6344
6345  parsers.IndirectImage = parsers.indirect_image
6346                        / indirect_image
6347
6348  parsers.Image         = parsers.DirectImage + parsers.IndirectImage
6349
6350  -- avoid parsing long strings of * or _ as emph/strong
6351  parsers.UlOrStarLine  = parsers.asterisk^4 + parsers.underscore^4
6352                        / writer.string
6353
6354  parsers.EscapedChar   = parsers.backslash * C(parsers.escapable) / writer.string
6355
6356  parsers.InlineHtml    = parsers.emptyelt_any / writer.inline_html_tag
6357                        + (parsers.htmlcomment / self.parser_functions.parse_inlines_
6358                        / writer.inline_html_comment
6359                        + parsers.htmlinstruction
6360                        + parsers.openelt_any / writer.inline_html_tag
6361                        + parsers.closeelt_any / writer.inline_html_tag
6362
6363  parsers.HtmlEntity    = parsers.hexentity / entities.hex_entity  / writer.string
6364                        + parsers.decentity / entities.dec_entity  / writer.string
6365                        + parsers.tagentity / entities.char_entity / writer.string
```

3.1.5.7 Block Elements (local)

```
parsers.DisplayHtml  = (parsers.htmlcomment / self.parser_functions.parse_blocks_ne
                      / writer.block_html_comment
                      + 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
                      * parsers.blankline^1
                      / self.register_link

parsers.Paragraph    = parsers.nonindentspace * Ct(parsers.Inline^1)
                      * ( parsers.newline
                        * ( parsers.blankline^1
                          + #V("EndlineExceptions")
                        )
                        + 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
  end

  -- we use \001 as a separator between a tight list item and a
  -- nested list under it.
  parsers.NestedList            = Cs((parsers.optionallyindentedline
                                      - parsers.starter)^1)
                                / function(a) return "\001"..a end

  parsers.ListBlockLine         = parsers.optionallyindentedline
                                - parsers.blankline - (parsers.indent^-
1
                                                      * parsers.starter)

  parsers.ListBlock             = parsers.line * parsers.ListBlockLine^0

  parsers.ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
                                * parsers.ListBlock

  parsers.TightListItem = function(starter)
      return -parsers.ThematicBreak
             * (Cs(starter / "" * parsers.tickbox^-1 * parsers.ListBlock * parsers.Ne
1)
               / self.parser_functions.parse_blocks_nested)
             * -(parsers.blanklines * parsers.indent)
  end

  parsers.LooseListItem = function(starter)
      return -parsers.ThematicBreak
             * Cs( starter / "" * parsers.tickbox^-1 * parsers.ListBlock * Cc("\n")
               * (parsers.NestedList + parsers.ListContinuationBlock^0)
               * (parsers.blanklines / "\n\n")
               ) / self.parser_functions.parse_blocks_nested
  end

  parsers.BulletList = ( Ct(parsers.TightListItem(parsers.bullet)^1) * Cc(true)
                       * parsers.skipblanklines * -parsers.bullet
                       + Ct(parsers.LooseListItem(parsers.bullet)^1) * Cc(false)
                       * parsers.skipblanklines )
                     / writer.bulletlist

  local function ordered_list(items,tight,startnum)
    if options.startNumber then
      startnum = tonumber(startnum) or 1  -- fallback for '#'
      if startnum ~= nil then
        startnum = math.floor(startnum)
```

```
      end
    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 / ""
                       + V("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",
          [[reader->insert_pattern() was called outside an extension with ]]
          .. [[a PEG pattern instead of a rule name]])
        accountable_pattern = pattern
      end
      if pos == "instead of" then
        rule[index] = accountable_pattern
      else
        table.insert(rule, index, accountable_pattern)
      end
    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.

```
    local syntax =
      { "Blocks",

        Blocks                = V("InitializeState")
                              * ( V("ExpectedJekyllData")
                                * (V("Blank")^0 / writer.interblocksep))^-
    1
                              * V("Blank")^0
                              * V("Block")^-1
                              * ( V("Blank")^0 / writer.interblocksep
                                * V("Block"))^0
                              * V("Blank")^0 * parsers.eof,

        ExpectedJekyllData    = parsers.fail,

        Blank                 = parsers.Blank,
        Reference             = parsers.Reference,

        Blockquote            = parsers.Blockquote,
        Verbatim              = parsers.Verbatim,
        ThematicBreak         = parsers.ThematicBreak,
        BulletList            = parsers.BulletList,
        OrderedList           = parsers.OrderedList,
        Heading               = parsers.Heading,
        DisplayHtml           = parsers.DisplayHtml,
        Paragraph             = parsers.Paragraph,
        Plain                 = parsers.Plain,

        EndlineExceptions     = parsers.EndlineExceptions,
        BlockquoteExceptions  = parsers.BlockquoteExceptions,
```

```
6570
6571          Str                   = parsers.Str,
6572          Space                 = parsers.Space,
6573          OptionalIndent        = parsers.OptionalIndent,
6574          Endline               = parsers.Endline,
6575          UlOrStarLine          = parsers.UlOrStarLine,
6576          Strong                = parsers.Strong,
6577          Emph                  = parsers.Emph,
6578          Link                  = parsers.Link,
6579          Image                 = parsers.Image,
6580          Code                  = parsers.Code,
6581          AutoLinkUrl           = parsers.AutoLinkUrl,
6582          AutoLinkEmail         = parsers.AutoLinkEmail,
6583          AutoLinkRelativeReference
6584                                = parsers.AutoLinkRelativeReference,
6585          InlineHtml            = parsers.InlineHtml,
6586          HtmlEntity            = parsers.HtmlEntity,
6587          EscapedChar           = parsers.EscapedChar,
6588          Smart                 = parsers.Smart,
6589          Symbol                = parsers.Symbol,
6590          SpecialChar           = parsers.fail,
6591          InitializeState       = parsers.succeed,
6592        }
```

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`]`.

```
6593      self.update_rule = function(rule_name, get_pattern)
6594        assert(current_extension_name ~= nil)
6595        assert(syntax[rule_name] ~= nil,
6596          [[Rule ]] .. rule_name .. [[ -> ... does not exist in markdown grammar]])
6597        local previous_pattern
6598        local extension_name
6599        if walkable_syntax[rule_name] then
6600          local previous_accountable_pattern = walkable_syntax[rule_name][1]
6601          previous_pattern = previous_accountable_pattern[1]
6602          extension_name = previous_accountable_pattern[2] .. ", " .. current_extension
6603        else
6604          previous_pattern = nil
6605          extension_name = current_extension_name
6606        end
6607        local pattern
```

Instead of a function, a PEG pattern `pattern` may also be supplied with roughly the same effect as supplying the following function, which will define

walkable_syntax[left-hand side terminal symbol] unless it has been previously defined.

```
function(previous_pattern)
  assert(previous_pattern == nil)
  return pattern
end
```

```
6608        if type(get_pattern) == "function" then
6609          pattern = get_pattern(previous_pattern)
6610        else
6611          assert(previous_pattern == nil,
6612                 [[Rule ]] .. rule_name ..
6613                 [[ has already been updated by ]] .. extension_name)
6614          pattern = get_pattern
6615        end
6616        local accountable_pattern = { pattern, extension_name, rule_name }
6617        walkable_syntax[rule_name] = { accountable_pattern }
6618      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.

```
6619      local special_characters = {}
6620      self.add_special_character = function(c)
6621        table.insert(special_characters, c)
6622        syntax.SpecialChar = S(table.concat(special_characters, ""))
6623      end
6624
6625      self.add_special_character("*")
6626      self.add_special_character("[")
6627      self.add_special_character("]")
6628      self.add_special_character("<")
6629      self.add_special_character("!")
6630      self.add_special_character("\\")
```

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

```
6631      self.initialize_named_group = function(name, value)
6632        syntax.InitializeState = syntax.InitializeState
6633                               * Cg(Ct("") / value, name)
6634      end
```

Apply syntax extensions.

```
6635      for _, extension in ipairs(extensions) do
6636        current_extension_name = extension.name
6637        extension.extend_writer(writer)
```

```
6638        extension.extend_reader(self)
6639      end
6640      current_extension_name = nil
```

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

```
6641      if options.debugExtensions then
6642        local sorted_lhs = {}
6643        for lhs, _ in pairs(walkable_syntax) do
6644          table.insert(sorted_lhs, lhs)
6645        end
6646        table.sort(sorted_lhs)
6647
6648        local output_lines = {"{"}
6649        for lhs_index, lhs in ipairs(sorted_lhs) do
6650          local encoded_lhs = util.encode_json_string(lhs)
6651          table.insert(output_lines, [[    ]] ..encoded_lhs .. [[: []])
6652          local rule = walkable_syntax[lhs]
6653          for rhs_index, rhs in ipairs(rule) do
6654            local human_readable_rhs
6655            if type(rhs) == "string" then
6656              human_readable_rhs = rhs
6657            else
6658              local pattern_name
6659              if rhs[3] then
6660                pattern_name = rhs[3]
6661              else
6662                pattern_name = "Anonymous Pattern"
6663              end
6664              local extension_name = rhs[2]
6665              human_readable_rhs = pattern_name .. [[ (]] .. extension_name .. [[)]]
6666            end
6667            local encoded_rhs = util.encode_json_string(human_readable_rhs)
6668            local output_line = [[        ]] .. encoded_rhs
6669            if rhs_index < #rule then
6670              output_line = output_line .. ","
6671            end
6672            table.insert(output_lines, output_line)
6673          end
6674          local output_line = "    ]"
6675          if lhs_index < #sorted_lhs then
6676            output_line = output_line .. ","
6677          end
6678          table.insert(output_lines, output_line)
6679        end
6680        table.insert(output_lines, "}")
6681
```

```
6682        local output = table.concat(output_lines, "\n")
6683        local output_filename = options.debugExtensionsFileName
6684        local output_file = assert(io.open(output_filename, "w"),
6685          [[Could not open file "]] .. output_filename .. [[" for writing]])
6686        assert(output_file:write(output))
6687        assert(output_file:close())
6688      end
```

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

```
6689      walkable_syntax["IndentedInline"] = util.table_copy(
6690        walkable_syntax["Inline"])
6691      self.insert_pattern(
6692        "IndentedInline instead of Space",
6693        "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`.

```
6694      for lhs, rule in pairs(walkable_syntax) do
6695        syntax[lhs] = parsers.fail
6696        for _, rhs in ipairs(rule) do
6697          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.

```
6698          if type(rhs) == "string" then
6699            pattern = V(rhs)
6700          else
6701            pattern = rhs[1]
6702            if type(pattern) == "string" then
6703              pattern = V(pattern)
6704            end
6705          end
6706          syntax[lhs] = syntax[lhs] + pattern
6707        end
6708      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.

```
6709      if options.underscores then
6710        self.add_special_character("_")
6711      end
6712
6713      if not options.codeSpans then
```

```
6714        syntax.Code = parsers.fail
6715      else
6716        self.add_special_character("`")
6717      end
6718
6719      if not options.html then
6720        syntax.DisplayHtml = parsers.fail
6721        syntax.InlineHtml = parsers.fail
6722        syntax.HtmlEntity  = parsers.fail
6723      else
6724        self.add_special_character("&")
6725      end
6726
6727      if options.preserveTabs then
6728        options.stripIndent = false
6729      end
6730
6731      if not options.smartEllipses then
6732        syntax.Smart = parsers.fail
6733      else
6734        self.add_special_character(".")
6735      end
6736
6737      if not options.relativeReferences then
6738        syntax.AutoLinkRelativeReference = parsers.fail
6739      end
6740
6741      local blocks_nested_t = util.table_copy(syntax)
6742      blocks_nested_t.ExpectedJekyllData = parsers.fail
6743      parsers.blocks_nested = Ct(blocks_nested_t)
6744
6745      parsers.blocks = Ct(syntax)
6746
6747      local inlines_t = util.table_copy(syntax)
6748      inlines_t[1] = "Inlines"
6749      inlines_t.Inlines = V("InitializeState")
6750                        * parsers.Inline^0
6751                        * ( parsers.spacing^0
6752                          * parsers.eof / "")
6753      parsers.inlines = Ct(inlines_t)
6754
6755      local inlines_no_link_t = util.table_copy(inlines_t)
6756      inlines_no_link_t.Link = parsers.fail
6757      parsers.inlines_no_link = Ct(inlines_no_link_t)
6758
6759      local inlines_no_inline_note_t = util.table_copy(inlines_t)
6760      inlines_no_inline_note_t.InlineNote = parsers.fail
```

```
    parsers.inlines_no_inline_note = Ct(inlines_no_inline_note_t)

    local inlines_no_html_t = util.table_copy(inlines_t)
    inlines_no_html_t.DisplayHtml = parsers.fail
    inlines_no_html_t.InlineHtml = parsers.fail
    inlines_no_html_t.HtmlEntity = parsers.fail
    parsers.inlines_no_html = Ct(inlines_no_html_t)

    local inlines_nbsp_t = util.table_copy(inlines_t)
    inlines_nbsp_t.Endline = parsers.NonbreakingEndline
    inlines_nbsp_t.Space = parsers.NonbreakingSpace
    parsers.inlines_nbsp = Ct(inlines_nbsp_t)
```

Return a function that converts markdown string `input` into a plain TEX output and returns it..

```
    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.

```
      input = input:gsub("\r\n?", "\n")
      if input:sub(-1) ~= "\n" then
        input = input .. "\n"
      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 span 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

      self.insert_pattern("Inline after Emph",
                          Span, "Span")
    end
  }
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.

```
M.extensions.citations = function(citation_nbsps)
  return {
    name = "built-in citations syntax extension",
    extend_writer = function(self)
```

Define `writer->citations` as a function that will transform an input array of citations `cites` to the output format. If `text_cites` is enabled, the citations should be rendered in-text, when applicable. The `cites` array contains tables with the following keys and values:

- `suppress_author` – If the value of the key is true, then the author of the work should be omitted in the citation, when applicable.

- `prenote` – The value of the key is either `nil` or a rope that should be inserted before the citation.
- `postnote` – The value of the key is either `nil` or a rope that should be inserted after the citation.
- `name` – The value of this key is the citation name.

```
6854      function self.citations(text_cites, cites)
6855        local buffer = {"\\markdownRenderer", text_cites and "TextCite" or "Cite",
6856          "{", #cites, "}"}
6857        for _,cite in ipairs(cites) do
6858          buffer[#buffer+1] = {cite.suppress_author and "-" or "+", "{",
6859            cite.prenote or "", "}{", cite.postnote or "", "}{", cite.name, "}"}
6860        end
6861        return buffer
6862      end
6863    end, extend_reader = function(self)
6864      local parsers = self.parsers
6865      local writer = self.writer
6866
6867      local citation_chars
6868                    = parsers.alphanumeric
6869                    + S("#$%&-+<>~/_")
6870
6871      local citation_name
6872                    = Cs(parsers.dash^-1) * parsers.at
6873                    * Cs(citation_chars
6874                        * (((citation_chars + parsers.internal_punctuation
6875                            - parsers.comma - parsers.semicolon)
6876                           * -#((parsers.internal_punctuation - parsers.comma
6877                                - parsers.semicolon)^0
6878                               * -(citation_chars + parsers.internal_punctuation
6879                                  - parsers.comma - parsers.semicolon)))^0
6880                          * citation_chars)^-1)
6881
6882      local citation_body_prenote
6883                    = Cs((parsers.alphanumeric^1
6884                         + parsers.bracketed
6885                         + parsers.inticks
6886                         + (parsers.anyescaped
6887                           - (parsers.rbracket + parsers.blankline^2))
6888                         - (parsers.spnl * parsers.dash^-1 * parsers.at))^0)
6889
6890      local citation_body_postnote
6891                    = Cs((parsers.alphanumeric^1
6892                         + parsers.bracketed
6893                         + 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

          local cites = {}
          for i = 1,#raw_cites,4 do
              cites[#cites+1] = {
```

```
                prenote = normalize(raw_cites[i]),
                suppress_author = raw_cites[i+1] == "-",
                name = writer.identifier(raw_cites[i+2]),
                postnote = normalize(raw_cites[i+3]),
            }
        end
        return writer.citations(text_cites, cites)
    end

    local TextCitations
                  = Ct((parsers.spnl
                  * Cc("")
                  * citation_name
                  * ((parsers.spnl
                      * parsers.lbracket
                      * citation_headless_body
                      * parsers.rbracket) + Cc("")))^1)
                  / function(raw_cites)
                      return citations(true, raw_cites)
                    end

    local ParenthesizedCitations
                  = Ct((parsers.spnl
                  * parsers.lbracket
                  * citation_body
                  * parsers.rbracket)^1)
                  / function(raw_cites)
                      return citations(false, raw_cites)
                    end

    local Citations = TextCitations + ParenthesizedCitations

    self.insert_pattern("Inline after Emph",
                        Citations, "Citations")

    self.add_special_character("@")
    self.add_special_character("-")
  end
}
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 ""),"}"}
      elseif languages_json[suf] then
        return {"\\markdownRendererContentBlockCode{",suf,"}",
                                "{",self.string(languages_json[suf]),"}",
                                "{",self.string(src),"}",
```

```
                                "{",self.uri(src),"}",
                                "{",self.string(tit or ""),"}"}
      else
        return {"\\markdownRendererContentBlock{",suf,"}",
                                "{",self.string(src),"}",
                                "{",self.uri(src),"}",
                                "{",self.string(tit or ""),"}"}
      end
    end
  end, extend_reader = function(self)
    local parsers = self.parsers
    local writer = self.writer

    local contentblock_tail
                = parsers.optionaltitle
                * (parsers.newline + parsers.eof)

    -- case insensitive online image suffix:
    local onlineimagesuffix
                = (function(...)
                    local parser = nil
                    for _, suffix in ipairs({...}) do
                      local pattern=nil
                      for i=1,#suffix do
                        local char=suffix:sub(i,i)
                        char = S(char:lower()..char:upper())
                        if pattern == nil then
                          pattern = char
                        else
                          pattern = pattern * char
                        end
                      end
                      if parser == nil then
                        parser = pattern
                      else
                        parser = parser + pattern
                      end
                    end
                    return parser
                  end)("png", "jpg", "jpeg", "gif", "tif", "tiff")

    -- online image url for iA Writer content blocks with mandatory suffix,
    -- 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.

```
7108 M.extensions.definition_lists = function(tight_lists)
7109   return {
7110     name = "built-in definition_lists syntax extension",
7111     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.

```
7112       local function dlitem(term, defs)
7113         local retVal = {"\\markdownRendererDlItem{",term,"}"}
7114         for _, def in ipairs(defs) do
7115           retVal[#retVal+1] = {"\\markdownRendererDlDefinitionBegin ",def,
7116                                "\\markdownRendererDlDefinitionEnd "}
7117         end
7118         retVal[#retVal+1] = "\\markdownRendererDlItemEnd "
7119         return retVal
7120       end
7121
7122       function self.definitionlist(items,tight)
7123         if not self.is_writing then return "" end
7124         local buffer = {}
7125         for _,item in ipairs(items) do
7126           buffer[#buffer + 1] = dlitem(item.term, item.definitions)
7127         end
7128         if tight and tight_lists then
7129           return {"\\markdownRendererDlBeginTight\n", buffer,
7130             "\n\\markdownRendererDlEndTight"}
7131         else
7132           return {"\\markdownRendererDlBegin\n", buffer,
7133             "\n\\markdownRendererDlEnd"}
7134         end
7135       end
7136     end, extend_reader = function(self)
7137       local parsers = self.parsers
7138       local writer = self.writer
7139
7140       local defstartchar = S("~:")
7141
7142       local defstart = ( defstartchar * #parsers.spacing
7143                                       * (parsers.tab + parsers.space^-
    3)
7144                        + parsers.space * defstartchar * #parsers.spacing
7145                                        * (parsers.tab + parsers.space^-
    2)
7146                        + parsers.space * parsers.space * defstartchar
7147                                        * #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.

```
function self.fancyitem(s,num)
  if num ~= nil then
```

```
          return {"\\markdownRendererFancyOlItemWithNumber{",num,"}",s,
                  "\\markdownRendererFancyOlItemEnd "}
        else
          return {"\\markdownRendererFancyOlItem ",s,"\\markdownRendererFancyOlItemEn
        end
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local options = self.options
      local writer = self.writer

      local label = parsers.dig + parsers.letter
      local numdelim = parsers.period + parsers.rparent
      local enumerator = C(label^3 * numdelim) * #parsers.spacing
                       + C(label^2 * numdelim) * #parsers.spacing
                                               * (parsers.tab + parsers.space^1)
                       + C(label * numdelim) * #parsers.spacing
                                             * (parsers.tab + parsers.space^-
    2)
                       + parsers.space * C(label^2 * numdelim)
                                       * #parsers.spacing
                       + parsers.space * C(label * numdelim)
                                       * #parsers.spacing
                                       * (parsers.tab + parsers.space^-
    1)
                       + parsers.space * parsers.space * C(label^1
                                       * numdelim) * #parsers.spacing
      local starter = parsers.bullet + enumerator

      local NestedList = Cs((parsers.optionallyindentedline
                           - starter)^1)
                       / function(a) return "\001"..a end

      local ListBlockLine  = parsers.optionallyindentedline
                           - parsers.blankline - (parsers.indent^-1
                                                  * starter)

      local ListBlock = parsers.line * ListBlockLine^0

      local ListContinuationBlock = parsers.blanklines * (parsers.indent / "")
                                  * ListBlock

      local TightListItem = function(starter)
          return -parsers.ThematicBreak
                 * (Cs(starter / "" * parsers.tickbox^-1 * ListBlock * NestedList^-
    1)
                   / self.parser_functions.parse_blocks_nested)
```

```
7258                * -(parsers.blanklines * parsers.indent)
7259        end
7260
7261        local LooseListItem = function(starter)
7262            return -parsers.ThematicBreak
7263                   * Cs( starter / "" * parsers.tickbox^-1 * ListBlock * Cc("\n")
7264                     * (NestedList + ListContinuationBlock^0)
7265                     * (parsers.blanklines / "\n\n")
7266                     ) / self.parser_functions.parse_blocks_nested
7267        end
7268
7269        local function roman2number(roman)
7270          local romans = { ["L"] = 50, ["X"] = 10, ["V"] = 5, ["I"] = 1 }
7271          local numeral = 0
7272
7273          local i = 1
7274          local len = string.len(roman)
7275          while i < len do
7276            local z1, z2 = romans[ string.sub(roman, i, i) ], romans[ string.sub(roman,
7277            if z1 < z2 then
7278                numeral = numeral + (z2 - z1)
7279                i = i + 2
7280            else
7281                numeral = numeral + z1
7282                i = i + 1
7283            end
7284          end
7285          if i <= len then numeral = numeral + romans[ string.sub(roman,i,i) ] end
7286          return numeral
7287        end
7288
7289        local function sniffstyle(itemprefix)
7290          local numstr, delimend = itemprefix:match("^([A-Za-z0-9]*)([.)]*)")
7291          local numdelim
7292          if delimend == ")" then
7293            numdelim = "OneParen"
7294          elseif delimend == "." then
7295            numdelim = "Period"
7296          else
7297            numdelim = "Default"
7298          end
7299          numstr = numstr or itemprefix
7300
7301          local num
7302          num = numstr:match("^([IVXL]+)")
7303          if num then
7304            return roman2number(num), "UpperRoman", numdelim
```

```
7305          end
7306          num = numstr:match("^([ivxl]+)")
7307          if num then
7308            return roman2number(string.upper(num)), "LowerRoman", numdelim
7309          end
7310          num = numstr:match("^([A-Z])")
7311          if num then
7312            return string.byte(num) - string.byte("A") + 1, "UpperAlpha", numdelim
7313          end
7314          num = numstr:match("^([a-z])")
7315          if num then
7316            return string.byte(num) - string.byte("a") + 1, "LowerAlpha", numdelim
7317          end
7318          return math.floor(tonumber(numstr) or 1), "Decimal", numdelim
7319        end
7320
7321        local function fancylist(items,tight,start)
7322          local startnum, numstyle, numdelim = sniffstyle(start)
7323          return writer.fancylist(items,tight,
7324                                  options.startNumber and startnum,
7325                                  numstyle or "Decimal",
7326                                  numdelim or "Default")
7327        end
7328
7329        local FancyList = Cg(enumerator, "listtype") *
7330                          ( Ct(TightListItem(Cb("listtype"))
7331                              * TightListItem(enumerator)^0)
7332                          * Cc(true) * parsers.skipblanklines * -enumerator
7333                          + Ct(LooseListItem(Cb("listtype"))
7334                              * LooseListItem(enumerator)^0)
7335                          * Cc(false) * parsers.skipblanklines
7336                          ) * Cb("listtype") / fancylist
7337
7338        self.update_rule("OrderedList", FancyList)
7339      end
7340    }
7341  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 the Pandoc 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

      local function captures_geq_length(_,i,a,b)
```

```
7381          return #a >= #b and i
7382        end
7383
7384        local tilde_infostring
7385                              = C((parsers.linechar
7386                                 - (parsers.spacechar^1 * parsers.newline))^0)
7387
7388        local backtick_infostring
7389                              = C((parsers.linechar
7390                                 - (parsers.backtick
7391                                   + parsers.spacechar^1 * parsers.newline))^0)
7392
7393        local fenceindent
7394        local fencehead       = function(char, infostring)
7395          return                C(parsers.nonindentspace) / function(s) fenceindent = #s
7396                              * Cg(char^3, "fencelength")
7397                              * parsers.optionalspace
7398                              * infostring
7399                              * (parsers.newline + parsers.eof)
7400        end
7401
7402        local fencetail       = function(char)
7403          return                parsers.nonindentspace
7404                              * Cmt(C(char^3) * Cb("fencelength"), captures_geq_length)
7405                              * parsers.optionalspace * (parsers.newline + parsers.eof)
7406                              + parsers.eof
7407        end
7408
7409        local fencedline      = function(char)
7410          return                C(parsers.line - fencetail(char))
7411                              / function(s)
7412                                  local i = 1
7413                                  local remaining = fenceindent
7414                                  while true do
7415                                    local c = s:sub(i, i)
7416                                    if c == " " and remaining > 0 then
7417                                      remaining = remaining - 1
7418                                      i = i + 1
7419                                    elseif c == "\t" and remaining > 3 then
7420                                      remaining = remaining - 4
7421                                      i = i + 1
7422                                    else
7423                                      break
7424                                    end
7425                                  end
7426                                  return s:sub(i)
7427                                end
```

```
7428        end
7429
7430        local TildeFencedCode
7431               = fencehead(parsers.tilde, tilde_infostring)
7432               * Cs(fencedline(parsers.tilde)^0)
7433               * fencetail(parsers.tilde)
7434
7435        local BacktickFencedCode
7436               = fencehead(parsers.backtick, backtick_infostring)
7437               * Cs(fencedline(parsers.backtick)^0)
7438               * fencetail(parsers.backtick)
7439
7440             local infostring_with_attributes
7441                             = Ct(C((parsers.linechar
7442                                     - ( parsers.optionalspace
7443                                       * parsers.attributes))^0)
7444                                  * parsers.optionalspace
7445                                  * Ct(parsers.attributes))
7446
7447        local FencedCode
7448                 = (TildeFencedCode + BacktickFencedCode)
7449                 / function(infostring, code)
7450                     local expanded_code = self.expandtabs(code)
7451
7452                     if allow_raw_blocks then
7453                       local raw_attr = lpeg.match(parsers.raw_attribute,
7454                                                   infostring)
7455                       if raw_attr then
7456                         return writer.rawBlock(expanded_code, raw_attr)
7457                       end
7458                     end
7459
7460                     local attr = nil
7461                     if allow_attributes then
7462                       local match = lpeg.match(infostring_with_attributes,
7463                                                infostring)
7464                       if match then
7465                         infostring, attr = table.unpack(match)
7466                       end
7467                     end
7468                     return writer.fencedCode(expanded_code, infostring, attr)
7469                   end
7470
7471        self.insert_pattern("Block after Verbatim",
7472                            FencedCode, "FencedCode")
7473
7474        local fencestart
```

```
7475        if blank_before_code_fence then
7476          fencestart = parsers.fail
7477        else
7478          fencestart = fencehead(parsers.backtick, backtick_infostring)
7479                     + fencehead(parsers.tilde, tilde_infostring)
7480        end
7481
7482        self.update_rule("EndlineExceptions", function(previous_pattern)
7483          if previous_pattern == nil then
7484            previous_pattern = parsers.EndlineExceptions
7485          end
7486          return previous_pattern + fencestart
7487        end)
7488
7489        self.add_special_character("`")
7490        self.add_special_character("~")
7491      end
7492    }
7493  end
```

3.1.6.7 Fenced Divs

The `extensions.fenced_divs` function implements the Pandoc fenced div syntax extension. When the `blank_before_div_fence` parameter is `true`, the syntax extension requires a blank line between a paragraph and the following fenced code block.

```
7494  M.extensions.fenced_divs = function(blank_before_div_fence)
7495    return {
7496      name = "built-in fenced_divs syntax extension",
7497      extend_writer = function(self)
```

Define `writer->div_begin` as a function that will transform the beginning of an input fenced div with with attributes `attributes` to the output format.

```
7498        function self.div_begin(attributes)
7499          local start_output = {"\\markdownRendererFencedDivAttributeContextBegin\n",
7500                                self.attributes(attributes)}
7501          local end_output = {"\n\\markdownRendererFencedDivAttributeContextEnd "}
7502          return self.push_attributes("div", attributes, start_output, end_output)
7503        end
```

Define `writer->div_end` as a function that will produce the end of a fenced div in the output format.

```
7504        function self.div_end()
7505          return self.pop_attributes("div")
7506        end
7507      end, extend_reader = function(self)
7508        local parsers = self.parsers
7509        local writer = self.writer
```

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

```
7510        local fenced_div_infostring
7511                                = C((parsers.linechar
7512                                   - ( parsers.spacechar^1
7513                                     * parsers.colon^1))^1)
7514
7515        local fenced_div_begin = parsers.nonindentspace
7516                                * parsers.colon^3
7517                                * parsers.optionalspace
7518                                * fenced_div_infostring
7519                                * ( parsers.spacechar^1
7520                                  * parsers.colon^1)^0
7521                                * parsers.optionalspace
7522                                * (parsers.newline + parsers.eof)
7523
7524        local fenced_div_end = parsers.nonindentspace
7525                              * parsers.colon^3
7526                              * parsers.optionalspace
7527                              * (parsers.newline + parsers.eof)
```

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

```
7528        self.initialize_named_group("div_level", "0")
7529
7530        local function increment_div_level(increment)
7531          local function update_div_level(s, i, current_level) -- luacheck: ignore s i
7532            current_level = tonumber(current_level)
7533            local next_level = tostring(current_level + increment)
7534            return true, next_level
7535          end
7536
7537          return Cg( Cmt(Cb("div_level"), update_div_level)
7538                   , "div_level")
7539        end
7540
7541        local FencedDiv = fenced_div_begin
7542                        / function (infostring)
7543                            local attr = lpeg.match(Ct(parsers.attributes), infostring)
7544                            if attr == nil then
7545                              attr = {"." .. infostring}
7546                            end
7547                            return attr
7548                          end
7549                        / writer.div_begin
7550                        * increment_div_level(1)
7551                        * parsers.skipblanklines
7552                        * 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)
                        * (Cc("") / writer.div_end)

      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 attribute 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", Heading)
    end
  }
end
```

3.1.6.9 Inline Code Attributes

The `extensions.inline_code_attributes` function implements the Pandoc inline code attribute syntax extension.

```
M.extensions.inline_code_attributes = function()
  return {
    name = "built-in inline_code_attributes syntax extension",
    extend_writer = function()
    end, extend_reader = function(self)
      local writer = self.writer

      local CodeWithAttributes = parsers.inticks
                               * Ct(parsers.attributes)
                               / writer.code

      self.insert_pattern("Inline before Code",
                          CodeWithAttributes,
                          "CodeWithAttributes")
    end
  }
end
```

3.1.6.10 Line Blocks

The `extensions.line_blocks` function implements the Pandoc line block 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.hard_line_break }
        end
```

```
7675          buffer[#buffer + 1] = lines[#lines]
7676
7677          return {"\\markdownRendererLineBlockBegin\n"
7678                    ,buffer,
7679                    "\n\\markdownRendererLineBlockEnd "}
7680        end
7681      end, extend_reader = function(self)
7682        local parsers = self.parsers
7683        local writer = self.writer
7684
7685        local LineBlock = Ct(
7686                          (Cs(
7687                            ( (parsers.pipe * parsers.space)/""
7688                            * ((parsers.space)/entities.char_entity("nbsp"))^0
7689                            * parsers.linechar^0 * (parsers.newline/""))
7690                            * (-parsers.pipe
7691                              * (parsers.space^1/" ")
7692                              * parsers.linechar^1
7693                              * (parsers.newline/"")
7694                              )^0
7695                            * (parsers.blankline/"")^0
7696                          ) / self.parser_functions.parse_inlines)^1) / writer.linebloc
7697
7698        self.insert_pattern("Block after Blockquote",
7699                             LineBlock, "LineBlock")
7700      end
7701    }
7702  end
```

3.1.6.11 Link Attributes

The `extensions.link_attributes` function implements the Pandoc link attribute syntax extension.

```
7703  M.extensions.link_attributes = function()
7704    return {
7705      name = "built-in link_attributes syntax extension",
7706      extend_writer = function()
7707      end, extend_reader = function(self)
7708        local parsers = self.parsers
7709        local writer = self.writer
7710        local options = self.options
7711
```

The following patterns define link reference definitions with attributes.

```
7712
7713        local define_reference_parser = parsers.define_reference_parser
7714                                      * ( parsers.spnl
7715                                        * Ct(parsers.attributes))^-1
7716
```

```
7717        local ReferenceWithAttributes = define_reference_parser
7718                                      * parsers.blankline^1
7719                                      / self.register_link
7720
7721        self.update_rule("Reference", ReferenceWithAttributes)
7722
```

The following patterns define direct and indirect links with attributes.

```
7723
7724        local function indirect_link(label, sps, tag,
7725                                     attribute_text,
7726                                     attributes)
7727          return writer.defer_call(function()
7728            local r, fallback = self.lookup_reference(label, sps, tag,
7729                                                      attributes)
7730            if r then
7731              return writer.link(
7732                self.parser_functions.parse_inlines_no_link(label),
7733                r.url, r.title, r.attributes)
7734            else
7735              local buf = {fallback}
7736              if attributes then
7737                table.insert(buf, writer.string(attribute_text))
7738              end
7739              return buf
7740            end
7741          end)
7742        end
7743
7744        local DirectLinkWithAttributes = parsers.direct_link
7745                                       * (Ct(parsers.attributes))^-1
7746                                       / writer.link
7747
7748        local IndirectLinkWithAttributes = parsers.indirect_link
7749                                         * (C(Ct(parsers.attributes)))^-1
7750                                         / indirect_link
7751
7752        local LinkWithAttributes = DirectLinkWithAttributes
7753                                 + IndirectLinkWithAttributes
7754
```

Here, we directly update the `Link` grammar rule to keep the method `reader->parser_functions.parse_inlines_no_link` aware of `LinkWithAttributes` and prevent nested links.

If we used `reader->insert_pattern` instead of `reader->update_rule`, this correspondence would have been lost and link text would be able to contain nested links.

```
7755        self.update_rule("Link", LinkWithAttributes)
7756
```

The following patterns define direct and indirect images with attributes.

```
7757
7758        local function indirect_image(label, sps, tag,
7759                                      attribute_text,
7760                                      attributes)
7761          return writer.defer_call(function()
7762            local r, fallback = self.lookup_reference(label, sps, tag,
7763                                                      attributes)
7764            if r then
7765              return writer.image(writer.string(label),
7766                                  r.url, r.title, r.attributes)
7767            else
7768              local buf = {"!", fallback}
7769              if attributes then
7770                table.insert(buf, writer.string(attribute_text))
7771              end
7772              return buf
7773            end
7774          end)
7775        end
7776
7777        local DirectImageWithAttributes = parsers.direct_image
7778                                        * Ct(parsers.attributes)
7779                                        / writer.image
7780
7781        local IndirectImageWithAttributes = parsers.indirect_image
7782                                          * C(Ct(parsers.attributes))
7783                                          / indirect_image
7784
7785        local ImageWithAttributes = DirectImageWithAttributes
7786                                  + IndirectImageWithAttributes
7787
7788        self.insert_pattern("Inline before Image",
7789                            ImageWithAttributes,
7790                            "ImageWithAttributes")
7791
```

The following patterns define autolinks with attributes.

```
7792
7793        local AutoLinkUrlWithAttributes
7794                        = parsers.auto_link_url
7795                        * Ct(parsers.attributes)
7796                        / self.auto_link_url
7797
7798        self.insert_pattern("Inline before AutoLinkUrl",
```

```
                          AutoLinkUrlWithAttributes,
                          "AutoLinkUrlWithAttributes")

      local AutoLinkEmailWithAttributes
                      = parsers.auto_link_email
                      * Ct(parsers.attributes)
                      / self.auto_link_email

      self.insert_pattern("Inline before AutoLinkEmail",
                          AutoLinkEmailWithAttributes,
                          "AutoLinkEmailWithAttributes")

      if options.relativeReferences then

        local AutoLinkRelativeReferenceWithAttributes
                        = parsers.auto_link_relative_reference
                        * Ct(parsers.attributes)
                        / self.auto_link_url

        self.insert_pattern(
          "Inline before AutoLinkRelativeReference",
          AutoLinkRelativeReferenceWithAttributes,
          "AutoLinkRelativeReferenceWithAttributes")

      end

    end
  }
end
```

3.1.6.12 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)
```

```
7837        local parsers = self.parsers
7838        local writer = self.writer
7839
7840        if inline_notes then
7841          local InlineNote
7842                      = parsers.circumflex
7843                      * (parsers.tag / self.parser_functions.parse_inlines_no_inline_no
7844                      / writer.note
7845
7846          self.insert_pattern("Inline after Emph",
7847                              InlineNote, "InlineNote")
7848        end
7849        if notes then
7850          local function strip_first_char(s)
7851            return s:sub(2)
7852          end
7853
7854          local RawNoteRef
7855                        = #(parsers.lbracket * parsers.circumflex)
7856                        * parsers.tag / strip_first_char
7857
7858          local rawnotes = {}
7859
7860          -- like indirect_link
7861          local function lookup_note(ref)
7862            return writer.defer_call(function()
7863              local found = rawnotes[self.normalize_tag(ref)]
7864              if found then
7865                return writer.note(
7866                  self.parser_functions.parse_blocks_nested(found))
7867              else
7868                return {"[",
7869                  self.parser_functions.parse_inlines("^" .. ref), "]"}
7870              end
7871            end)
7872          end
7873
7874          local function register_note(ref,rawnote)
7875            rawnotes[self.normalize_tag(ref)] = rawnote
7876            return ""
7877          end
7878
7879          local NoteRef = RawNoteRef / lookup_note
7880
7881          local NoteBlock
7882                      = parsers.leader * RawNoteRef * parsers.colon
7883                      * parsers.spnl * parsers.indented_blocks(parsers.chunk)
```

```
                    / register_note

        local Blank = NoteBlock + parsers.Blank
        self.update_rule("Blank", Blank)

        self.insert_pattern("Inline after Emph",
                            NoteRef, "NoteRef")
      end

      self.add_special_character("^")
    end
  }
end
```

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

```
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 = {}
      for j = 1, num_columns do
        rectangular_row[j] = row[j] or ""
      end
      table.insert(rectangular_rows, rectangular_row)
    end
    return rectangular_rows
  end

  local function pipe_table_row(allow_empty_first_column
                                , nonempty_column
                                , column_separator
                                , column)
    local row_beginning
    if allow_empty_first_column then
      row_beginning = -- empty first column
                      #(parsers.spacechar^4
                        * column_separator)
                    * parsers.optionalspace
                    * column
                    * parsers.optionalspace
```

```
                    -- non-empty first column
                    + parsers.nonindentspace
                    * nonempty_column^-1
                    * parsers.optionalspace
  else
    row_beginning = parsers.nonindentspace
                    * nonempty_column^-1
                    * parsers.optionalspace
  end

  return Ct(row_beginning
            * (-- single column with no leading pipes
               #(column_separator
                 * parsers.optionalspace
                 * parsers.newline)
               * column_separator
               * parsers.optionalspace
               -- single column with leading pipes or
               -- more than a single column
               + (column_separator
                 * parsers.optionalspace
                 * column
                 * parsers.optionalspace)^1
               * (column_separator
                 * parsers.optionalspace)^-1))
end

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))
```

```
8015
8016        local table_caption
8017        if table_captions then
8018          table_caption = #table_caption_beginning
8019                        * table_caption_beginning
8020                        * Ct(parsers.IndentedInline^1)
8021                        * parsers.newline
8022        else
8023          table_caption = parsers.fail
8024        end
8025
8026        local PipeTable = Ct(table_row * parsers.newline
8027                             * table_hline
8028                             * (parsers.newline * table_row)^0)
8029                        / make_pipe_table_rectangular
8030                        * table_caption^-1
8031                        / writer.table
8032
8033        self.insert_pattern("Block after Blockquote",
8034                            PipeTable, "PipeTable")
8035      end
8036    }
8037  end
```

3.1.6.14 Raw Attributes The `extensions.raw_inline` function implements the Pandoc raw attribute syntax extension for inline code spans.

```
8038  M.extensions.raw_inline = function()
8039    return {
8040      name = "built-in raw_inline syntax extension",
8041      extend_writer = function(self)
8042        local options = self.options
8043
```

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

```
8044        function self.rawInline(s, attr)
8045          if not self.is_writing then return "" end
8046          local name = util.cache_verbatim(options.cacheDir, s)
8047          return {"\\markdownRendererInputRawInline{",
8048                  name,"}{", self.string(attr),"}"}
8049        end
8050      end, extend_reader = function(self)
8051        local writer = self.writer
8052
8053        local RawInline = parsers.inticks
8054                        * parsers.raw_attribute
8055                        / writer.rawInline
```

```
8056
8057        self.insert_pattern("Inline before Code",
8058                            RawInline, "RawInline")
8059      end
8060    }
8061  end
```

3.1.6.15 Strike-Through

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

```
8062  M.extensions.strike_through = function()
8063    return {
8064      name = "built-in strike_through syntax extension",
8065      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.

```
8066        function self.strike_through(s)
8067          return {"\\markdownRendererStrikeThrough{",s,"}"}
8068        end
8069      end, extend_reader = function(self)
8070        local parsers = self.parsers
8071        local writer = self.writer
8072
8073        local StrikeThrough = (
8074          parsers.between(parsers.Inline, parsers.doubletildes,
8075                          parsers.doubletildes)
8076        ) / writer.strike_through
8077
8078        self.insert_pattern("Inline after Emph",
8079                            StrikeThrough, "StrikeThrough")
8080
8081        self.add_special_character("~")
8082      end
8083    }
8084  end
```

3.1.6.16 Subscripts

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

```
8085  M.extensions.subscripts = function()
8086    return {
8087      name = "built-in subscripts syntax extension",
8088      extend_writer = function(self)
```

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

```
8089        function self.subscript(s)
```

```
        return {"\\markdownRendererSubscript{",s,"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local Subscript = (
        parsers.between(parsers.Str, parsers.tilde, parsers.tilde)
      ) / writer.subscript

      self.insert_pattern("Inline after Emph",
                          Subscript, "Subscript")

      self.add_special_character("~")
    end
  }
end
```

3.1.6.17 Superscripts

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

```
M.extensions.superscripts = function()
  return {
    name = "built-in superscripts syntax extension",
    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
```

3.1.6.18 Tex Math

The `extensions.tex_math` function implements the Pandoc math syntax extensions.

```
M.extensions.tex_math = function(tex_math_dollars,
                                 tex_math_single_backslash,
                                 tex_math_double_backslash)
  return {
    name = "built-in tex_math syntax extension",
    extend_writer = function(self)
```

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

```
      function self.display_math(s)
        if not self.is_writing then return "" end
        return {"\\markdownRendererDisplayMath{",self.math(s),"}"}
      end
```

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

```
      function self.inline_math(s)
        if not self.is_writing then return "" end
        return {"\\markdownRendererInlineMath{",self.math(s),"}"}
      end
    end, extend_reader = function(self)
      local parsers = self.parsers
      local writer = self.writer

      local function between(p, starter, ender)
        return (starter * C(p * (p - ender)^0) * ender)
      end

      local allowed_before_closing = B( parsers.backslash * parsers.any
                                      + parsers.any * (parsers.nonspacechar - parsers
```

The following patterns implement the Pandoc dollar math syntax extension.

```
      local dollar_math_content = parsers.backslash^-1
                                * parsers.any
                                - parsers.blankline^2
                                - parsers.dollar

      local inline_math_opening_dollars = parsers.dollar
                                        * #(parsers.nonspacechar)

      local inline_math_closing_dollars = allowed_before_closing
                                        * parsers.dollar
                                        * -#(parsers.digit)

      local inline_math_dollars = between(C( dollar_math_content),
                                          inline_math_opening_dollars,
```

```
                                          inline_math_closing_dollars)

    local display_math_opening_dollars  = parsers.dollar
                                        * parsers.dollar

    local display_math_closing_dollars  = parsers.dollar
                                        * parsers.dollar

    local display_math_dollars = between(C( dollar_math_content),
                                         display_math_opening_dollars,
                                         display_math_closing_dollars)
```

The following patterns implement the Pandoc single and double backslash math syntax extensions.

```
    local backslash_math_content  = parsers.any
                                  - parsers.blankline^2
```

The following patterns implement the Pandoc double backslash math syntax extension.

```
    local inline_math_opening_double  = parsers.backslash
                                      * parsers.backslash
                                      * parsers.lparent
                                      * #(parsers.nonspacechar)

    local inline_math_closing_double  = allowed_before_closing
                                      * parsers.backslash
                                      * parsers.backslash
                                      * parsers.rparent

    local inline_math_double  = between(C( backslash_math_content),
                                        inline_math_opening_double,
                                        inline_math_closing_double)

    local display_math_opening_double = parsers.backslash
                                      * parsers.backslash
                                      * parsers.lbracket

    local display_math_closing_double = allowed_before_closing
                                      * parsers.backslash
                                      * parsers.backslash
                                      * parsers.rbracket

    local display_math_double = between(C( backslash_math_content),
                                        display_math_opening_double,
                                        display_math_closing_double)
```

The following patterns implement the Pandoc single backslash math syntax extension.

```
    local inline_math_opening_single  = parsers.backslash
```

```
                                       * parsers.lparent
                                       * #(parsers.nonspacechar)

      local inline_math_closing_single  = allowed_before_closing
                                       * parsers.backslash
                                       * parsers.rparent

      local inline_math_single  = between(C( backslash_math_content),
                                          inline_math_opening_single,
                                          inline_math_closing_single)

      local display_math_opening_single = parsers.backslash
                                       * parsers.lbracket

      local display_math_closing_single = allowed_before_closing
                                       * parsers.backslash
                                       * parsers.rbracket

      local display_math_single = between(C( backslash_math_content),
                                          display_math_opening_single,
                                          display_math_closing_single)

      local display_math = parsers.fail

      local inline_math = parsers.fail

      if tex_math_dollars then
        display_math = display_math + display_math_dollars
        inline_math = inline_math + inline_math_dollars
      end

      if tex_math_double_backslash then
        display_math = display_math + display_math_double
        inline_math = inline_math + inline_math_double
      end

      if tex_math_single_backslash then
        display_math = display_math + display_math_single
        inline_math = inline_math + inline_math_single
      end

      local TexMath = display_math / writer.display_math
                    + inline_math / writer.inline_math

      self.insert_pattern("Inline after Emph",
                          TexMath, "TexMath")
```

```
      if tex_math_dollars then
        self.add_special_character("$")
      end

      if tex_math_single_backslash or tex_math_double_backslash then
        self.add_special_character("\\")
        self.add_special_character("[")
        self.add_special_character("]")
        self.add_special_character(")")
        self.add_special_character("(")
      end
    end
  }
end
```

3.1.6.19 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)

        if not p then
          table.insert(buf, "\\markdownRendererJekyllDataBegin")
        end

        if #d > 0 then
            table.insert(buf, "\\markdownRendererJekyllDataSequenceBegin{")
```

```
            table.insert(buf, self.identifier(p or "null"))
            table.insert(buf, "}{")
            table.insert(buf, #keys)
            table.insert(buf, "}")
        else
            table.insert(buf, "\\markdownRendererJekyllDataMappingBegin{")
            table.insert(buf, self.identifier(p or "null"))
            table.insert(buf, "}{")
            table.insert(buf, #keys)
            table.insert(buf, "}")
        end

        for _, k in ipairs(keys) do
          local v = d[k]
          local typ = type(v)
          k = tostring(k or "null")
          if typ == "table" and next(v) ~= nil then
            table.insert(
              buf,
              self.jekyllData(v, t, k)
            )
          else
            k = self.identifier(k)
            v = tostring(v)
            if typ == "boolean" then
              table.insert(buf, "\\markdownRendererJekyllDataBoolean{")
              table.insert(buf, k)
              table.insert(buf, "}{")
              table.insert(buf, v)
              table.insert(buf, "}")
            elseif typ == "number" then
              table.insert(buf, "\\markdownRendererJekyllDataNumber{")
              table.insert(buf, k)
              table.insert(buf, "}{")
              table.insert(buf, v)
              table.insert(buf, "}")
            elseif typ == "string" then
              table.insert(buf, "\\markdownRendererJekyllDataString{")
              table.insert(buf, k)
              table.insert(buf, "}{")
              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 b
                    * 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", ExpectedJekyllData)
      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

  if options.definitionLists then
    local definition_lists_extension = M.extensions.definition_lists(
      options.tightLists)
    table.insert(extensions, definition_lists_extension)
  end
```

```
8421
8422  if options.fencedCode then
8423    local fenced_code_extension = M.extensions.fenced_code(
8424      options.blankBeforeCodeFence,
8425      options.fencedCodeAttributes,
8426      options.rawAttribute)
8427    table.insert(extensions, fenced_code_extension)
8428  end
8429
8430  if options.fencedDivs then
8431    local fenced_div_extension = M.extensions.fenced_divs(
8432      options.blankBeforeDivFence)
8433    table.insert(extensions, fenced_div_extension)
8434  end
8435
8436  if options.headerAttributes then
8437    local header_attributes_extension = M.extensions.header_attributes()
8438    table.insert(extensions, header_attributes_extension)
8439  end
8440
8441  if options.inlineCodeAttributes then
8442    local inline_code_attributes_extension =
8443      M.extensions.inline_code_attributes()
8444    table.insert(extensions, inline_code_attributes_extension)
8445  end
8446
8447  if options.jekyllData then
8448    local jekyll_data_extension = M.extensions.jekyll_data(
8449      options.expectJekyllData)
8450    table.insert(extensions, jekyll_data_extension)
8451  end
8452
8453  if options.linkAttributes then
8454    local link_attributes_extension =
8455      M.extensions.link_attributes()
8456    table.insert(extensions, link_attributes_extension)
8457  end
8458
8459  if options.lineBlocks then
8460    local line_block_extension = M.extensions.line_blocks()
8461    table.insert(extensions, line_block_extension)
8462  end
8463
8464  if options.pipeTables then
8465    local pipe_tables_extension = M.extensions.pipe_tables(
8466      options.tableCaptions)
8467    table.insert(extensions, pipe_tables_extension)
```

```
  end

  if options.rawAttribute then
    local raw_inline_extension = M.extensions.raw_inline()
    table.insert(extensions, raw_inline_extension)
  end

  if options.strikeThrough then
    local strike_through_extension = M.extensions.strike_through()
    table.insert(extensions, strike_through_extension)
  end

  if options.subscripts then
    local subscript_extension = M.extensions.subscripts()
    table.insert(extensions, subscript_extension)
  end

  if options.superscripts then
    local superscript_extension = M.extensions.superscripts()
    table.insert(extensions, superscript_extension)
  end

  if options.texMathDollars or
     options.texMathSingleBackslash or
     options.texMathDoubleBackslash then
    local tex_math_extension = M.extensions.tex_math(
      options.texMathDollars,
      options.texMathSingleBackslash,
      options.texMathDoubleBackslash)
    table.insert(extensions, tex_math_extension)
  end

```

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

```
  if options.footnotes or options.inlineFootnotes or
     options.notes or options.inlineNotes then
    local notes_extension = M.extensions.notes(
      options.footnotes or options.notes,
      options.inlineFootnotes or options.inlineNotes)
    table.insert(extensions, notes_extension)
  end

  if options.citations then
    local citations_extension = M.extensions.citations(options.citationNbsps)
    table.insert(extensions, citations_extension)
  end
```

```
8512
8513  if options.fancyLists then
8514    local fancy_lists_extension = M.extensions.fancy_lists()
8515    table.insert(extensions, fancy_lists_extension)
8516  end
```

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

```
8517  for _, user_extension_filename in ipairs(options.extensions) do
8518    local user_extension = (function(filename)
```

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

```
8519      local pathname = util.lookup_files(filename)
8520      local input_file = assert(io.open(pathname, "r"),
8521        [[Could not open user-defined syntax extension "]]
8522        .. pathname .. [[" for reading]])
8523      local input = assert(input_file:read("*a"))
8524      assert(input_file:close())
8525      local user_extension, err = load([[
8526        local sandbox = {}
8527        setmetatable(sandbox, {__index = _G})
8528        _ENV = sandbox
8529      ]] .. input)()
8530      assert(user_extension,
8531        [[Failed to compile user-defined syntax extension "]]
8532        .. pathname .. [[": ]] .. (err or [[]]))
```

Then, validate the user-defined syntax extension.

```
8533      assert(user_extension.api_version ~= nil,
8534        [[User-defined syntax extension "]] .. pathname
8535        .. [[" does not specify mandatory field "api_version"]])
8536      assert(type(user_extension.api_version) == "number",
8537        [[User-defined syntax extension "]] .. pathname
8538        .. [[" specifies field "api_version" of type "]]
8539        .. type(user_extension.api_version)
8540        .. [[" but "number" was expected]])
8541      assert(user_extension.api_version > 0
8542         and user_extension.api_version <= metadata.user_extension_api_version,
8543        [[User-defined syntax extension "]] .. pathname
8544        .. [[" uses syntax extension API version "]]
8545        .. user_extension.api_version .. [[ but markdown.lua ]]
8546        .. metadata.version .. [[ uses API version ]]
8547        .. metadata.user_extension_api_version
8548        .. [[, which is incompatible]])
8549
8550      assert(user_extension.grammar_version ~= nil,
8551        [[User-defined syntax extension "]] .. pathname
8552        .. [[" does not specify mandatory field "grammar_version"]])
8553      assert(type(user_extension.grammar_version) == "number",
```

```
        [[User-defined syntax extension "]] .. pathname
        .. [[" specifies field "grammar_version" of type "]]
        .. type(user_extension.grammar_version)
        .. [[" but "number" was expected]])
      assert(user_extension.grammar_version == metadata.grammar_version,
        [[User-defined syntax extension "]] .. pathname
        .. [[" uses grammar version "]] .. user_extension.grammar_version
        .. [[ but markdown.lua ]] .. metadata.version
        .. [[ uses grammar version ]] .. metadata.grammar_version
        .. [[, which is incompatible]])

      assert(user_extension.finalize_grammar ~= nil,
        [[User-defined syntax extension "]] .. pathname
        .. [[" does not specify mandatory "finalize_grammar" field]])
      assert(type(user_extension.finalize_grammar) == "function",
        [[User-defined syntax extension "]] .. pathname
        .. [[" 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
```

If Kpathsea has not been loaded before or if LuaTEX has not yet been initialized, configure Kpathsea on top of loading it.

```
local kpse
(function()
  local should_initialize = package.loaded.kpse == nil
                        or tex.initialize ~= nil
  local ran_ok
  ran_ok, kpse = pcall(require, "kpse")
  if ran_ok and should_initialize then
    kpse.set_program_name("luatex")
  end
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\markdownRendererHardLineBreakPrototype{\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}%
\def\markdownRendererDisplayMathPrototype#1{$$#1$$}%
\def\markdownRendererInlineMathPrototype#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
\def\markdownRendererSectionBeginPrototype{}%
\def\markdownRendererSectionEndPrototype{}%
```

3.2.2.1 Raw Attributes

In the raw block and inline raw span renderer prototypes, execute the content with TeX when the raw attribute is `tex`, display the content as markdown when the raw attribute is `md`, and ignore the content otherwise.

```
\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 ||
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_counter_tl
        }
          {
            \@@_get_option_value:nN
              { #1 }
              \l_tmpa_tl
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { #1~=~  \l_tmpa_tl   ,~ }
          }
        {
          \str_if_eq_p:VV
            \l_tmpa_tl
            \c_@@_option_type_clist_tl
        }
          {
            \@@_get_option_value:nN
```

```
              { #1 }
              \l_tmpa_tl
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { #1~=~\c_left_brace_str }
            \clist_map_inline:Vn
              \l_tmpa_tl
              {
                \tl_gput_right:Nx
                  \g_@@_formatted_lua_options_tl
                  { "##1" ,~ }
              }
            \tl_gput_right:Nx
              \g_@@_formatted_lua_options_tl
              { \c_right_brace_str ,~ }
          }
      }
      {
        \@@_get_option_value:nN
          { #1 }
          \l_tmpa_tl
        \tl_gput_right:Nx
          \g_@@_formatted_lua_options_tl
          { #1~=~ " \l_tmpa_tl " ,~ }
      }
  }
\cs_generate_variant:Nn
  \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_ε `\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.

```
    |markdownIfOption{frozenCache}{}{@
      |immediate|openout|markdownOutputFileStream@
        |markdownOptionInputTempFileName|relax@
      |markdownInfo{Buffering markdown input into the temporary @
        input file "|markdownOptionInputTempFileName" and scanning @
        for the closing token sequence "#1"}@
    }@
```

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.

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

```
  |def|markdownReadAndConvertStripPercentSign##1{@
    |markdownIfOption{stripPercentSigns}{@
      |if##1%@
        |expandafter|expandafter|expandafter@
          |markdownReadAndConvertProcessLine@
      |else@
        |expandafter|expandafter|expandafter@
          |markdownReadAndConvertProcessLine@
          |expandafter|expandafter|expandafter##1@
      |fi@
    }{@
      |expandafter@
        |markdownReadAndConvertProcessLine@
        |expandafter##1@
    }@
  }@
```

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.

```
  |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` (XETEX) 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()
```

If Kpathsea has not been loaded before or if LuaTEX has not yet been initialized, configure Kpathsea on top of loading it.

```
        local kpse
        (function()
          local should_initialize = package.loaded.kpse == nil
                                    or tex.initialize
          local ran_ok
          ran_ok, kpse = pcall(require, "kpse")
          if ran_ok and should_initialize then
            kpse.set_program_name("luatex")
          end
        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 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 input with backslash tokens with category code letter and then undo the replacement. This means that if any unbraced backslash tokens with category code letter exist in the input, they will be replaced with forward slashes. However, this should be extremely rare.

```
          \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 }
\ExplSyntaxOff
```

3.3.4 Token Renderer Prototypes

The following configuration should be considered placeholder. If the `plain` package option has been enabled (see Section 2.3.2.2), 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={
  hardLineBreak = {\\},
  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,[28] 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}
                \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}%
```

[28] 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}%
      \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}},
  displayMath = {\begin{displaymath}#1\end{displaymath}},
  inlineMath = {\begin{math}#1\end{math}},
```

```
  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[29], we will pass HTML elements to the output HTML document unchanged.

```
\@ifundefined{HCode}{}{
  \markdownSetup{
    rendererPrototypes = {
      inlineHtmlTag = {%
        \ifvmode
          \IgnorePar
          \EndP
        \fi
        \HCode{#1}%
      },
      inputBlockHtmlElement = {%
        \ifvmode
          \IgnorePar
        \fi
        \EndP
        \special{t4ht*<#1}%
        \par
        \ShowPar
      },
    },
  }
}
```

[29] See https://tug.org/tex4ht/.

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

Here is an implementation for hypertext links and relative references.

```
\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\markdownRendererHardLineBreak{\\}%
        \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 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}%
      },
    }
  }
}{}
```

3.3.4.11 Image Attributes

If the `linkAttributes` option is enabled, we will load the graphicx package. Furthermore, in image attribute contexts, we will make attributes in the form ⟨*key*⟩=⟨*value*⟩ set the corresponding keys of the graphicx package to the corresponding values.

```
\ExplSyntaxOn
\@@_if_option:nTF
  { linkAttributes }
  {
    \RequirePackage{graphicx}
    \markdownSetup{
      rendererPrototypes = {
        imageAttributeContextBegin = {
          \group_begin:
          \markdownSetup{
            rendererPrototypes = {
              attributeKeyValue = {
                \setkeys
                  { Gin }
                  { { ##1 } = { ##2 } }
              },
            },
          }
        },
        imageAttributeContextEnd = {
          \group_end:
        },
      },
    }
  }
  { }
\ExplSyntaxOff
```

3.3.4.12 Raw Attributes

In the raw block and inline raw span renderer prototypes, default to the plain TeX renderer prototypes, translating raw attribute `latex` to `tex`.

```
\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\markdownRendererHardLineBreakPrototype{\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\markdownRendererHardLineBreak{
    }%
    \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}}
\def\markdownRendererDisplayMathPrototype#1{\startformula#1\stopformula}%
\def\markdownRendererInlineMathPrototype#1{$#1$}%
```

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 Attributes

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